Date: February 2016

OBJECT MANAGEMENT GROUP®

Dependability Assurance Framework for Safety-Sensitive Consumer Devices™ (DAF™)

Version 1.0

OMG Document Number: formal/2016-02-01
Standard document URL: http://www.omg.org/spec/DAF
Machine Consumable Files:
Normative:
http://www.omg.org/spec/DAF/20150801/Argumentation.xsd
http://www.omg.org/spec/DAF/20150801/DependabilityAllocationArgument.xml
http://www.omg.org/specDAF/20150801/DynamicDependabilityAnalysisArgument.xml
http://www.omg.org/spec/DAF/20150801/Evidence.xsd
http://www.omg.org/spec/DAF/20150801/EvolutionaryDevelopmentargument.xml
http://www.omg.org/spec/DAF/20150801/IntegrationArgument.xml
http://www.omg.org/spec/DAF/20150801/ModificationArgument.xml
http://www.omg.org/spec/DAF/20150801/ProvenInUseArgument.xml
http://www.omg.org/spec/DAF/20150801/SACM.xsd
http://www.omg.org/spec/DAF/20150801/StandardsComplianceArgument.xml
http://www.omg.org/spec/DAF/20150801/StaticDependabilityAnalysisArgument.xml
http://www.omg.org/spec/DAF/20150801/SSCD_DCM.xmi
http://www.omg.org/spec/DAF/20150801/SSCD_DPM.xmi
Non-normative:
http://www.omg.org/spec/DAF/20150801/SSCD-DCM_Astah_UML.asta

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

LICENSES

PATENTS

GENERAL USE RESTRICTIONS

OMG'S ISSUE REPORTING PROCEDURE

Table of Contents

List of Figures

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language®); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel™); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org/*.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG specifications are available from the OMG website at:

http://www.omg.org/spec

Specifications are organized by the following categories:

Business Modeling Specifications

Middleware Specifications

- **CORBA/IIOP**
- **Data Distribution Services**
- **Specialized CORBA**

IDL/Language Mapping Specifications

Modeling and Metadata Specifications

- **UML, MOF, CWM, XMI**
- **UML Profile**

Modernization Specifications

Platform Independent Model (PIM), Platform Specific Model (PSM), Interface Specifications

- **CORBAServices**
- **CORBAFacilities**

OMG Domain Specifications

CORBA Embedded Intelligence Specifications

CORBA Security Specifications

Signal and Image Processing Specifications

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the link cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
109 Highland Avenue
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: pubs@omg.org

Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents, Report a Bug/Issue

1 Scope

The objective of this document is to provide a new system assurance methodology for the dependability argumentation for consumer devices, which is achieved by integrating conventional system assurance approaches such as risk analysis and assessments with a new way of approaching unique characteristics of consumer devices. The scope of this specification supports the objectives of the integration, and includes the dependability case for argumentation, as well as the dependability development process to be newly defined. The focus is to include the dependability argumentation particularly for consumer devices. In the future, it may be desirable to introduce additional argumentation methodology for other systems such as avionics or railways. However, they are outside of the scope for the current effort as the authors are not experts in other systems rather than consumer devices.

2 Conformance

This specification is intended to be an umbrella specification, which allows several existing specifications/standards either by OMG or other standardization bodies in a single framework.

For any specification/standard of a specific SSCD to be in conformance with the Dependability Conceptual Model requires that the conceptual model of the standard/specification shall include all models in DCM. It shall extend DCM specified in this specification to support new dependability, assurance, and process concepts for that specific SSCD as long as it will not cause any semantic inconsistency between DCM and the new conceptual model.

For conformance to Dependability Assurance Case, argumentation for SSCD dependability shall follow the argument structure specified in Clause 8. DAC shall conform to SACM.

For conformance to Dependability Process Model, the development process for SSCD shall follow the process defined in Clause 9.

Name of Model	**Clause Number**	**Requirement for conformance**
Dependability Conceptual Model	Clause 7	Each class defined in composed of Clause 7 shall be utilized to form a specification or a standard that defines a dependability conceptual model, dependability assurance case, and dependability process model for an application to design an SSCD.
Dependability Assurance Case	Clause 8	Argumentation for SSCD dependability shall use the DAC templates defined in Clause 8.
Dependability Process Model	Clause9	The development process for SSCD shall follow the process defined in Clause 9.

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

- Software Process Engineering Metamodel (SPEM), Version 2.0, OMG Document formal/2008-04-01, (http://www.omg.org/spec/SPEM/2.0/)

- Business Process Model and Notation (BPMN), Version 2.0.2 ,OMG Document formal/2013-12-09, (http://www.omg.org/spec/BPMN/2.0.2/)
- OMG Structured Assurance Case Metamodel, Version 1.0 , OMG Document formal/2013-02-01, (http://www.omg.org/spec/SACM/1.0/)
- OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.4.1, OMG Document formal/2011-08-05, (http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF/)

4 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

Safety-Sensitive Consumer device (SSCD) a category of industrial products used by consumer users, including automobiles, service robots, medical devices, and clinical systems, and smart houses. Preventing failures of the embedded software in SSCDs is going to be vital for consumer safety.

5 Symbols and Abbreviated Terms

DAC	Dependability Assurance Case
DAF	Dependability Assurance Framework
DCM	Dependability Conceptual Models
DPM	Dependability Process Models
SSCD	Safety-Sensitive Consumer device

6 Overview of the Specification

6.1 Introduction

Back in 2010, system quality caused serious problems in the automotive industry in the U.S. The electrical throttle control system was questionable, which may have caused unintended accelerations because of software bugs or system errors. The US government, NASA, and TOYOTA worked together to find out where the issue lay in the electrical throttle control system, disclosing all the documents and specifications that TOYOTA had for designing the system. The investigation results are open at the NASA website and they have confirmed that the system had no issue in the end. The reports concluded that the unintended acceleration might have been caused by floor mats which are, in general, piled up on top of previous ones as the owner of car often purchased new ones, which may have caused the accelerator pedal to become stuck between the floor mats.

Note – NASA is the National Aeronautics and Space Administration of USA, and TOYOTA is the Toyota Motor Corporation.

In such circumstances, can we really say that electronics systems are safe and that quality control procedures are in place and that the system validation process is robust? Will they continue to be as safe as they have been?

Taking the future of electronics systems into consideration, each electronics system is going to be one of the terminals of Internet of things and will be expected to play a significant role as a part of smart city. This consideration indicates that the safety of electronics systems cannot be achieved alone, but have to be achieved together with other electrical and electronic systems as a whole.

This series of questions sheds light on three aspects. One is that the customers' perspective regarding the quality for control systems has significantly changed and they would like to know how manufacturers ensure the quality of "invisible" control systems. Manufacturers have to act more proactively and take responsibility for the accountability for quality. Another is that a brand-new system assurance methodology will be required for System of Systems as a whole, though we will still have different standards in place for the industries for each category of electronics systems, respectively, such as automotive, medical devices, smart houses, and service robots. Also, the safety is not the only attribute to consider. The electronics systems have to achieve safety, reliability, availability, and even integrity at the same time. And the last is that use case scenarios for cars are quite difficult to capture, as the use case for the floor mats case suggests. However, the use case of the floor mat issue cannot be regarded as "out of scope" even with any difficulties. The "out of scope" is no longer "out of scope" and manufacturers will have to make the impossible possible and do whatever it takes in order to enhance the quality of their products.

In this specification, the methodology to resolve the issues above is specified and named as the Dependability Assurance Framework (DAF), with which a standard or a specification to assure the dependability of SSCD can be created. First, a new concept of the system assurance is specified, defining a new notion of consumer devices as well as dependability. Secondly, a parallel argumentation method with a Dependability-Case is introduced so that multi-standards as well as multi-attributes can be adequately addressed at the same time as part of argument structure. Thirdly, a rapid and iterative development process is defined, contrary to the V-process, in order to completely describe a common engineering process in the automotive industry.

6.2 Key Features

Dependability Assurance Framework (DAF) is a new approach for the system assurance of Safety-Sensitive Consumer Devices (SSCD) which can provide a comprehensive methodology for the argumentation for SSCD. Historically, each attribute of "Dependability" such as safety, security, integrity, and so forth, have been separately discussed in different way of assurance framework because of different existing standards. Now, given the fact that SSCD is a system to implement certain function, which aggregates systems to individually implement certain different functions, DAF can provide a model based system assurance methodology for each system and the entire system, also can provide a model based dependability assurance methodology to construct argumentation for each attribute of "Dependability" and the entire "Dependability," simultaneously.

DAF, however, cannot provide each single method to build up the argumentation for "Dependability" because it would be too huge to specify everything for aspects of system assurance in this specification, but can provide developers of SSCD with how to build up their own dependability assurance standard, aggregating knowledge and experiences from existing standards, in terms of what kind of technical terms to be incorporated into, what kind of process to develop argumentation in, and what kind of aspects to take into account for argumentation.

DAF consists of: Dependability Conceptual Model (DCM) that defines objects and relations which are required for the SSCD dependability argumentation; Dependability Process model (DPM) that defines a differential development and a rapid iterative development process as a part of the conventional V-process; and Dependability Assurance Case (DAC) which employs the SACM for SSCD dependability argumentation as well as the notion of Proven In Use with concrete usage.

DAF with DCM, DPM, and DAC can provide an efficient method for the argumentation regardless of any properties of system assurance such as safety, reliability, availability, and so forth. Additionally, DAF is expected to work as a supplement to existing standards such as ISO26262, where any argumentations for both existing standards and whatever standards are needed for specific systems.

This specification is an abstraction of existing standards related to dependability assurance such as ISO26262. This specification can be referred to in terms of what aspects are required to consider for enhancing the dependability assurance of SSCDs.

6.2.1 Key Capabilities of DAF

DAF provides the following capabilities to develop the dependability argumentation for consumer devices.

a) Dependability assurance methodology for Safety-Sensitive Consumer Devices: The DAF can provide a big picture on how to establish the dependability assurance for SSCDs. With this methodology and the conceptual models, developers can understand what kind of aspects they need to consider for full support of the dependability construction. Also, the dependability argumentations for SSCDs can be discussed in parallel, evaluating the consistency among them. This methodology also provides the parallel discussions for each attribute of the dependability.

b) Template for dependability argumentation: A DAC template provides a way of argumentation particularly for SSCDs which can guide developers to construct the dependability argumentation when developing their own products. The template contains the notion of compositional assurance in which developers can discuss the dependability assurance component by component based on the structured system of their products in line with the system engineering approach. With this template, developers can not only make their argumentation for the dependability of their products clearer but also make the scope of their discussion clearer. In addition, the template can provide a way of reusability of the assurance for the development efficiency.

c) Dependability assurance process: DPM provides a process to develop the dependability assurance which can work together with the conventional system development process. With this process, developers can create the dependability case for their products while developing their own products at the same time.

6.2.2 Procedure

This clause describes the procedure to apply this specification for the dependability assurance for SSCDs.

a) Create your own DCM to define each dependability concept for your product, referring to Clause 7.
b) Create your own DPM to define your own dependability process for your product, referring to Clause 9.
c) Create your own DAC for the dependability argumentation for your product, referring to Clause 8.

The relationship between the three models are illustrated in the following package in Figure 6.1, where Dependability Conceptual Model are referenced by the other two models (Dependability Process Model, Dependability Assurance Case Template) while Dependability Assurance Case Template depends on Dependability Process Model to come up with argumentation for SSCD.

Defining the three models is a minimum set for creating a specification of Dependability Assurance for a SSCD. To specify a complete set of the specification that you are going to create shall reference existing standards in terms of how the concepts are utilized to build up argumentation for an SSCD according to the dependability process.

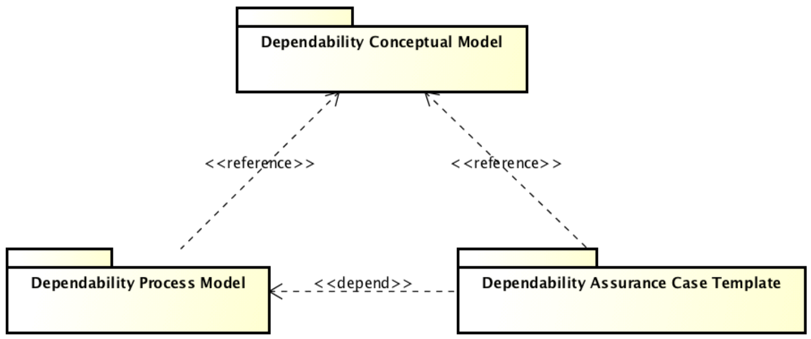

Dependability Conceptual Model
<<reference>>
<<reference>>
Dependability Process Model
<<depend>>
Dependability Assurance Case Template

Figure 6.1 - Dependability Assurance Framework

6.2.3 How to Read this Specification

If you want to understand the background of this specification, Clause 6.1 provides the background of this specification for readers to understand a big picture of the specification. The rest of this document contains the technical content of this specification, with which users can create their own dependability assurance for their own SSCD.

Clause 7 provides the Dependability Conceptual Model with which developers are going to develop their own dependability assurance for their own products.

Clause 8 provides the Dependability Assurance Case template with which developers are going to develop their own dependability argumentation according to the template of argumentation particularly for SSCDs.

Clause 9 provides the Dependability Process Model with which developers are going to develop their own dependability assurance while engineering their own products.

This page intentionally left blank.

7 Dependability Conceptual Model (DCM)

7.1 General

This clause specifies the semantic model of Dependability Conceptual Model (DCM). The main aim of the DCM is to lay the foundations of this specification in which all the terminology/vocabulary used in this specification will be presented as semantic models in UML class diagrams followed by narratives which specify the terms, definitions, and abbreviated terms in English. The semantic models in this specification are not meant to be implemented for any purposes but to support assurance concept/activities/processes to ensure dependability of Safety Sensitive Consumer Devices (SSCDs). It must be emphasized that this version of the DCM only constitutes the minimum core of dependability assurance concepts, which supports other parts, i.e., Dependability Process Model (DPM) and Dependability Assurance Case (DAC) in this version of SSCDs specification.

As the SSCD covers a broad product category including automobiles, service robots, and smart houses and so on, this specification is intended to be an umbrella specification, which allows several existing specifications/standards either by OMG or other standardization bodies in a single framework. We took special care as to how this can be realized. The main idea of realizing this is to provide some room to plug in other specifications/standards to our semantic models without any interference in terms of the underlying semantics with this core specification. In order to do so, out semantic models are provided at an abstract level where several specifications/standards can be accommodated and harmonized.

7.1.1 DCM Top-level Package

The Dependability Conceptual Model package in Figure 7.1 is specified as the top-level package of all the other packages for DCM.

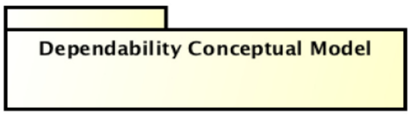


Figure 7.1 - Dependability Conceptual Model

The whole structure of the DCM is grouped together under the UML package, which consists of the following sub-packages:

1. Architectural Concept
2. Dependability Assurance Concept
3. Requirement Concept
4. Dependability Process Concept
5. System Environment Concept

The Architectural Concept package aims to specify what system concepts are used in this specification. The basic notions in this package are based on conventional systems engineering concepts. The Dependability Assurance Concept provides the conceptual model supporting dependability assurance in this specification. This is the core part of the DCM which provides the basis for several substantial sub-packages. The Dependability Process Concept covers the concepts used in the Dependability Process Model (DPM) in this specification. This part should be read with the process model in BPMN in Clause 9. The Requirement Concept package specifies what kinds of requirements are dealt with in this specification. The System Environment Concept package specifies how the system boundary is set in this specification.

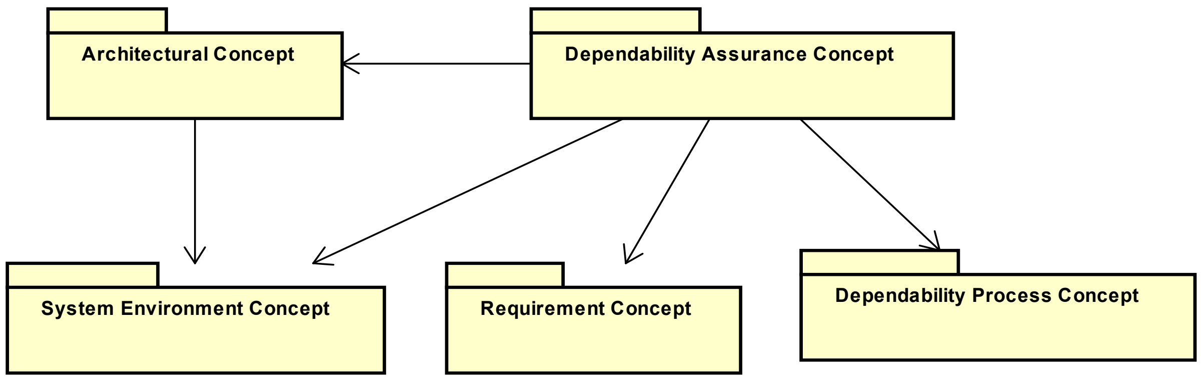

Architectural Concept
Dependability Assurance Concept
System Environment Concept
Requirement Concept
Dependability Process Concept

Figure 7.2 - Dependability Conceptual Model package

The UML (Unified Modeling Language) class diagrams are used throughout in this specification. Concepts in the Dependability Conceptual Model are modularized topically to separate concerns and to facilitate understanding by developers. These packages are interrelated, but independently specified in this specification. A dashed line between packages represents dependency between them. The detailed explanation of each package can be found in the corresponding clauses that follow.

7.2 Architectural Concept

This Architectural Concept package provides the overall architecture and its elements of SSCD. Each element is positioned sequentially in four levels, the System of Systems level, the System level (including Subsystem level), the Component level, and the Implementation level. The System level (including Subsystem level), the Component level, and the Implementation level identify the Development Category that is New Development or Modification.

The System of Systems level is not mandatory, and it is used as the uppermost layer when several systems are combined and integrated.

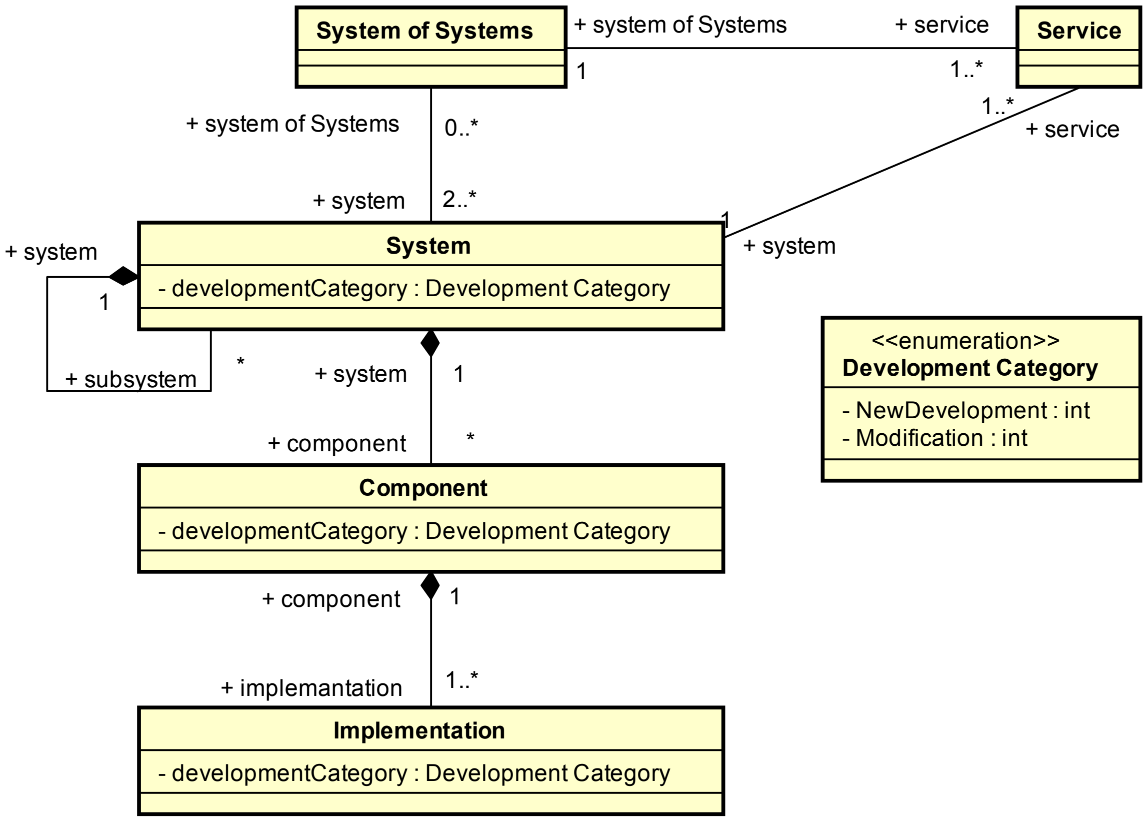


Figure 7.3 - Architectural Concept

This package describes what is meant by Architecture in this specification. This specification faithfully follows the standard notion of systems engineering (From "ISO/IEC15288: 2008"). It is broken down into detailed parts gradually from an abstract concept.

7.2.1 System of Systems

Description

It is possible that System does not belong to any System of Systems. Also, it is possible that System belongs to System of Systems.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

system: System[2..*]

System of Systems is composed of two or more systems.

service: Service[1..*]

System of Systems can provide one or more services.

7.2.2 System

Description

A system is a collection of subsystems or components that are organized for a common purpose. If it is complex, it is composed of subsystems. If subsystems are necessary, the System is composed of subsystems, and the subsystem is composed of components.

Generalizations

No additional generalizations

Attributes

developmentCategory: DevelopmentCategory

It is distinguished by new development or modification.

Associations

subsystem: System[*]

The System (including subsystem) is composed of zero or more subsystems.

component: Component[*]

The System (including subsystem) is composed of zero or more components.

service: Service[*]

The System (not including subsystem) can provide one or more services.

7.2.3 Component

Description

Component is a very important part of System or Subsystem. It consists of Implementations that are Hardware and/or Software. It consists of Implementations that are more than one Hardware and/or Software.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

implementation: Implementation[1..*]

The Component is composed of one or more implementations.

7.2.4 Implementation

Description

Implementation is the smallest unit in Architecture. Component is decomposed into Implementation(s) that can contain Hardware or Software. It is an implementation of hardware or Software composing the Component.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.2.5 Service

Description

A System of Systems or a System provides one or more Services. A Service provides value, (satisfying a goal for example) or an effect to an Actor.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.2.6 Development Category

Description

The Development Category provides the distinction between new development of a system and modification of an existing system. This notion defines an enumeration data type which has two separate values.

Generalizations

No additional generalizations

Attributes

NewDevelopment: int

Designate a system to be newly developed.

Modification: int

An existing system to be modified respectively.

Associations

No additional associations

Note – <<enumeration>> is used to indicate that this is a class for enumeration data type.

7.3 Dependability Assurance Concept

This package accommodates all the basic notions of system assurance in this specification. The concepts relevant to ensuring dependability of target systems are specified in terms of a number of sub-packages, as shown in Figure 7.4, providing coverage for the level of complexity and breadth required, as follows:

1. Dependability Assurance Case Concept
2. Dependability Concept
3. Dependability Assurance Level
4. Error Model
5. Assessment
6. Proven In Use

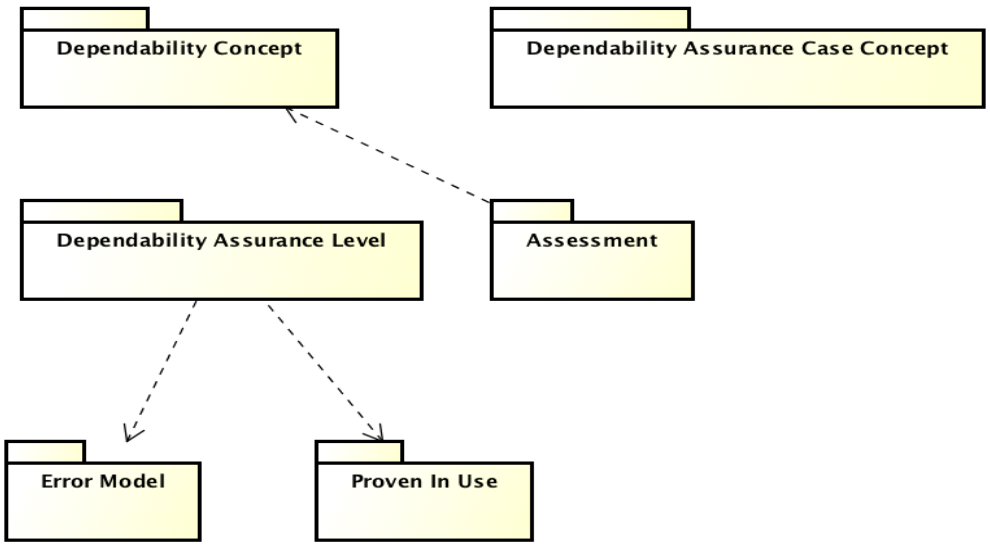

Dependability Concept
Dependability Assurance Case Concept
Dependability Assurance Level
Assessment
Error Model
Proven In Use

Figure 7.4 - Dependability Assurance Concept package

The Dependability Assurance Case Concept package specifies how dependability assurance cases are addressed in this specification. The Dependability Assurance Level package defines the criteria for threat assessment in this specification. The Error Model package specifies basic notions surrounding the classic notion of errors such as faults and failures, and how they are incorporated into dependability. The Assessment package deals with how assessment is done in this specification. Finally the Proven In Use package specifies how a modified system is assessed.

7.3.1 Dependability Assurance Case Concept

Description

This package contains the Dependability Assurance Case Concept. A Dependability Assurance Case Concept consists of a Dependability Claim, Dependability Assurance Case, and Evidence.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

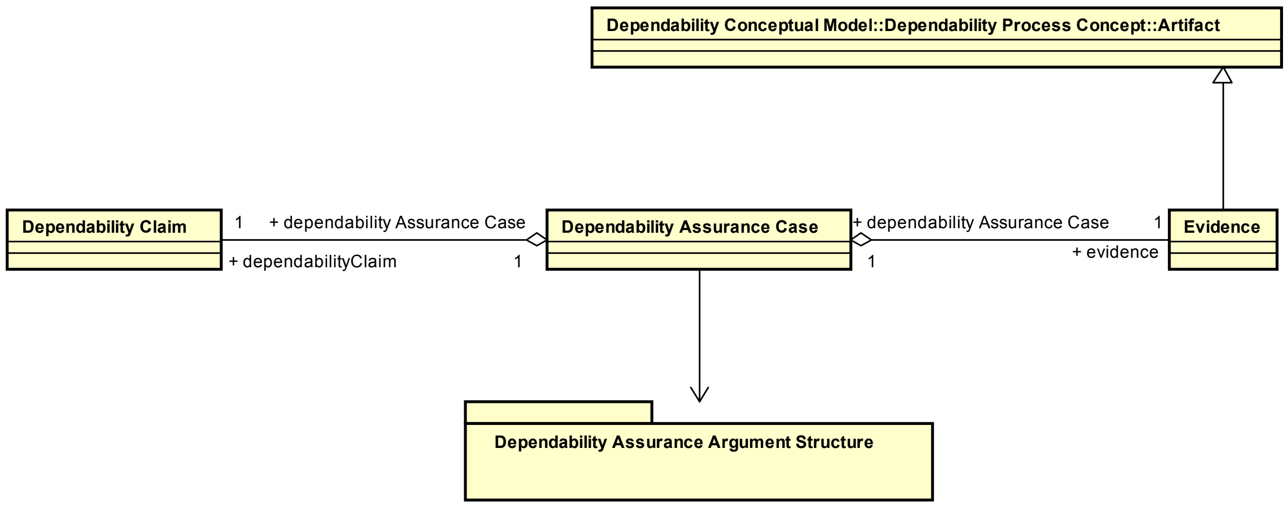

Dependability Conceptual Model::Dependability Process Concept::Artifact
Dependability Claim
1
+ dependability Assurance Case
+ dependabilityClaim
1
Dependability Assurance Case
+ dependability Assurance Case
1
1
+ evidence
Evidence
Dependability Assurance Argument Structure

Figure 7.5 - Dependability Assurance Case Concept

7.3.1.1 Dependability Claim

Description

The Dependability Claim is a proposition about the dependability of the target system or system of systems, which is to be assured.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.3.1.2 Evidence

Description

Evidence is the basis of the argument for the dependability claim.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.3.1.3 Dependability Assurance Case

Description

A structured argument, supported by a body of evidence that provides a compelling, comprehensible, and valid case that a system of system, or a system is dependable for a given application in a given environment.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

dependabilityClaim: Dependability Claim[1]

    evidence: Evidence[1]

7.3.1.4 Dependability Assurance Argument Structure

Description

This package contains Dependability Assurance Argument classes, which represent argument structures for assuring the dependability of the target safety sensitive consumer devices.

Generalizations

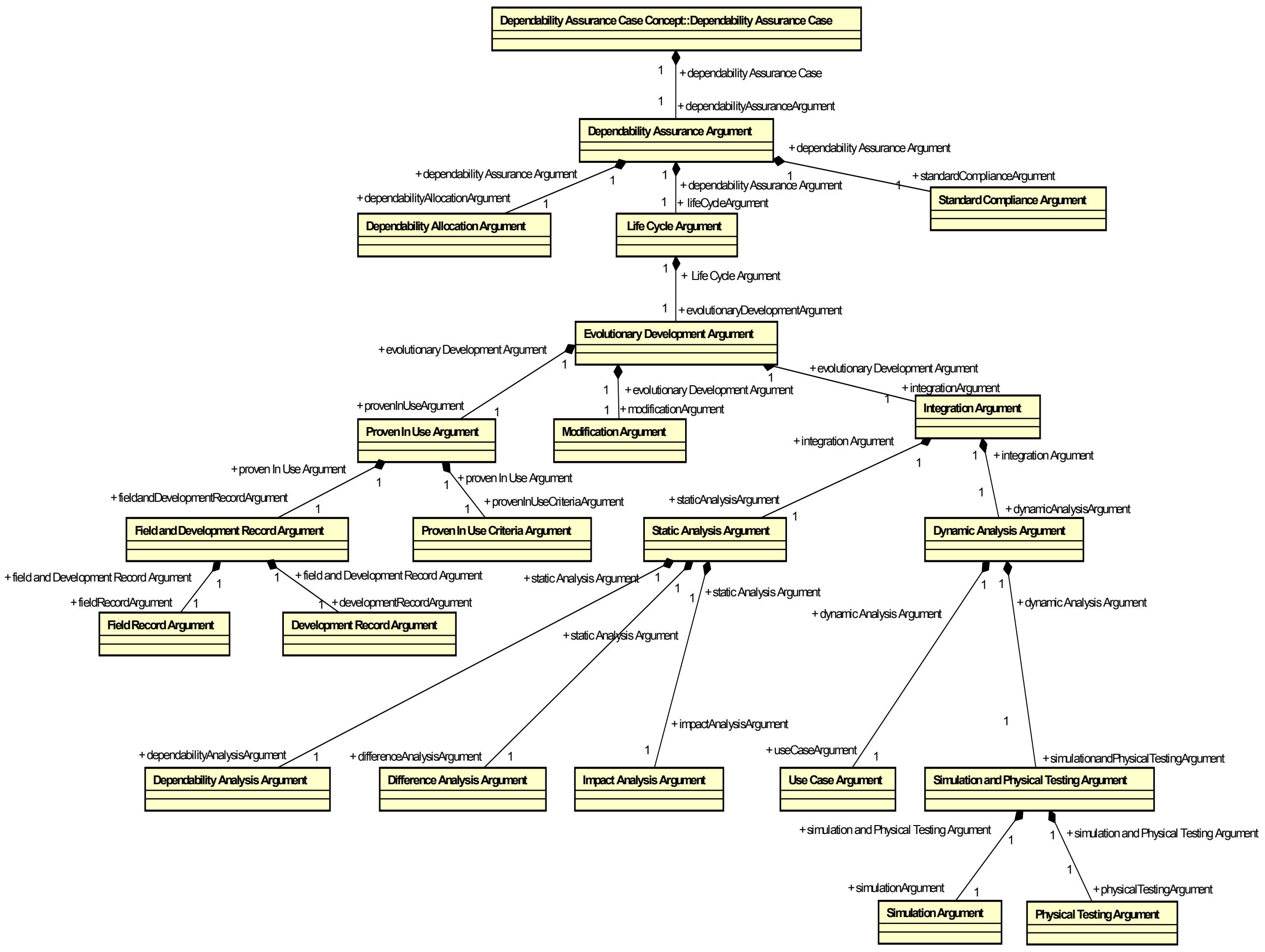
No additional generalizations

Attributes

No additional attributes

Associations

No additional associations



Dependability Assurance Case Concept::Dependability Assurance Case
Dependability Assurance Argument
Dependability Allocation Argument
Life Cycle Argument
Standard Compliance Argument
Evolutionary Development Argument
Proven In Use Argument
Modification Argument
Integration Argument
Field and Development Record Argument
Proven In Use Criteria Argument
Static Analysis Argument
Dynamic Analysis Argument
Field Record Argument
Development Record Argument
Dependability Analysis Argument
Difference Analysis Argument
Impact Analysis Argument
Use Case Argument
Simulation and Physical Testing Argument
Simulation Argument
Physical Testing Argument

Figure 7.6 - Dependability Assurance Argument Structure

7.3.1.5 Dependability Assurance Argument

Description

The Dependability Assurance Argument class represents the argument structure for assuring dependability of the target architecture. The argument structure consists of three sub structures: Dependability Allocation Argument, Standard Compliance Argument, and Lifecycle Argument.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

dependabilityAllocationArgument: Dependability Allocation Argument[1]

lifeCycleArgument: Life Cycle Argument[1]

standardComplianceArgument: Standard Compliance Argument

7.3.1.6 Dependability Allocation Arguments

Description

The Dependability Allocation Argument Class represents the argument structure for assuring the adequacy of dependability allocation to each sub-architecture(s) of the target architecture.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.3.1.7 Standard Compliance Argument

Description

The Standard Compliance Argument class represents the argument structure for assuring that the target architecture complies with other standards which are not covered by this specification.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.3.1.8 Lifecycle Argument

Description

The Lifecycle Argument class represents an argument structure for assuring that the target architecture has been developed in a lifecycle complying with the DPM. It consists of an argument structure for assuring that the development of the target architecture has adequately been developed. Safety-sensitive consumer devices should be developed evolutionally. Therefore, the structure consists of an Evolutionary Development Argument structure.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

evolutionaryDevelopmentArgument: Evolutionary Development Argument[1]

7.3.1.9 Evolutionary Development Argument

Description

The Evolutionary Development Argument class represents the argument structure for assuring that the development of the target architecture has adequately been developed over the generations. Evolutionary Development Argument class consists of a Proven In Use Argument, a Modification Argument, and an Integration Argument.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

provenInUseArgument: Proven In Use Argument[1]

modificationArgument: Modification Argument[1]

integrationArgument: Integration Argument[1]

nextGenerationArgument: Next Generation Argument[1]

7.3.1.10 Modification Argument

Description

The Modification Argument class represents the argument structure for assuring that the modified and impacted (by the modification) parts of the target architecture have adequately modified. Modification Argument consists of zero or more development argument structures for the modified and impacted parts.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.3.1.11 Proven In Use Argument

Description

The Proven In Use Argument class represents the argument structure for assuring that the unchanged and un-impacted (by the modification) parts of the target architecture adequately satisfy the allocated dependability attributes by proven in use. The structure consists of the Proven In Use Criteria Argument and Field and Development Record Argument.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

fieldandDevelopmentRecordArgument: Field and Development Record Argument[1]

provenInUseCriteriaArgument: Proven In Use Criteria Argument[1]

7.3.1.12 Proven In Use Criteria Argument

Description

The Proven In Use Criteria Argument class represents the argument structure that the unchanged and un-impacted (by the modification) parts satisfy the criteria to be assured by the proven in use argument.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.3.1.13 Field and Development Record Argument

Description

The Field and Development Record Argument class represents the argument structure for assuring that the unchanged and un-impacted (by the modification) parts of the target architecture adequately satisfy allocated dependability attributes by the proven in use argument using a field and a development record. The structure consists of a Field Record Argument and a Development Record Argument.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

fieldRecordArgument: Field Record Argument[1]

developmentRecordArgument: Development Record Argument[1]

7.3.1.14 Field Record Argument

Description

The Field Record Argument class represents the argument structure for assuring that the field record of the unchanged and un-impacted (by the modification) parts of the architectural constituent are adequate enough for the Proven In Use argument in the operation.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.3.1.15 Development Record Argument

Description

The Development Record Argument class represents the argument structure for assuring that the development record of the unchanged and un-impacted (by the modification) parts of the architectures are adequate for the Proven In Use Argument in the development.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.3.1.16 Integration Argument

Description

The Integration Argument class represents the argument structure for assuring that the integrated architecture adequately satisfies the allocated dependability attributes. The structure consists of a Static Analysis Argument and a Dynamic Analysis Argument structures.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

staticAnalysisArgument: Static Analysis Argument[1]

dynamicAnalysisArgument: Dynamic Analysis Argument[1]

7.3.1.17 Static Analysis Argument

Description

The Static Analysis Argument class represents the argument structure for assuring that the static analysis for the integrated architecture has been adequately done. The structure consists of Dependability Analysis Argument, Difference Argument, and Impact Analysis Argument.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

dependabilityAnalysisArgument: Dependability Analysis Argument [1]

differenceAnalysisArgument: Difference Analysis Argument[1]

impactAnalysisArgument: Impact Analysis Argument[1]

7.3.1.18 Dependability Analysis Argument

Description

The Dependability Analysis Argument class represents the argument structure for assuring that the dependability analysis for the identified threats in the integrated architecture has been adequately done.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.3.1.19 Difference Analysis Argument

Description

The Difference Analysis Argument class represents the argument structure for assuring that the difference analysis for the existing and to be developed architectural constituent has been adequately done.

Generalizations

No additional Generalizations

Attributes

No additional attributes

Associations

No additional associations

7.3.1.20 Impact Analysis Argument

Description

The Impact Analysis Argument class represents the argument structure for assuring that the impact analysis of the modification in the integrated architecture has been adequately done.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.3.1.21 Dynamic Analysis Argument

Description

The Dynamic Analysis Argument class represents the argument structure for assuring that the dynamic analysis for the integrated architectures has been adequately done. The structure consists of Use Case and Simulation and Physical Testing Argument structures.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

useCaseArgument: Use Case Argument[1]

simulationandPhysicalTestingArgument: Simulation and Physical Testing Argument[1]

7.3.1.22 Use Case Argument

Description

The Use Case Argument class represents the argument structure for assuring that the use cases for dynamic analysis have been adequately identified, and do not contain redundant use cases.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.3.1.23 Simulation and Physical Testing Argument

Description

The Simulation and Physical Testing Argument class represents the argument structure for assuring that the simulation and physical testing have been adequately done for the integrated architecture. The structure consists of a Simulation Argument and Physical Testing Argument.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

simulationArgument: Simulation Argument[1]

physicalTestingArgument: Physical Testing Argument[1]

Simulation Argument and Physical Testing Argument classes are associated.

7.3.1.24 Simulation Argument

Description

The Simulation Argument class represents the argument structure for assuring that the simulation has been adequately done for the integrated architectural constituent.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.3.1.25 Physical Testing Argument

The Physical Testing Argument class represents the argument structure for assuring that physical testing has been adequately done for the integrated architectural constituent.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.3.2 Dependability Concept

As given in Figure 7.7, the Dependability Concept package specifies the notion of dependability in this specification. In the broadest sense, dependability is defined as a system state which enables the system to provide continuous, uninterrupted provisioning of services. Compared with other system attributes such as safety and reliability, which have a long tradition and their definitions being well understood, there has been far less consensus around the notion of dependability to date. According to the seminal paper by Laprie, et. al [2], dependability is defined as an umbrella concept which includes various system attributes such as availability, reliability, safety, integrity, and maintainability. In this specification, we neither advocate a new notion of nor adopt any existing notion of dependability. Rather, a framework for specific dependability for a specific domain, product-line, product, or service is provided. The main reason for this design decision is that SSCDs cover a wide range of industrial products that may have different notion of dependability. For this reason, this specification does not force any subordinate specifications/standards to comply with a single notion of dependability.

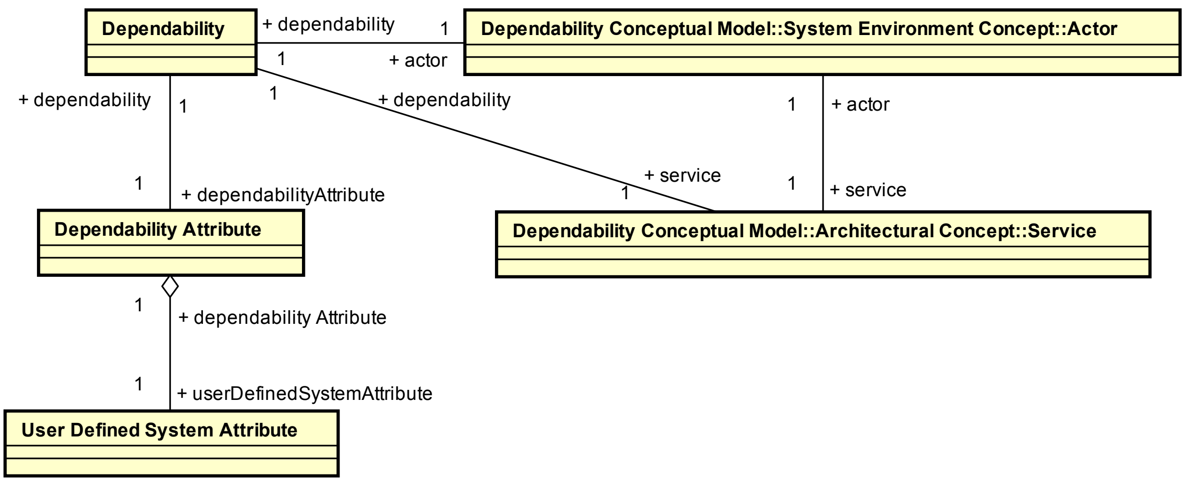

Dependability
+ dependability
1
Dependability Conceptual Model::System Environment Concept::Actor
1
+ actor
+ dependability
1
1
+ dependability
1
+ actor
1
+ dependabilityAttribute
+ service
1
1
+ service
Dependability Attribute
Dependability Conceptual Model::Architectural Concept::Service
1
+ dependability Attribute
1
+ userDefinedSystemAttribute
User Defined System Attribute

Figure 7.7 - Dependability Concept package

7.3.2.1 Dependability

Description

Dependability is the composite system attribute which consists of various kinds of system attributes.

Dependability ensures that services required by an actor are continuously provided.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

service: Service[1]

Specifies that dependability ensures that services are continuously provided.

dependabilityAttribute: Dependability Attribute[1]

Specifies that the Dependability concept may have several Dependability attributes.

Note – Actor and Service are provided by the System Environment Concept package and the Architectural Concept package respectively.

7.3.2.2 Dependability Attribute

Description

Dependability Attribute is an anchor point where any specific notion of dependability of a particular specification/ standard under this SSCD specification could be defined particularly specifying system attributes by which the dependability in that specification/standard is defined.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

userDefinedSystemAttribute: User Defined System Attribute[1]

Specifies that User Defined System Attribute is a part of Dependability attribute.

Note – We will show a sample figure in order to illustrate how this concept may be used. This class can accommodate Laprie's [2] definition of dependability as shown in Figure 7.8 where five system attributes; Availability, Reliability Safety, Maintainability, and Integrity are defined as essential constituents of the dependability attribute.

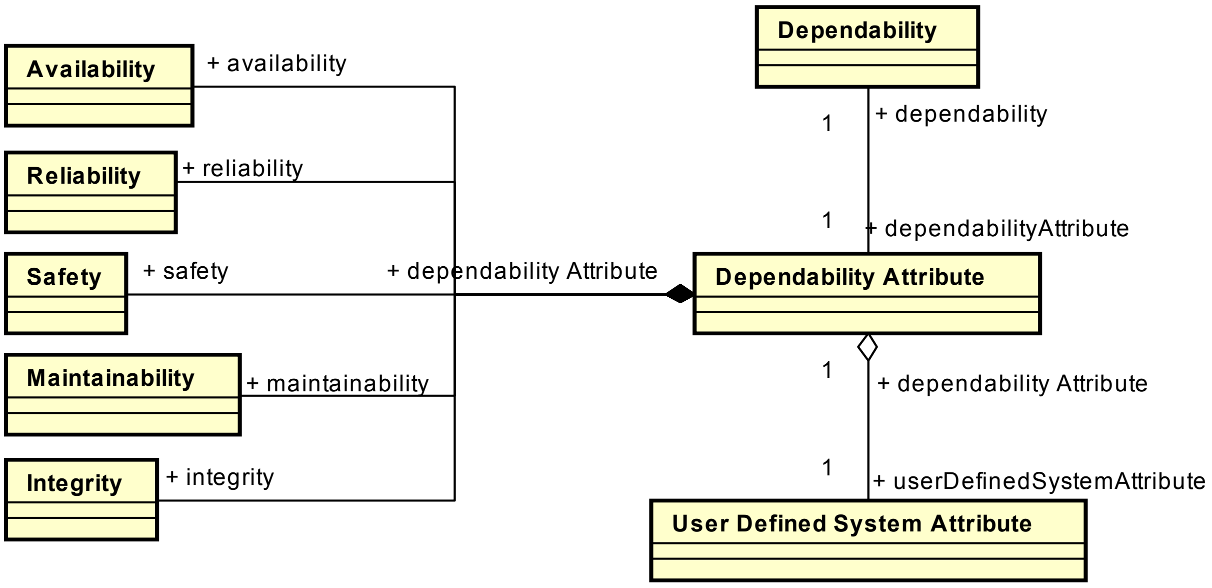


Figure 7.8 - Sample extension of Dependability Attribute (Informative)

7.3.2.3 User Defined System Attribute

Description

This concept specifies that any user may define system attributes which consist of the dependability attributes in a particular specification/standard.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

Note – User Defined System Attribute is a part of the Dependability Attribute.

7.3.3 Dependability Assurance Level

The package in Figure 7.9 accommodates all the notions as to how threat is assessed in the specification.

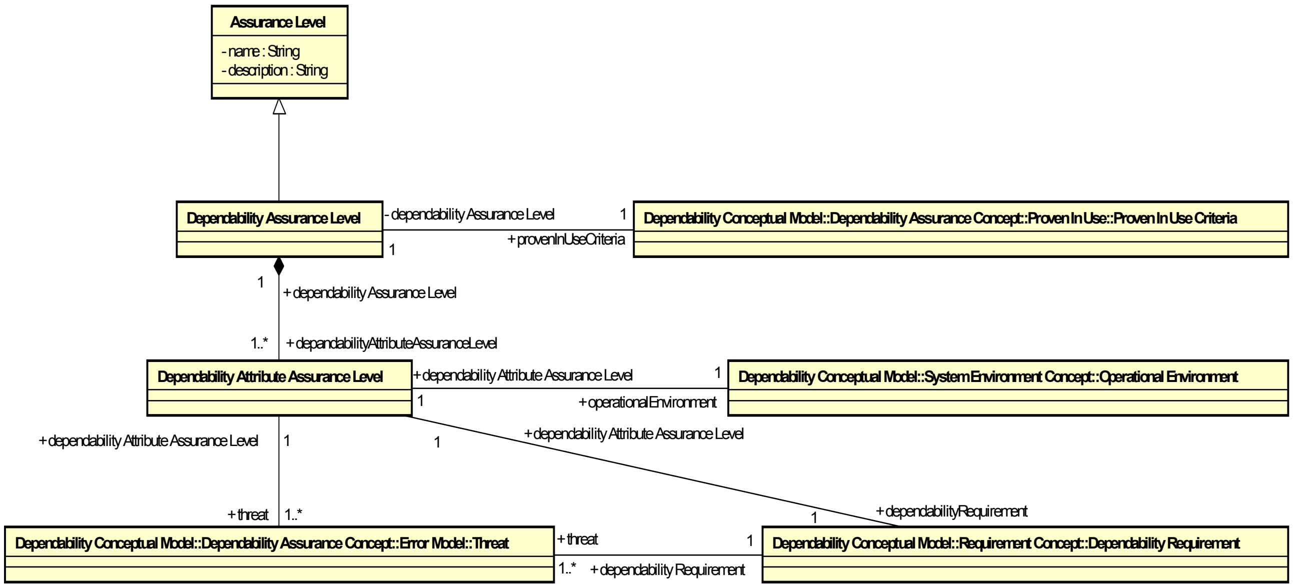

Assurance Level
- name : String
- description : String
Dependability Assurance Level
- dependability Assurance Level
1
1
Dependability Conceptual Model::Dependability Assurance Concept::Proven In Use::Proven In Use Criteria
+ provenInUseCriteria
1
+ dependability Assurance Level
1..*
+ depandabilityAttributeAssuranceLevel
Dependability Attribute Assurance Level
+ dependability Attribute Assurance Level
1
1
Dependability Conceptual Model::System Environment Concept::Operational Environment
+ operationalEnvironment
+ dependability Attribute Assurance Level
1
1
+ dependability Attribute Assurance Level
+ threat
1..*
1
+ dependabilityRequirement
Dependability Conceptual Model::Dependability Assurance Concept::Error Model::Threat
+ threat
1..*
+ dependability Requirement
1
Dependability Conceptual Model::Requirement Concept::Dependability Requirement

Figure 7.9 - Dependability Assurance Level Package

This package accommodates concepts associated with the assurance levels for threat assessment. The diagram above also specifies relationships among some relevant notions in separate packages such as Threat in the Error Model package and Operational Environment in the System Environment Concept, Dependability Requirements in the Requirement Concept, and Proven In Use Criteria in the Proven In Use Package.

Risk assessment in safety functional standards in several industrial domains is based on the integrity level. For instance, risk assessment in IEC 61508 [3] for electrical/electronic/programmable devices is achieved using SIL (Safety Integrity Level) and that in ISO 26262 [1] for automotive uses ASIL (Automotive Safety Integrity Level). In the security domain, Common Criteria (ISO/IEC 15408 [4]) uses EAL (Evaluation Assurance Level) and a security standard for industrial automation and control systems uses SAL (Security Assurance Level) [5]. It must be noted that the term integrity in the integrity levels in some of those standards have nothing to do with the system attribute integrity. Historically the term is used for the metrics for assessing risks involved in those industrial domains. We did not follow this tradition and use the term Assurance Level instead.

Dependability Assurance Level in Figure 7.9 is the top concept for assessing threats in SSCDs (Please refer to the Error Model Package in Figure 7.12 for the exact meaning of the Threat). Dependability Attribute Assurance Level corresponds to any particular system attribute which composes the dependability concept of a target system. For instance, safety assurance level in functional safety standards is a sub-notion of Dependability Attribute Assurance Level. The basic norm behind this composition is that a target system is not assessed by the single dependability assurance level, but assessed by a combination of assurance levels of each system attribute which consists of the notion of dependability.

The Dependability Attribute Assurance Level is allocated to the Dependability Requirement (in the Requirement Concept package), which mitigates a Threat (in the Error Model package) together with an Operational Environment (in the System Environment Concept package). Risk is assessed by combinations of Threat and Operational Environment.

7.3.3.1 Assurance Level

Description

This concept is the top-level concept, which accommodates all the relevant assurance levels of a particular system attribute.

Generalizations

No additional generalizations

Attributes

name: String

Specifies the name of the Assurance Level.

description: String

Specifies the description of the Assurance Level.

Associations

No additional associations

7.3.3.2 Dependability Assurance Level

Description

Dependability Assurance Level specifies a particular dependability assurance level.

Generalizations

Assurance Level on the previous subsection.

Attributes

No additional attributes

Associations

depandabilityAttributeAssuranceLevel: Dependability Assurance Level[1]

Specifies that Dependability Attribute Assurance Level is a part of Dependability Assurance Level.

provenInUseCriteria: Proven In Use Criteria[1]

Specifies that Dependability Assurance Level is assigned to Proven in Use Criterion.

7.3.3.3 Dependability Attribute Assurance Level

Description

As was previously mentioned, dependability is an umbrella concept which consists of several system attributes such as safety, reliability and so on. Therefore this notion is provided in order to accommodate an assurance level for each system attribute which consists of the notion of dependability.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

threat: Threat[1..*]

Specifies that Dependability Attribute Assurance Level is assessed by Threat.

dependabilityRequirement: Dependability Requirement [1]

Specifies that Dependability Attribute Assurance Level is allocated to Dependability Requirement.

operationalEnvironment: Operational Environment[1]

Specifies that Dependability Attribute Assurance Level is assessed by Operational Environment.

Note – First of all, we will demonstrate how new assurance levels for dependability can be defined, which follows the definition by Laprie [2], and introduce an assurance level for each dependability attribute.

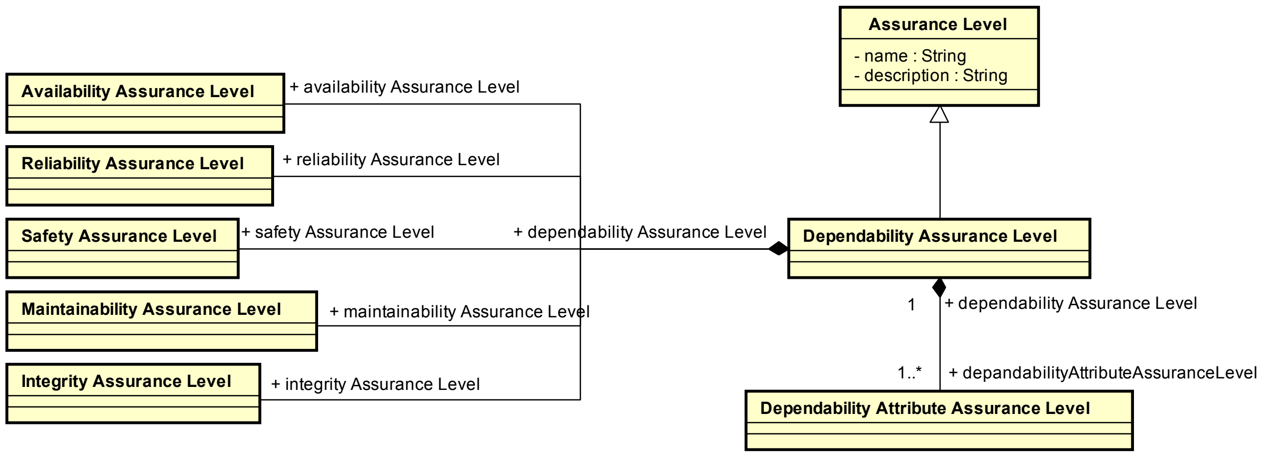

Assurance Level
- name : String
- description : String
Availability Assurance Level
+ availability Assurance Level
Reliability Assurance Level
+ reliability Assurance Level
Safety Assurance Level
+ safety Assurance Level
+ dependability Assurance Level
Dependability Assurance Level
Maintainability Assurance Level
+ maintainability Assurance Level
1
+ dependability Assurance Level
Integrity Assurance Level
+ integrity Assurance Level
1..*
+ depandabilityAttributeAssuranceLevel
Dependability Attribute Assurance Level

Figure 7.10 - Sample of Dependability Attribute Assurance Levels (Informative)

Calculation of each assurance level depends on the specifics of the domain and product for a particular SSCD. For example, for some systems, MTBF (Mean Time Between Failures) and MTTR (Mean Time To Repair) may be applicable at the Availability Assurance Level. For those same or other systems, you might want to use SIL in IEC 61508 for the Safety Assurance Level and so on. Domain- and product- specific requirements should be used to refine the definition of dependability and related assurance level for a given implementation.

As an additional illustration, functional safety standards such as IEC 61508 can be supported using the DCM in Figure 7.11. IEC 61508 defines SIL (Safety Integrity Level) to assess the potential risk of electrical and/or electronic devices based on the probability of failure and the severity of harm. One way of incorporating SIL into our specification is to create Safety Integrity Level (SIL) class and to place it under the Dependability Attribute Assurance Level as a sub-class. As Dependability Attribute Assurance Level is allocated to a dependability requirement, so is SIL to a safety requirement.

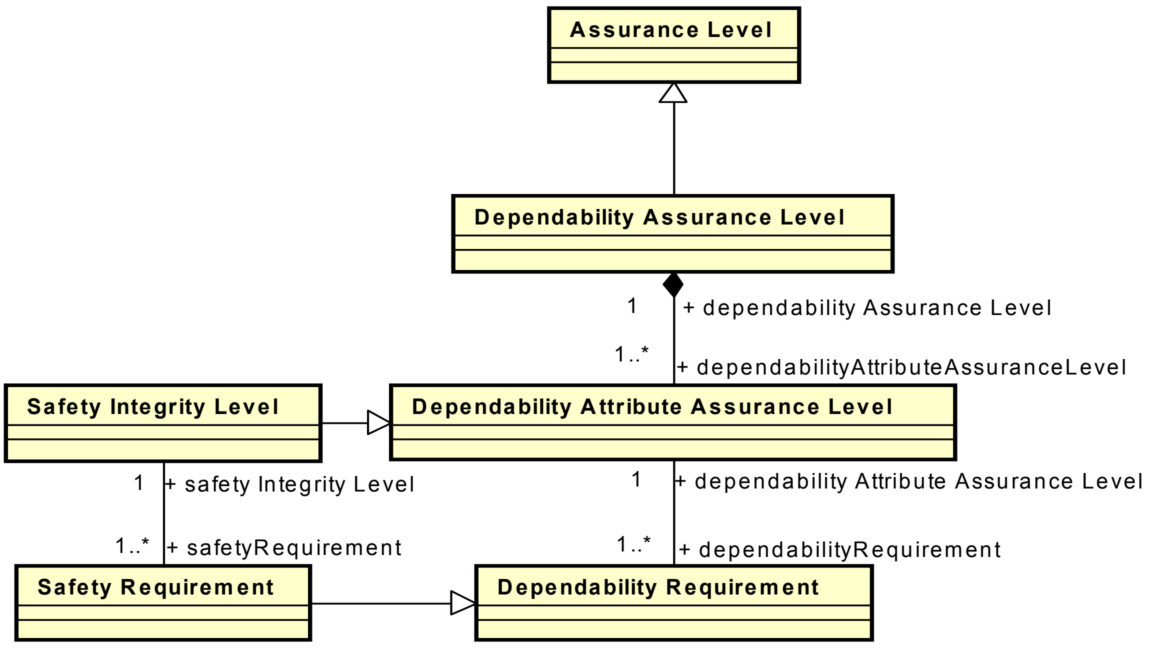

Assurance Level
Dependability Assurance Level
1
+ dependability Assurance Level
1..*
+ dependabilityAttributeAssuranceLevel
Safety Integrity Level
Dependability Attribute Assurance Level
1
+ safety Integrity Level
1
+ dependability Attribute Assurance Level
1..*
+ safetyRequirement
1..*
+ dependabilityRequirement
Safety Requirement
Dependability Requirement

Figure 7.11 - Sample of SIL in IEC 61508 in SSCD specification (Informative)

7.3.4 Error Model

This sub clause specifies the semantic model for the Error Model. This model in Figure 7.12 contains the basic structural elements for defining the error on which the dependability argumentation is laid out. The model is referenced with the conventional error model following the seminal work by Laprie[2] to provide consistency.

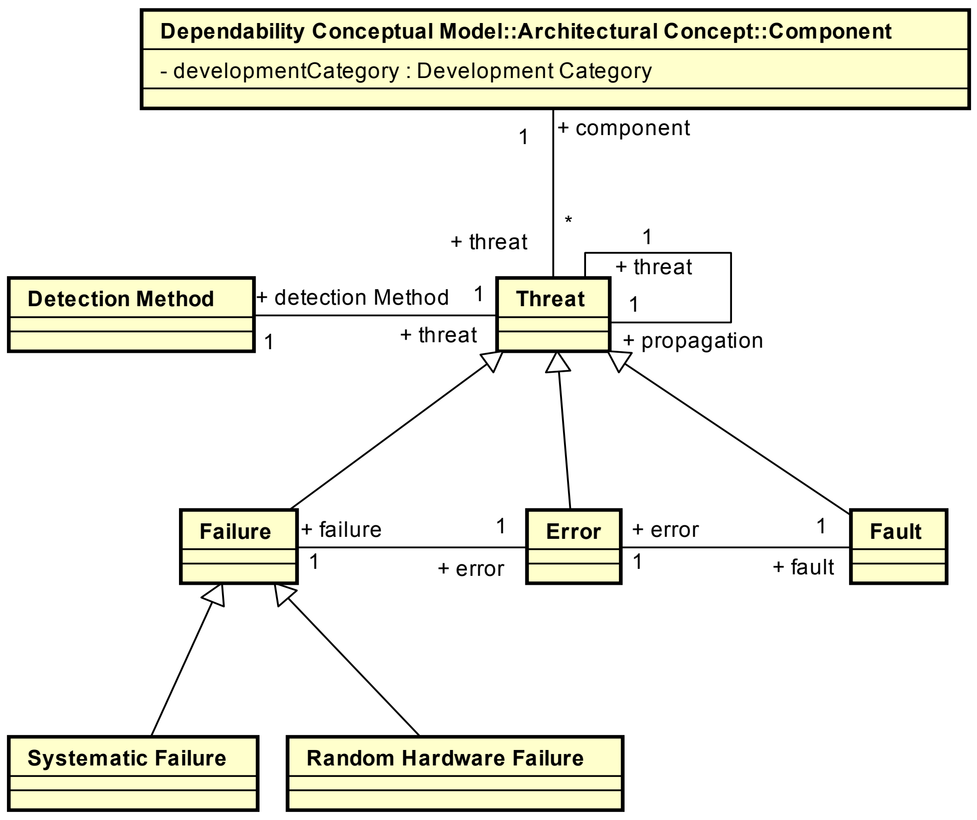


Figure 7.12 - Error Model

7.3.4.1 Threat

Description

Threat is an abstracted notion of fault, error, and failure that occurs in a Component or an Element.

Generalizations

No additional generalization

Attributes

No additional attributes

Associations

component

Specifies that threat happens in a component[1].

threat:Threat

Specifies that threat propagation occurs from one to other[1].

detection Method:Detection Method

Specifies that threat is detected by a detection Method [1].

7.3.4.2 Failure

Description

Failure is an event that occurs when the delivered service deviates from correct service. It is also described as a transition from correct service to incorrect service.

Generalizations

Systematic Failure

Random Hardware Failure

Attributes

No additional attributes

Associations

error:Error

Specifies that failure is caused by an error[1].

7.3.4.3 Random Hardware Failure

Description

Random Hardware Failure is a failure that can occur unpredictably during the lifetime of a hardware element and that follows a probability distribution.

Generalizations

No additional generalization

Attributes

No additional attributes

Associations

No additional associations

7.3.4.4 Systematic Failure

Description

Systematic Failure is a failure related in a deterministic way to a certain cause, which can only be eliminated by a change of the design or of the manufacturing process, operational procedures, documentation, or other relevant factors.

Generalizations

No additional generalization

Attributes

No additional attributes

Associations

No additional associations

7.3.4.5 Error

Description

Error is a deviation from correct service, which defines one or more discrepancies between a computed, observed, or measured value or condition, and the true, specified, or theoretically correct value or condition.

Generalizations

No additional generalization

Attributes

No additional attributes

Associations

fault: Fault

> Specifies that error is caused by an fault[1].

7.3.4.6 Fault

Description

Fault is an abnormal condition that can cause a system or a component to fail.

Generalizations

No additional generalization

Attributes

No additional attributes

Associations

No additional associations

7.3.4.7 Detection Method

Description

Detection Method is a method to identify a Threat.

Generalizations

No additional generalization

Attributes

No additional attributes

Associations

No additional associations

7.3.5 Assessment

The Assessment package in Figure 7.13 accommodates all of the relevant notions of assessment in the SSCD domains.

Dependability of a system is assessed based on relevant assurance requirements. The objects to be assessed are artifacts produced through the dependability process.

An assessment should be done using a Confirmation Review, which is includes a Confirmation Measure to assess the degree of dependability to be achieved.

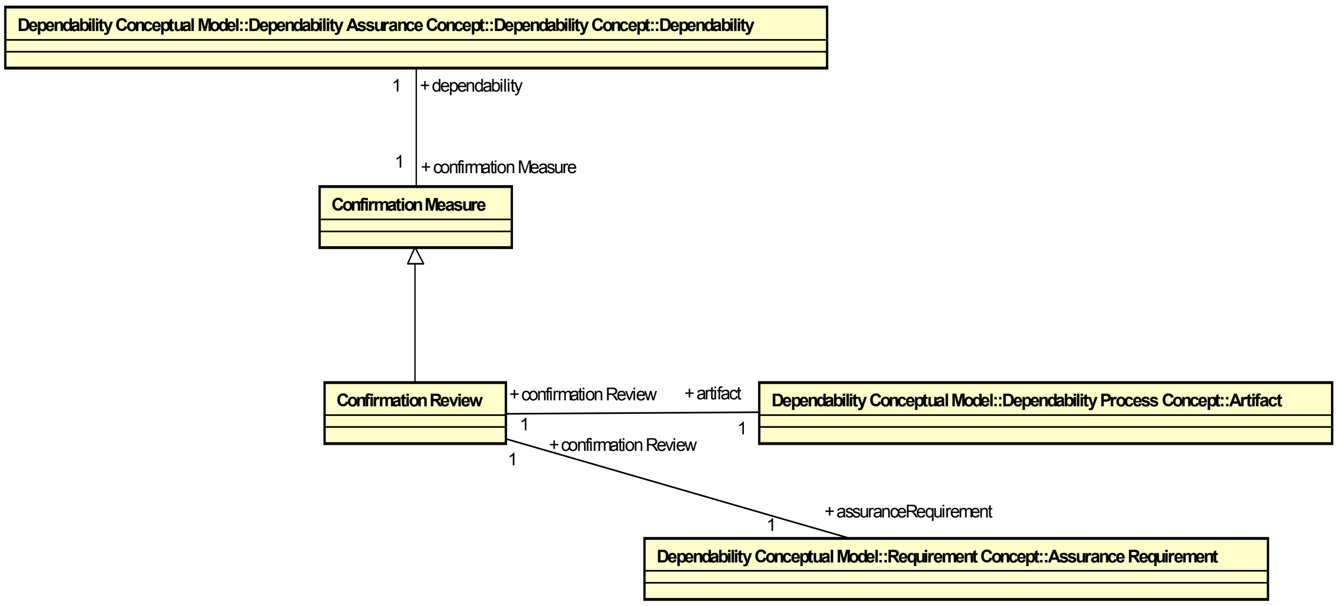

Dependability Conceptual Model::Dependability Assurance Concept::Dependability Concept::Dependability
1
+dependability
1
+confirmation Measure
Confirmation Measure
Confirmation Review
+confirmation Review
1
+artifact
1
Dependability Conceptual Model::Dependability Process Concept::Artifact
+confirmation Review
1
1
+assuranceRequirement
Dependability Conceptual Model::Requirement Concept::Assurance Requirement

Figure 7.13 - Assessment package

7.3.5.1 Confirmation Review

Description

Confirmation Review means to confirm whether artifacts produced during the development cycle satisfy the relevant Assurance Requirements.

Generalizations

Confirmation Measure in the next subsection.

Attributes

No additional attributes

Associations

artifact: Artifact [1]

Specifies whether Artifact satisfies the Assurance Requirement.

assuranceRequirement: Assurance Requirement [1]

Specifies the Assurance Requirement is referenced by Confirmation Review.

7.3.5.2 Confirmation Measure

Description

A Confirmation Measure specifies how the degree of dependability is achieved.

Generalizations

No additional generalizations

Attributes

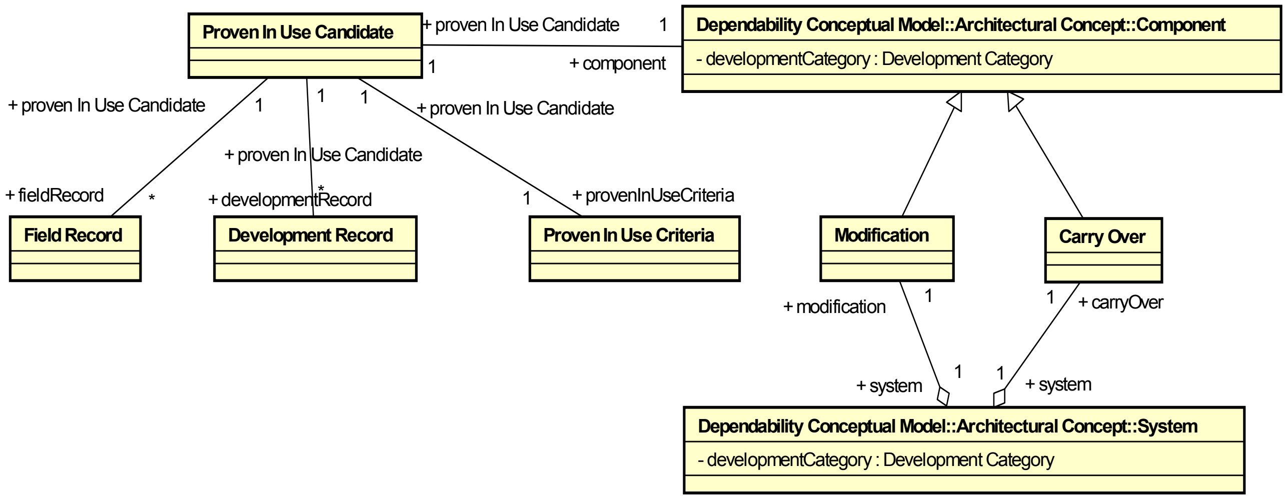
No additional attributes

Associations

No additional associations

7.3.6 Proven In Use

This package depicted in Figure 7.14 provides all the relevant notions related to the Proven In Use. Proven In Use is the notion which describes that a certain part of the existing system is fit for purpose without any further assessment provided that some certain conditions are met. Many industrial products have legacy parts with proven track records to ensure their dependability so we regard this notion as a focal point for their dependability assurance.



Proven In Use Candidate
+ proven In Use Candidate
1
+ component
1
Dependability Conceptual Model::Architectural Concept::Component
- developmentCategory : Development Category
+ proven In Use Candidate
1
1
1
+ proven In Use Candidate
+ proven In Use Candidate
+ fieldRecord
*
+ developmentRecord
*
1
+ provenInUseCriteria
Field Record
Development Record
Proven In Use Criteria
Modification
Carry Over
+ modification
1
1
+ carryOver
+ system
1
1
+ system
Dependability Conceptual Model::Architectural Concept::System
- developmentCategory : Development Category

Figure 7.14 - Proven In Use package

7.3.6.1 Modification

Description

Modification specifies any component of the system which has been modified.

Generalizations

Component in Architectural Concept package

Attributes

No additional attributes

Associations

No additional associations

Note – Modification may include program updates, design changes, and so on.

7.3.6.2 Carry Over

Description

This notion specifies any component of the system which did not change in a new development cycle.

Generalizations

Component in the Architectural Concept package

Attributes

No additional attributes

Associations

No additional associations

7.3.6.3 Proven In Use Candidate

Description

This notion specifies what is assessed for proven in use. The proven in use candidate is a component of a system.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

provenInUseCriteria: Proven in Use Criteria [1]

Specifies that Proven in Use Candidate of a component of the System is assessed by Proven In Use Criteria.

developmentRecord: Development Record [0..*]

Specifies that Proven in Use Candidate is assessed using Development Record.

fieldRecord: Field Record [0..*]

Specifies that a Proven in Use Candidate may have a Field Record.

component: Component [1]

Specifies that Proven in Use Candidate is a component.

7.3.6.4 Proven In Use Criteria

Description

Proven In Use Criteria are those by which a system is proven to be safe based on a proven track record.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

proven In Use Candidate: Proven In Use Candidate [1]

Specifies that Proven In Use Criteria is met to be a Proven In Use Candidate.

7.3.6.5 Field Record

Description

This class signifies any data recorded and any evidence produced while a proven in use candidate is in operation.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

proven In Use Candidate: Proven In Use Candidate [1]

Specifies that Field Record is an evidence from field for a Proven In Use Candidate.

Note – Examples of Field Record include failure rates of any particular parts of the system and incident rates of the system.

7.3.6.6 Development Record

Description

This class specifies any data recorded and evidence produced during the development of the system. This may include Artifacts (defined in the Dependability Process Model) produced during the system development and any records of that development.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

proven In Use Candidate: Proven In Use Candidate [1]

Specifies that Development Record is an evidence during development for a Proven In Use Candidate.

Note – Examples of the Development Record include fault rates of programs.

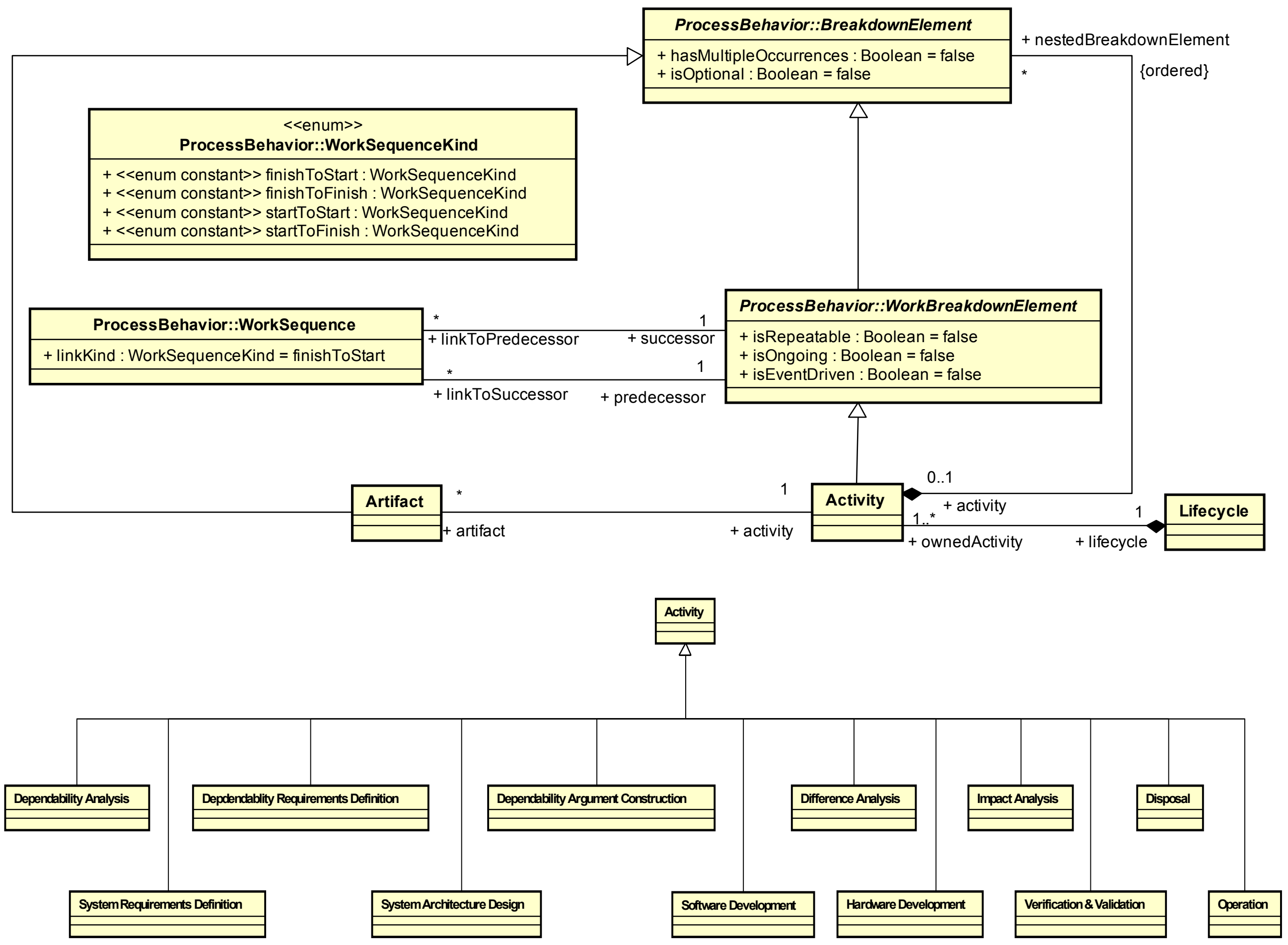
7.4 Dependability Process Concept

This sub clause specifies the semantic model for the Dependability Process Model. This model contains the basic structural elements for defining the Dependability Process. The Dependability process is defined in iterations of each Dependability specific process, that is, design systems, simulation and operation, etc. repeatedly. In the Dependability Process meta-model, the process represents this nature. The iteration process is prescribed partially using the “Software & Systems Process Engineering meta-model Specification version 2.0 (SPEM 2.0)” to represent the framework.

- Sub clause 7.4.1 indicates the conceptual model for the Dependability Process.
- The sub clause after 7.4.1 illustrates each constituent element represented in the conceptual model.

7.4.1 Conceptual Model for Dependability Process

The Dependability Process Model is realized by iterative processes which are composed of dependability specific activities. For establishing the iterative processes, SPEM 2.0 is introduced. BreakdownElement, WorkBreakDown Element, and Work Sequence are imported from SPEM 2.0. (For simplification of the model diagrams, these classes are shown as if they were defined as part of this Dependability Process package). The Dependability specific elements are prescribed in the framework. Each concrete dependability activity is represented as a leaf class. The meta-model for the Dependability Process Model is shown in Figure 7.15.



ProcessBehavior::BreakdownElement
+ hasMultipleOccurrences : Boolean = false
+ isOptional : Boolean = false
+ nestedBreakdownElement
*
{ordered}
<<enum>>
ProcessBehavior::WorkSequenceKind
+ <<enum constant>> finishToStart : WorkSequenceKind
+ <<enum constant>> finishToFinish : WorkSequenceKind
+ <<enum constant>> startToStart : WorkSequenceKind
+ <<enum constant>> startToFinish : WorkSequenceKind
ProcessBehavior::WorkSequence
+ linkKind : WorkSequenceKind = finishToStart
*
+ linkToPredecessor
1
+ successor
*
+ linkToSuccessor
1
+ predecessor
ProcessBehavior::WorkBreakdownElement
+ isRepeatable : Boolean = false
+ isOngoing : Boolean = false
+ isEventDriven : Boolean = false
Artifact
*
+ artifact
1
+ activity
Activity
0..1
+ activity
1..*
+ ownedActivity
1
+ lifecycle
Lifecycle
Activity
Dependability Analysis
Depdendablity Requirements Definition
Dependability Argument Construction
Difference Analysis
Impact Analysis
Disposal
System Requirements Definition
System Architecture Design
Software Development
Hardware Development
Verification & Validation
Operation

Figure 7.15 - Dependability Process Model

7.4.2 Activity

Description

An Activity is a specialization of WorkBreakdownElement that constitutes the iterative process for the dependability process. The Activity requires and/or produces some artifacts. Therefore, it has to possess Artifacts, that is, Activity has an association to the Artifact.

Activity consists of iterative processes, which implies a dependability process. A BreakdownElement is a generalization of Activity, that is, a specific action (for example, Difference Analysis, Dependability Analysis, Dependability Requirement Definition, Dependability Argument Construction, System Requirement Definition, System Architecture Design, etc.) An Activity can have an Artifact, which implies input and/or output of each concrete work. Therefore, the Activity has a relationship to the Artifacts.

The Activity constitutes the Lifecycle, that is, the Activity is related to Lifecycle as an Aggregation.

In this document, the term Activity is used according to SPEM2.0, instead of the term Task to BPMN as the conceptional model of process follows SPEM2.0.

Generalizations

WorkBreakdownElement

Attributes

No additional attributes

Associations

Artifact: Artifact[*]

> References the artifacts which are developed in each activity.

nestedElement: BreakdownElement[*]

> References the BreakdownElement which constructs arbitrary structure of activity recursively.

lifecycle: Lifecycle[1]

> References lifecycle which consists of each activity.

7.4.3 Artifact

Description

An Artifact implies a work product which is produced and/or referred by activities, that is, the Artifact is an activity-specific occurrence of input/output materials. The Artifact needs to be related to a corresponding Activity (as a specialized class). Furthermore, the identical Artifact can be referred to by multiple Activities. The Artifact can be evidence of dependability processes. The BreakdownElement is a generalization of Artifact.

The Artifact instance is an activity-specific object and represents the occurrence of a real work product in the Activity. Therefore, the Artifact has relationship to the Activity. The Artifact is a specialization of BreakdownElement.

An Artifact implies a work product which is produced and/or referred to by activities, that is, the Artifact is an activity-specific occurrence of input/output materials. The Artifact needs to be related to a corresponding Activity (as a specialized class). Furthermore, the identical Artifact can be referred to by multiple Activities. The Artifact can be evidence of dependability processes. The BreakdownElement is a generalization of Artifact.

The Artifact instance is an activity-specific object and represents the occurrence of a real work product in the Activity. Therefore, the Artifact has a relationship to the Activity.

The Artifact is a specialization of BreakdownElement.

Generalizations

BreakdownElement

Attributes

No additional attributes

Associations

activity: Activity[1]

> References the activity which produces the artifacts.

7.4.4 BreakdownElement (from SPEM 2.0)

Description

BreakdownElement is an abstract generalization for any type of process element that is part of a breakdown structure. It defines a set of properties available to all of its specialization. Any of its concrete subclass can be ‘placed inside’ an Activity (via the nested BreakdownElement association) to become part of a breakdown of Activities. As Activities are BreakdownElements themselves and therefore can be nested inside other activities, an n-level break structure is defined by n nested Activities. In addition to Activity, other BreakdownElement can be nested inside Activities as leaf elements of the breakdown.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

aggregateElement: Activity[0..1]

> References activities are defined recursively.

7.4.5 Disposal

Description

A Decommission represents a work item for the dependability. The Decommission is a specialization of the Activity. The Decommission implies work which disposes of the devices.

Generalizations

Activity

Attributes

No additional attributes

Associations

No additional associations

7.4.6 Difference Analysis

Description

A Difference Analysis represents a work item for the dependability process. The Difference Analysis is a specialization of the Activity. The Difference Analysis implies the work which identifies differences in requirements from the previous development.

Generalizations

Activity

Attributes

No additional attributes

Associations

No additional associations

7.4.7 Dependability Analysis

Description

A Dependability Analysis represents a work item for dependability. The Dependability Analysis is a specialization of the Activity. The Dependability Analysis implies the work which analyzes dependability factors.

Generalizations

Activity

Attributes

No additional attributes

Associations

No additional associations

7.4.8 Dependability Argument Construction

Description

A Dependability Argument Construction represents a work item for dependability. The Dependability Argument Construction implies tasks which are required to build the argument structure of dependability. The Dependability Argument Construction is a specialization of Activity.

Generalizations

Activity

Attributes

No additional attributes

Associations

No additional associations

7.4.9 Dependability Requirements Definition

Description

A Dependability Requirement Definition represents a work item for dependability. The Dependability Requirement Definition is a specialization of Activity. The Dependability Requirement Definition implies work items which define the requirements for dependability.

Generalizations

Activity

Attributes

No additional attributes

Associations

No additional associations

7.4.10 Hardware Development

Description

Hardware Development represents a work item for dependability. The Hardware Development is a specialization of Activity. The Hardware Development implies work items which develop hardware.

Generalizations

Activity

Attributes

No additional attributes

Associations

No additional associations

7.4.11 Impact Analysis

Description

An Impact Analysis represents a work item for dependability. The Impact Analysis is a specialization of Activity. The Impact Analysis implies work items which analyze influence on the changed systems in Prove In Use. Namely, it is to detect defects which are caused by changes of Prove In Use.

Generalizations

Activity

Attributes

No additional attributes

Associations

No additional associations

7.4.12 Lifecycle

Description

A Lifecycle is a process, which implies entire development from dependability analysis to the decommissioning of a system. The Lifecycle is shown as a sequence (combination) of concrete works. In general, the Lifecycle is realized as an iterative process.

A Lifecycle designates the entire process. To indicate its circumstance, the Lifecycle is an aggregation of the Activity(s), which implies an entire sequence of concrete activities.

Generalizations

Activity

Attributes

No additional attributes

Associations

ownedActivity: Activity[1..*]

References activities which are owned by this lifecycle.

7.4.13 System Requirements Definition

Description

A System Requirements Definition is a specialization of Activity, which represents a work item for dependability. The System Requirements Definition implies work items which construct the dependability requirements.

Generalizations

Activity

Attributes

No additional attributes

Associations

No additional associations

7.4.14 System Architecture Design

Description

A System Architecture Design represents a work item for dependability. The System Architecture Design is a specialization of Activity. The System Architecture Design implies work items which design system architecture for dependability.

Generalizations

Activity

Attributes

No additional attributes

Associations

No additional associations

7.4.15 Software Development

Description

Software Development represents a work item for dependability. The Software Development is a specialization of Activity. The Software Development implies a work item which implements software in accordance with system requirements definition and system architecture design, etc.

Generalizations

Activity

Attributes

No additional attributes

Associations

No additional associations

7.4.16 Operation

Description

An Operation is a specialization of Activity, which represents a work item for dependability. The Operation implies work items which make the system function.

Generalizations

Activity

Attributes

No additional attributes

Associations

No additional associations

7.4.17 System Architecture

Description

A System Architecture represents a work item for dependability. The System Architecture is a specialization of Activity. The System Architecture implies work items which build the system architecture.

Generalizations

Activity

Attributes

No additional attributes

Associations

No additional associations

7.4.18 Verification & Validation

Description

Verification & Validation represents a work item for dependability. The Verification & Validation is a specialization of Activity. The Verification & Validation implies work items which verify & validate the developing system in accordance with the dependability concept.

Generalizations

Activity

Attributes

No additional attributes

Associations

No additional associations

7.4.19 WorkBreakdownElement (from SPEM 2.0)

Description

A Work Breakdown Element is a special Breakdown Element that provides specific properties for Breakdown Elements that represent work. See Clause 9.10 in SPEM 2.0.

Generalizations

Activity

Attributes

No additional attributes

Associations

- linkToPredecessor: WorkSequence[*]
  This association links a WorkBreakdownElement to its predecessor. Every WorkBreakdownElement can have predecessor information associated to it. This predecessor information is stored in instances of the class WorkSequence that defines the kind of predecessor another WorkBreakdownElement represents for another.
- linkToSuccessor: WorkSequence[*]
  This association links a WorkBreakdownElement to its successor. Every WorkBreakdownElement can have successor information associated to it. This successor information is stored in instances of the class WorkSequence that defines the kind of successor another WorkBreakdownElement represents for another.

7.4.20 WorkSequence (from SPEM 2.0)

Description

Work Sequence is a Breakdown Element that represents a relationship between two Work Breakdown Elements in which one Work Breakdown Elements depends on the start or finish of another Work Breakdown Elements in order to begin or end. See Clause 9.13 in SPEM 2.0.

Generalizations

No additional generalizations

Attributes

- linkKind: WorkSequenceKind
  This attribute express the type of the Work Sequence relationship by assigning a value from the Work Sequence Kind enumeration.

Associations

- successor: WorkBreakdownElement[1]
  This association links a WorkBreakdownElement to its successor. Every WorkBreakdownElement can have successor information associated to it. This successor information is stored in instances of the class WorkSequence that defines the kind of successor another WorkBreakdownElement represents for another.

- predecessor: WorkBreakdownElement[1]
  This association links a WorkBreakdownElement to its predecessor. Every WorkBreakdownElement can have predecessor information associated to it. This predecessor information is stored in instances of the class WorkSequence that defines the kind of predecessor another WorkBreakdownElement represents for another.

7.4.21 WorkSequenceKind (from SPEM 2.0)

Description

Work Sequence represents a relationship between two Work Breakdown Elements in which one Work Breakdown Element depends on the start or finish of another Work Breakdown Element in order to begin or end. This enumeration defines the different kinds of Work Sequence relationships available in SPEM 2.0 and is used to provide values for Work Order's linkKind attribute. See Clause 9.14 in SPEM 2.0.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

Enumeration Literals

- finishTo Start - a WorkBreakdownElement cannot start until another WorkBreakdownElement finishes. For example, if you have two WorkBreakdownElements, "Construct fence" and "Paint fence," "Paint fence" can't start until "Construct fence" finishes. This is the most common type of dependency and the default for a new WorkSequence instance.

- finishToFinish - a WorkBreadownElement cannot finish until another WorkBreakdownElement finishes. For example, if you have two WorkBreakdownElements, "Add wiring" and "Inspect electrical," "Inspect electrical" can't finish until "Add wiring" finishes.

- startToStart - a BreakdownElement cannot start until another WorkBreakdownElement starts. For example, if you have two WorkBreakdownElements, "Pour foundation" and "Level concrete," "Level concrete" can't begin until "Pour foundation" begins.

- startToFinish - a BreakdownElement cannot finish until another WorkBreakdownElement starts. This dependency type can be used for just-in-time scheduling up to a milestone or the project finish date to minimize the risk of a WorkBreakdownElement finishing late if its' dependent WorkBreakdownElements slip. If a related WorkBreakdownElement needs to finish before the milestone or project finish date, but it doesn't matter exactly when and you don't want a late finish to affect the just-in-time WorkBreakdownElement, you can create the dependency between the WorkBreakdownElement you want scheduled just in time (the predecessor) and its related WorkBreakdownElement (the successor). Then, if you update progress on the successor WorkBreakdownElement, it won't affect the scheduled dates of the predecessor WorkBreakdownElement.

7.5 Requirement Concept

This package defines classes related to requirements in general in this specification. Requirements are mainly divided into Assurance Requirements and System Requirements. An Assurance Requirement is any requirement, which ensures that some specific system attribute of a target system is realized. It may include mandatory requirements specifically stated in a specification/standard. Assurance Requirements are to ensure the dependability of a target system and are called Dependability Assurance Requirements. A Dependability Claim specified in a Dependability Assurance Case is specified in this diagram in order to emphasize that the Dependability Claim is a part of a Dependability Assurance Requirement. The relationship between the two explicitly signifies that any claim in a dependability assurance case may be part of a Dependability Assurance Requirement.

A System Requirement is further divided into Quality Requirements and Functional Requirements. The detailed explanation of these notions is included in the next sub clauses.

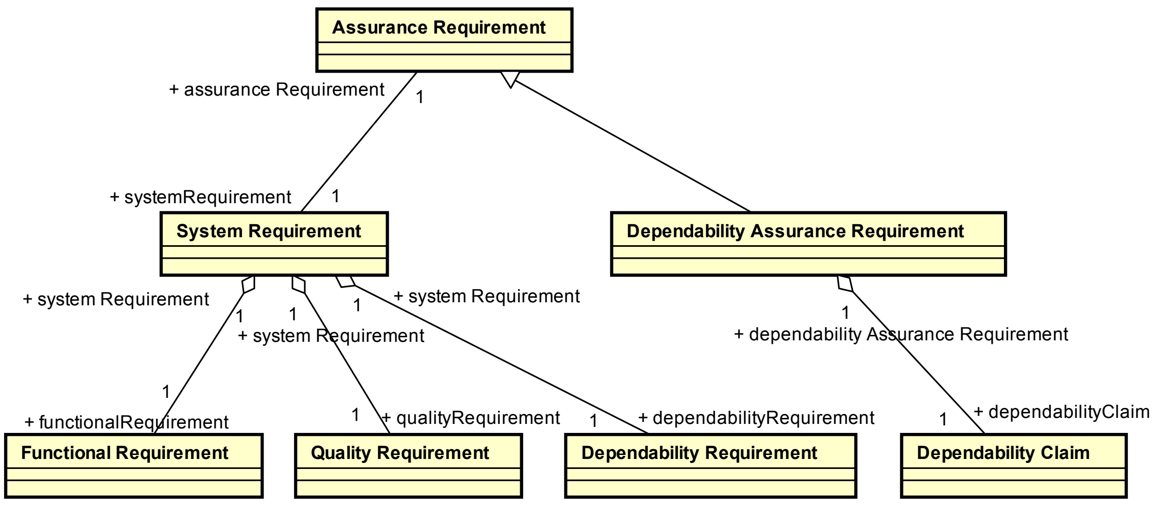


Figure 7.16 - Requirement Concept package

7.5.1 Assurance Requirement

Description

The Assurance Requirement specifies requirements related to system assurance.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

systemRequirement: System Requirement [1]

Specifies that System Requirement is used in Assurance Requirement.

7.5.2 System Requirement

Description

The System Requirement is for specifying requirements related to system architecture.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

functionalRequirement: Functional Requirement[1]

Specifies that Functional Requirement is a part of System Requirement.

qualityRequirement: Quality Requirement[1]

Specifies that Quality Requirement is a part of System Requirement

dependabilityRequirement: Dependability Requirement [1]

Specifies that Dependability Requirement is a part of System Requirement.

7.5.3 Quality Requirement

Description

The Quality Requirement describes the degree of a particular system attribute to be achieved. It is sometimes called a non-functional requirement [6]. In this specification, some crucial non-functional requirements such as safety requirements, reliability requirements, and maintainability requirements are included in Dependability Requirement. We introduced Quality Requirement to signify non-functional requirements other than Dependability Requirement.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.5.4 Functional Requirement

Description

The Functional Requirement signifies the functionality of a target system.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.5.5 Dependability Requirement

Description

The Dependability Requirement is used to achieve the dependability of the target system.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

Note – The definition of dependability must be defined in the Dependability Concept package.

7.5.6 Dependability Assurance Requirement

Description

A Dependability Assurance Requirement specifies assurance requirements for the target system's dependability.

Generalizations

Assurance Requirement specified in 7.5.1.

Attributes

No additional attributes

Associations

dependabilityClaim: Dependability Claim[1]

Specifies that Dependability Claim is a part of Dependability Assurance Requirement.

7.5.7 Dependability Claim

Description

A Dependability Claim states that the target architecture satisfies the Dependability Assurance Requirement.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

dependability Assurance Requirement: Dependability Assurance Requirement[1]

Specifies that Dependability Claim is a claim for Dependability Assurance Requirement.

7.6 System Environment Concept

This package includes all of the relevant notions as to external entities and a relationship between the environment surrounding the system and the system itself.

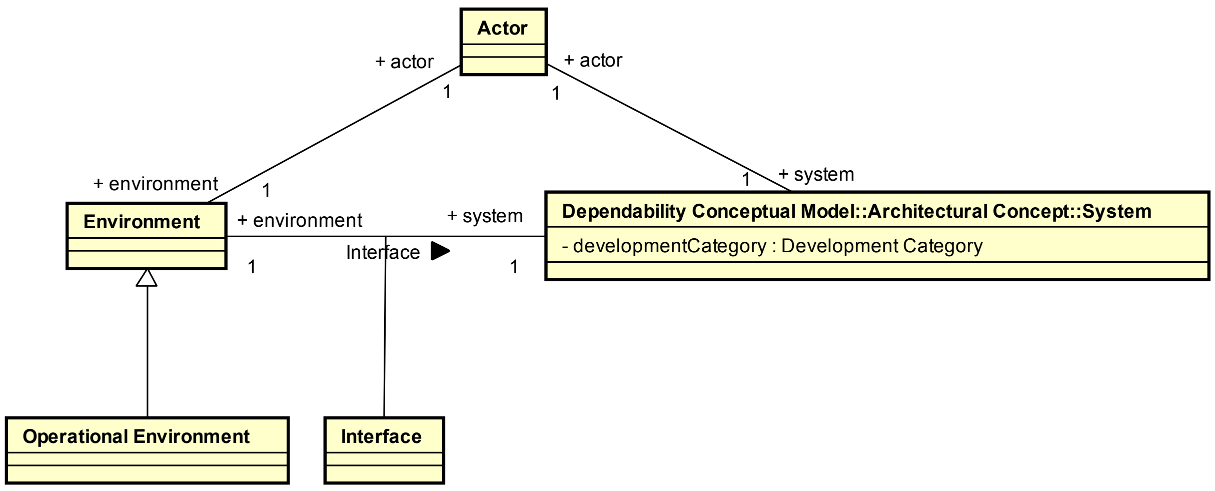


Figure 7.17 - System Environment Concept package

7.6.1 Actor

Description

The Actor may be a stakeholder or a user of the system.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

environment: Environment[1]

Specifies that an Environment influences an Actor.

system:System [1]

Specifies that an Actor interacts with a System.

7.6.2 Environment

Description

The Environment represents anything outside of the system, which may interact with the system. The Environment influences the Actors.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

system: System[1]

Specifies that the Environment may have some influence on the System.

7.6.3 Operational Environment

Description

The Operational Environment is a specific environment in which the System is in operation.

Generalizations

Environment specified in 7.6.2.

Attributes

No additional attributes

Associations

No additional associations

7.6.4 Interface

Description

The Interface represents a connection point between the surrounding environment of the system and the system itself.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

8 Dependability Assurance Case (DAC) Template

8.1 Introduction (Informative)

This Clause introduces DAC (Dependability Assurance Case) templates. The DAC templates are used for writing dependability assurance cases for the target SSCD architecture. A definition of assurance case is as follows.

> A structured argument, supported by a body of evidence that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given environment [1].

Assurance cases have been widely used for safety regulation in the UK and the EU. Safety cases (assurance cases for safety of systems) are required to be submitted to certification bodies for developing and operating safety critical systems, e.g., automotive, railway, defense, nuclear plants, and sea oils. There are several standards such as EUROCONTROL, Rail Yellow Book, and MoD Defense Standard 00-56, which mandate the use of safety cases. In 2010, the USA FDA (Food and Drug administration) required safety cases for introducing infusion pump.

The structure of DAC templates is defined in Dependability Assurance Argument Structure of DCM in sub clause 7.3.1.4. DAC templates are represented by instance diagrams of SACM 1.0 classes.

Dependability Claim in sub clause 7.3.1.1 for the target safety-sensitive consumer device is about the dependability requirements. In Clause 7.5, there are three kinds of requirements in System Requirement: Functional Requirement, Quality Requirement, and Dependability Requirement. Although they may be interrelated, the main concern is about the dependability requirements.

The Dependability Assurance Argument consists of three sub argument structures: Dependability Allocation Argument, Lifecycle Argument, and Standard Compliance Argument. The rationale of these three arguments is as follows.

- Dependability Allocation Argument: To assure that the target architecture is dependable, first we need to define the dependability of the target architecture. As architecture may consist of one or more sub architectures, the dependability attributes should be divided into sub dependability attributes to sub architectures. Therefore, the Dependability Allocation Argument structure becomes recursive according to the structure of the target architecture.
- Lifecycle Argument: To assure that architecture is dependable, this specification requires confirming to DPM (Dependability Process Metamodel). This DAC template is used for that purpose: using this DAC template, the stakeholders can write a dependability assurance case that the lifecycle process of the target architecture adequately conforms to DPM.
- Standard Compliance Argument: It is often the case that there are several other standards to which the target architecture needs to comply with for each SSCD system domain, such as automobile, robotics, smart houses, etc. This DAC template is provided for that purpose.

These three argument templates are developed based on the experiences on developing automobiles by the submitters of this specification. This specification requires using these three DAC templates as normative. The user of this specification may need more other structures of assurance cases depending on his/her system domain, and the user needs to define his/her own argument structures. In such cases, the three argument structures, Dependability Allocation Argument, Lifecycle Argument, and Standard Compliance Argument structures also must be used.

The DAC templates are based on DCM (Dependability Concept Model) in Clause 7 and DPM (Dependability Process Model).

8.2 Representation of DAC Template by SACM Instance Diagram

The DAC templates are defined by SACM Instance Diagrams. The main SACM classes used in this specification are as follows. For detail, please refer to the SACM 1.0 specification [SACM 1.0].

- Claim class: Claims are used to record the propositions of any structured Argumentation. Propositions are instances of statements that could be true or false, but cannot be true and false simultaneously [SACM 1.0].
- AssertedInference class: The AssertedInference association class records the inference that a user declares to exist between one or more Assertion (premises) and another Assertion (conclusion). It is important to note that such a declaration is itself an assertion on behalf of the user [SACM 1.0].
- AssertedContext: The AssertedContext association class declares that the information cited by an InformationElement provides a context for the interpretation and definition of a Claim or ArgumentReasoning element [SACM 1.0].
- InformationElement class: The InformationElement Class enables the citation of a source that relates to the structured argument. The citation is made by the InformationElement class. The declaration of relationship is made by the AssertedRelationship class [SACM 1.0].

8.3 Dependability Allocation Argument

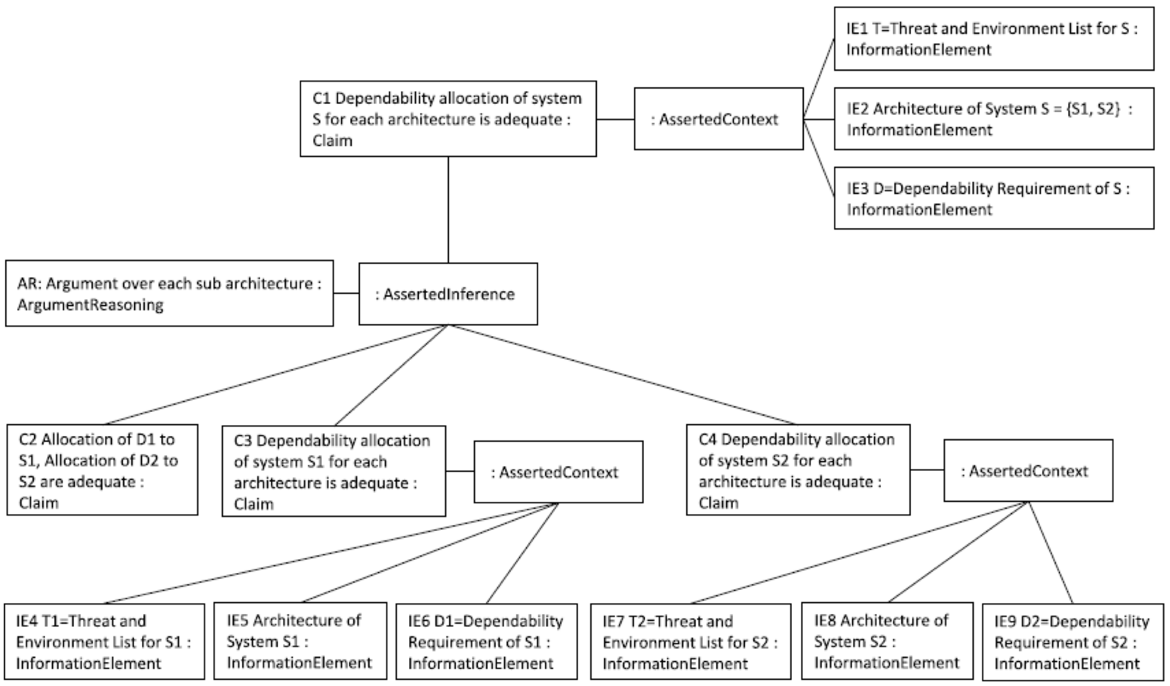

IE1 T=Threat and Environment List for S : InformationElement
C1 Dependability allocation of system S for each architecture is adequate : Claim
: AssertedContext
IE2 Architecture of System S = {S1, S2} : InformationElement
IE3 D=Dependability Requirement of S : InformationElement
AR: Argument over each sub architecture : ArgumentReasoning
: AssertedInference
C2 Allocation of D1 to S1, Allocation of D2 to S2 are adequate : Claim
C3 Dependability allocation of system S1 for each architecture is adequate : Claim
: AssertedContext
C4 Dependability allocation of system S2 for each architecture is adequate : Claim
: AssertedContext
IE4 T1=Threat and Environment List for S1 : InformationElement
IE5 Architecture of System S1 : InformationElement
IE6 D1=Dependability Requirement of S1 : InformationElement
IE7 T2=Threat and Environment List for S2 : InformationElement
IE8 Architecture of System S2 : InformationElement
IE9 D2=Dependability Requirement of S2 : InformationElement

Figure 8.1 - DAC template for Dependability Allocation Argument

Figure 8.1 depicts DAC templates for Dependability Allocation Argument. The DAC template for Dependability Allocation Argument represents that the allocation of dependability requirements of the target architecture is adequate. This template is recursively used for each sub-architecture. The term architecture is used for represents either “System of systems,” “System,” “Component,” or “Implementation” (see Architectural Concept in sub clause 7.2). System S consists of sub systems S1 and S2 (this template assume two sub systems, but the number can be modified according to the target system), and the threat and environmental list for S is derived, and the dependability requirement is D (derived from Dependability Requirements Analysis), then the top claim "C1 Dependability allocation of System S for each system/component/implementation is adequate" is decomposed into the following three sub claims: "C3 Dependability allocation of System S1 for each sub architecture is adequate," "C4 Dependability allocation of System S2 for each sub architecture is adequate," and "C2 Allocation of D1 to S1, Allocation of D2 to S2 are adequate." In this argument, the dependability requirement D is divided into D1 and D2, and they are allocated to S1 and S2, respectively. C3 and C4 are then decomposed into sub claims using this DAC template, according to the structure of S1 and S2, respectively. The adequacy of the decomposition of D into D1 and D2 is assured in the argument of sub claim C2. Threat and environment list is divided into T1 and T2. This division is derived as the result of Dependability Analysis of DPM. Note that the sum of T1 and T2 is not necessarily equals to T: the sum may be less than T.

The XMI file for the DAC template for Dependability Allocation Argument is DependabilityAllocationArgument.xmi (normative).

8.4 Lifecycle Argument

In DAF, the lifecycle of the target architecture must be evolutional, i.e., the architecture is to be developed iteratively over the generations. Therefore, the lifecycle argument structure shall be based on evolutional development of the system, as defined in Dependability Process Model (DPM) in Clause 9.

8.4.1 Evolutionary Development Argument

Evolutionary Development Argument is a kind of Lifecycle Argument. Developing a system by evolutionary development over generation is a main theme of DAF. The overview of the DAC template for Evolutionary Development Argument is shown in Figure 8.2. The DAC template is divided into 4 sub parts: "Top Structure," "Modification Argument," "Proven In Use Argument," and "Integration Argument" parts.

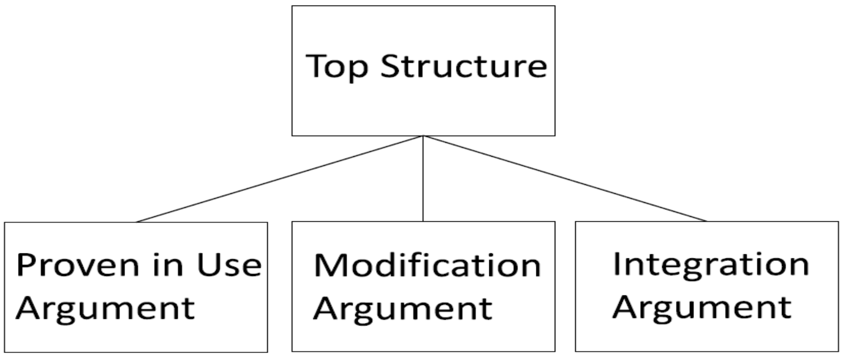


Figure 8.2 - Overview of Evolutionary Development Argument Template (Informative)

The DAC template represents system development based on systems engineering: each architecture is developed by integrating its sub architectures. This corresponds to the Architectural Concept of DPM (Clause 7.2). A System of Systems is developed by integrating its sub systems; a system is developed by integrating its sub systems or components, and so on. For example, a vehicle is a System of Systems, which consists of engine, body, and chassis. They are systems. An engine consists of intake, exhaust, and ECUs. They are components. This DAC template is intended to be recursively used for each modified architecture in Modification argument. For example, consider a development of automobile

(Figure 8.3). Assume that the next generation of the automobile is developed by modifying the existing engine and body parts of the automobile. Then the DAC template is used for writing the DAC of the new engine and body part of the next generation, and they are used as sub trees of the DAC for the next generation of the automobile.

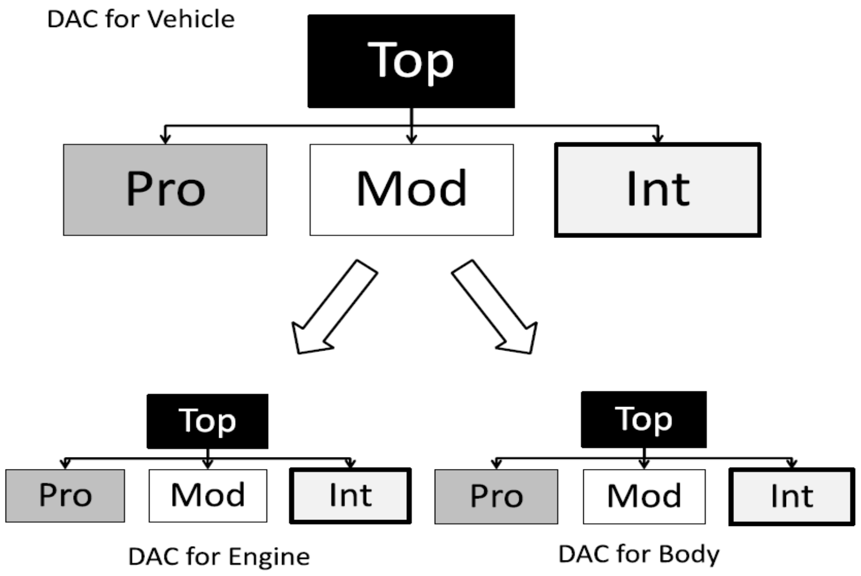


Figure 8.3 - An example of the use of DAC template for automobile (Informative)

Proven In Use argument aims to assure that unchanged part of the target architecture is dependable by existing field and development records. The dependability of the modified parts of the architecture is separately assured using the DAC template recursively.

Separately assuring the modified parts with proven in use argument is not enough for assuring the dependability of the whole target architecture. We also need to assure that the whole architecture satisfies the required dependability attributes by integration argument. This forms the three sub argument structures, and represents main motivations of the SSCD standards: proven in use, systems engineering, and evolutionary development.

8.4.2 Top Structure

Figure 8.4 shows Top Structure of the DAC for Evolutionary Development Argument. The top claim C1 states that the target architecture satisfies given dependability attributes. Information Element IE1 states the specification of the changed parts of the target architecture. Information Element IE2 states allocation of dependability attributes to the architecture and its sub architectures. The dependability of the unchanged parts of the target is assured in proven in use argument. The dependability of changed parts of the target architecture is assured in modification argument. Unchanged and changed parts are together assured their dependability as the whole target architecture in integration argument. {Architecture} and {DependabilityAttribute} are placeholders for the name of the target architecture and the dependability attributes. {Architecture} may be replaced with the name of users system, such as “automobile.” {DependabilityAttribute} is the name of the dependability attribute. Dependability attribute should correspond to the definition in sub clause 7.3.2.2.

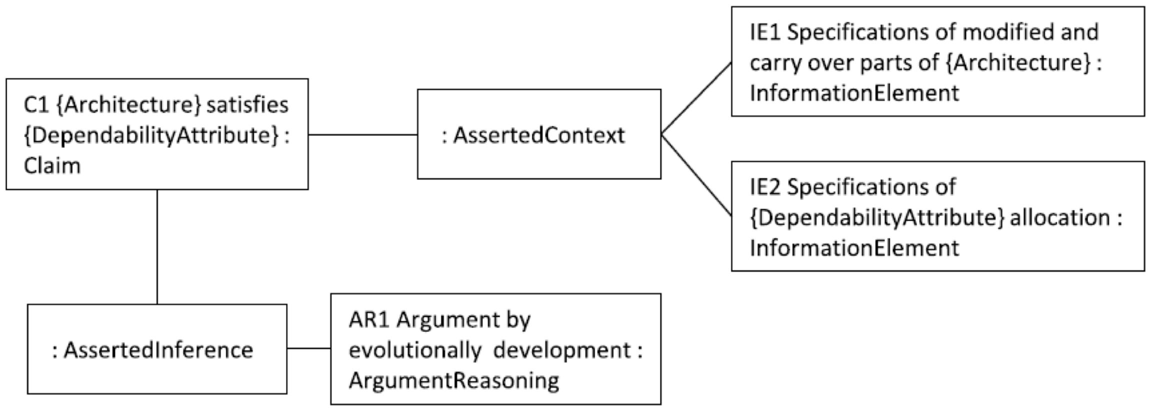

C1 {Architecture} satisfies {DependabilityAttribute} : Claim
: AssertedContext
IE1 Specifications of modified and carry over parts of {Architecture} : InformationElement
IE2 Specifications of {DependabilityAttribute} allocation : InformationElement
: AssertedInference
AR1 Argument by evolutionally development : ArgumentReasoning

Figure 8.4 - Top Structure of the DAC for Evolutionary Development Argument

The XMI file for the top structure of Evolutionary Development Argument is EvolutionaryDevelopmentArgument.xmi (normative).

8.4.3 Proven In Use Argument

In DCM, Proven In Use Argument is defined in sub clause 7.3.6. The Proven In Use Argument corresponds to the Proven In Use package (Figure 7.14). In the package, a system is divided into modification and carry over parts, where they are sub classes of Component class. Proven In Use Argument assures that the carry over part of the target component satisfies the allocated dependability attributes. The carry over parts need to be met with Proven In Use Criteria. If so, then the dependability of the carry over parts is assured using development and field record of the previous generation of the target system.

The SACM instance diagram for Proven In Use Argument DAC template is shown in Figure 8.5. Given proven-in use criteria in Information Element IE3, the argument is for assuring that carry over parts of the target architecture holds allocated dependability attributes. In the left sub tree of Claim C3 is for assuring that the carry over parts satisfies proven-in use criteria as a proven-in use candidate. Claim C3 is supported by Information Element IE4 Confirmation of prove-in use candidate linked by the AsseretedEvidence link. The right sub tree of Claim C4 is for assuring that the carry parts hold allocated dependability attributes using the development and field records. Argument Reasoning AR2 specifies that the sub claims C5 and C6 are decomposed for previous development and operating conditions. Claim C5 is for arguing that given Information Element IE6, which is the development record of the carry-over parts of the architecture, the carry-over part of the architecture satisfies allocated dependability attributes. This argument is supported by Information Element IE5: Artifacts of Development Record of previous architecture. IE5 is linked with C5 by the AsseretedEvidence. Claim C6 is for arguing that given Information Element IE7, which is the field record of previous architectures, the carry-over part of the architecture satisfies allocated dependability attributes. This argument is supported by Information Element IE8: Field Record of carry-over parts of the architecture. Assuring the dependability of the carry-over parts of the architecture by both development artifacts and field record strengthens confidence in the dependability of the architecture.

The structure terminates with five pieces of evidence: Information Elements IE4, IE5, and IE6, IE7, and IE8. Note that instead of these evidence can be replaced with manually written sub trees if necessary.

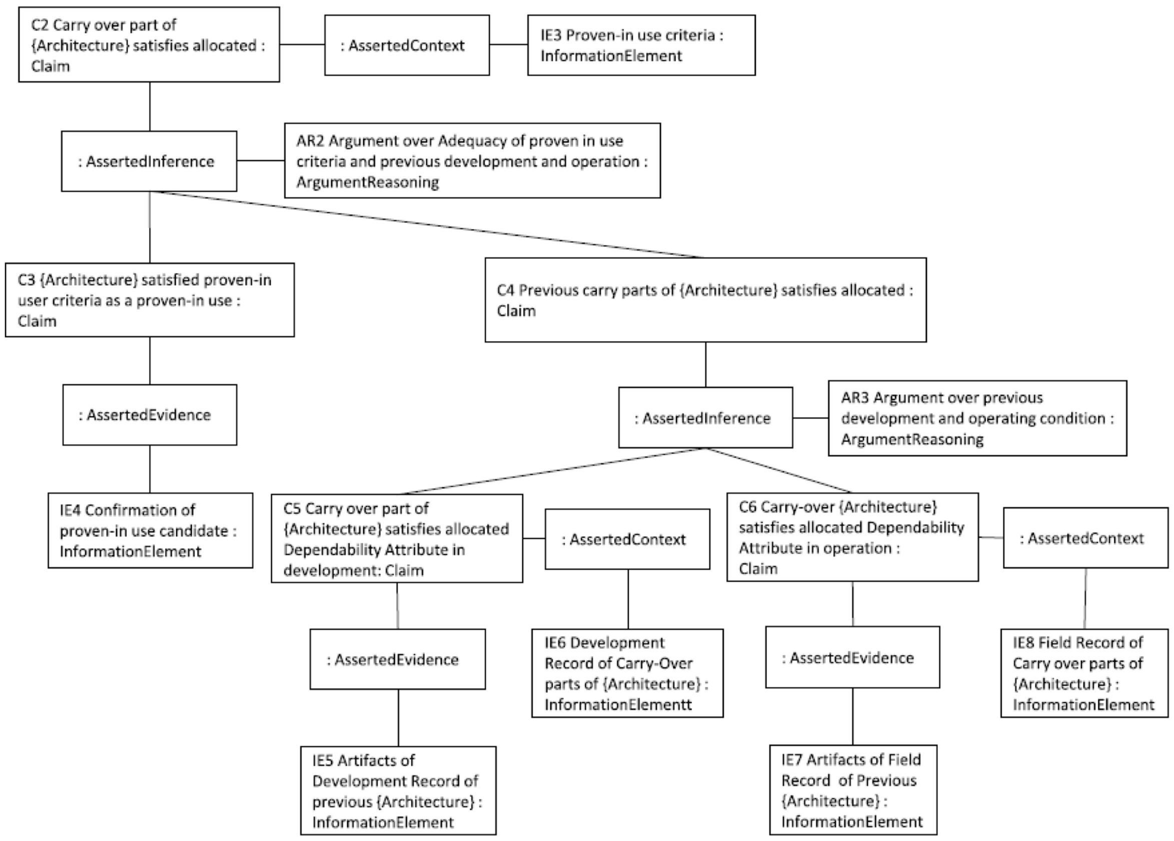

C2 Carry over part of {Architecture} satisfies allocated : Claim
: AssertedContext
IE3 Proven-in use criteria : InformationElement
: AssertedInference
AR2 Argument over Adequacy of proven in use criteria and previous development and operation : ArgumentReasoning
C3 {Architecture} satisfied proven-in user criteria as a proven-in use : Claim
C4 Previous carry parts of {Architecture} satisfies allocated : Claim
: AssertedEvidence
: AssertedInference
AR3 Argument over previous development and operating condition : ArgumentReasoning
IE4 Confirmation of proven-in use candidate : InformationElement
C5 Carry over part of {Architecture} satisfies allocated Dependability Attribute in development: Claim
: AssertedContext
C6 Carry-over {Architecture} satisfies allocated Dependability Attribute in operation : Claim
: AssertedContext
: AssertedEvidence
IE6 Development Record of Carry-Over parts of {Architecture} : InformationElementt
: AssertedEvidence
IE8 Field Record of Carry over parts of {Architecture} : InformationElement
IE5 Artifacts of Development Record of previous {Architecture} : InformationElement
IE7 Artifacts of Field Record of Previous {Architecture} : InformationElement

Figure 8.5 - Proven In Use Argument part of the Evolutionary Development Argument Structure

The XMI file for Proven In Use Argument part of the Evolutionary Development Argument Structure is ProvenInUseArgument.xmi (normative).

8.4.4 Modification Argument

Figure 8.6 represents the SACM instance diagram for Modification Argument DAC template. Claim C7 states modified and impacted parts of the target architecture satisfy each allocated dependability attribute. Information Element IE9 specifies modified and impacted parts of the target architecture. The information is derived from difference analysis and impact analysis defined in DPM. Modification argument structure is for assuring the dependability of modified and impacted parts. ArgumentReasoning AR4 requires the sub claims to be for each modified and impacted part of the architecture. The sub trees are constructed by recursively using the evolutional development argument structure for each sub modified and impacted parts of architectures.

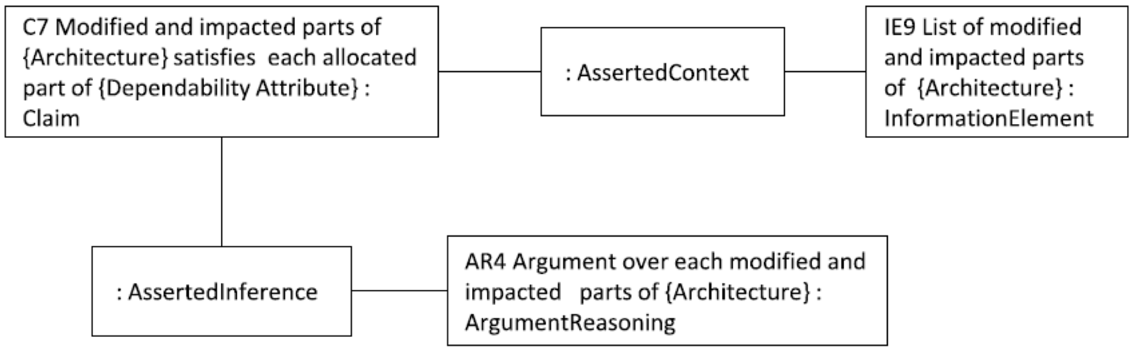

C7 Modified and impacted parts of {Architecture} satisfies each allocated part of {Dependability Attribute} : Claim
: AssertedContext
IE9 List of modified and impacted parts of {Architecture} : InformationElement
: AssertedInference
AR4 Argument over each modified and impacted parts of {Architecture} : ArgumentReasoning

Figure 8.6 - Modification Argument Part of the DAC template for Evolutionary Development Argument Structure

The XMI file for the Modification Argument is ModificationArgument.xmi (normative).

8.4.5 Top Structure of Integration Argument

Integration Argument consists of two sub parts: the static dependability analysis argument and the dynamic dependability analysis argument. The top structure of the Integration Argument is defined in Figure 8.7. Claim C8 states that the integrated target architecture satisfies the dependability attribute(s). In the DAC template, the dependability of the integrated target architecture is assured by both static and dynamic dependability attribute analysis. Static dependability analysis includes conventional difference and impact analysis, and threat analysis specified in the dependability analysis phase of DPM. Dynamic dependability analysis consists of simulation and physical testing.

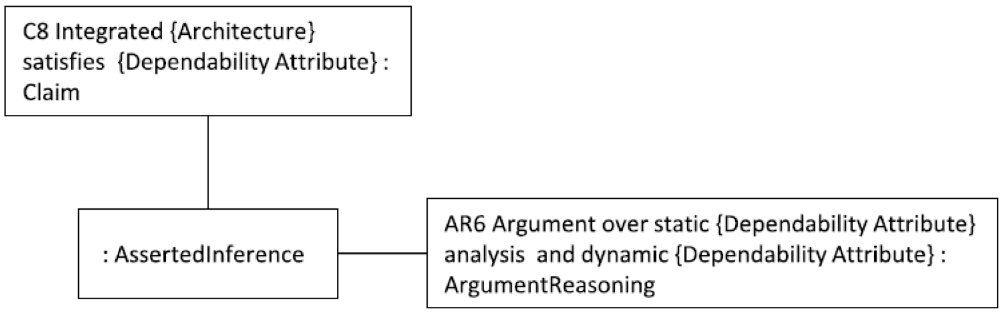

C8 Integrated {Architecture} satisfies {Dependability Attribute} : Claim
: AssertedInference
AR6 Argument over static {Dependability Attribute} analysis and dynamic {Dependability Attribute} : ArgumentReasoning

Figure 8.7 - Top Structure of Integration Argument part of the DAC template for Evolutionary Development Argument Structure

The XMI file for the top structure of Integration Argument part is IntegrationArgument.xmi (normative).

8.4.5.1 Static Dependability Analysis Argument

In the Static Dependability Analysis Argument, the conventional system assurance is discussed. The part of the DAC template is defined in Figure 8.8. The argument consists of about difference analysis, impact analysis, and dependability analysis defined in DPM. The SACM instance diagram for Static Dependability Analysis Argument DAC template is shown in Figure 8.8. InformationElement IE9 specifies Static Dependability Attribute analysis procedures defined in DPM (Difference and Impact analysis and Dependability Analysis). Claims C10, C11, and C12 are for assuring that Difference Analysis, Impact Analysis, and Dependability Analysis, respectively.

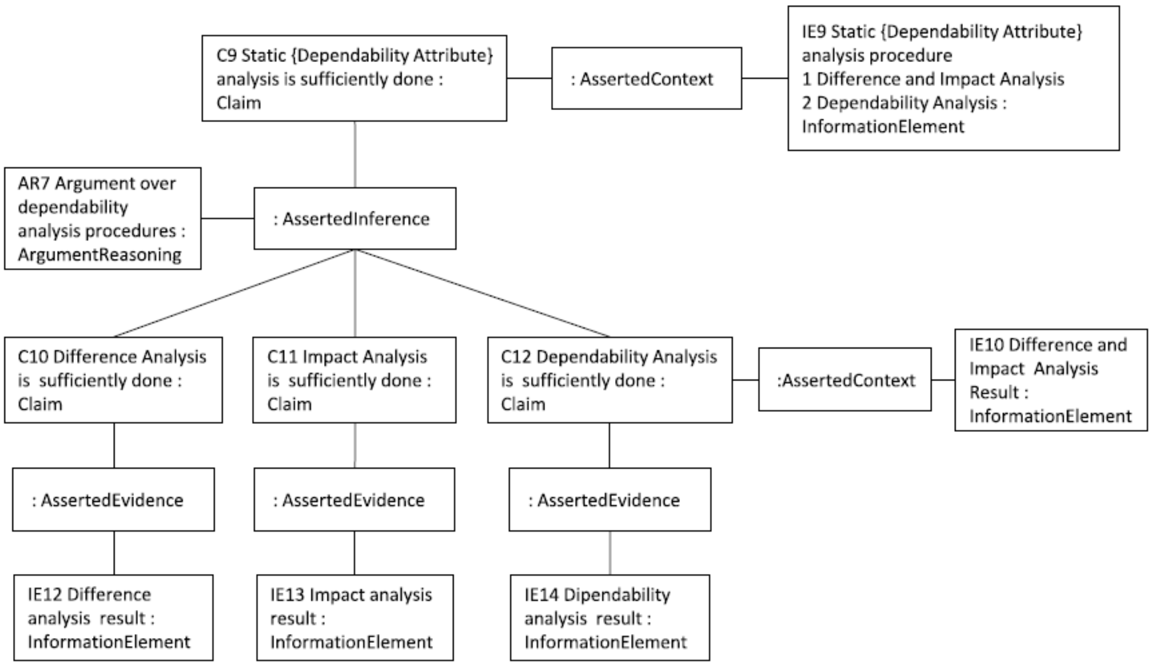

C9 Static {Dependability Attribute} analysis is sufficiently done : Claim
: AssertedContext
IE9 Static {Dependability Attribute} analysis procedure 1 Difference and Impact Analysis 2 Dependability Analysis : InformationElement
AR7 Argument over dependability analysis procedures : ArgumentReasoning
: AssertedInference
C10 Difference Analysis is sufficiently done : Claim
C11 Impact Analysis is sufficiently done : Claim
C12 Dependability Analysis is sufficiently done : Claim
:AssertedContext
IE10 Difference and Impact Analysis Result : InformationElement
: AssertedEvidence
: AssertedEvidence
: AssertedEvidence
IE12 Difference analysis result : InformationElement
IE13 Impact analysis result : InformationElement
IE14 Dipendability analysis result : InformationElement

Figure 8.8 - Static Dependability Analysis Argument part of the DAC template for Evolutionary Development Argument Structure

The XMI file for the Static Dependability Analysis Argument part is StaticDependabilityAnalysisArgument.xmi (normative).

8.4.5.2 Dynamic Dependability Analysis Argument

The Dynamic Dependability Analysis Argument is a unique methodology for the dependability argumentation for SSCDs, considering the characteristics of the products.

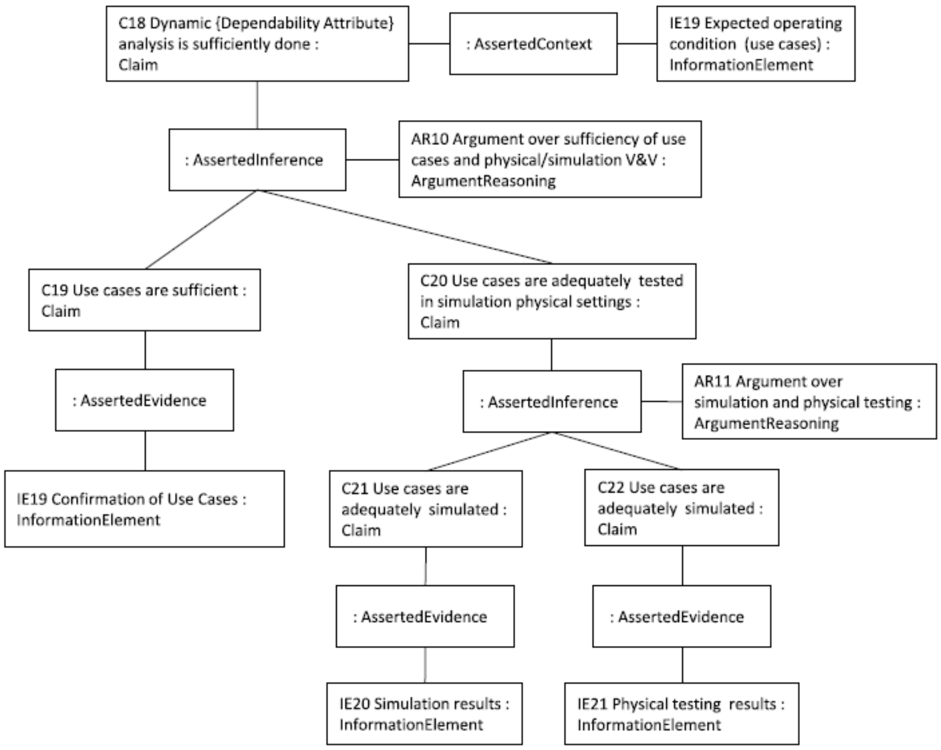

C18 Dynamic {Dependability Attribute} analysis is sufficiently done : Claim
: AssertedContext
IE19 Expected operating condition (use cases) : InformationElement
: AssertedInference
AR10 Argument over sufficiency of use cases and physical/simulation V&V : ArgumentReasoning
C19 Use cases are sufficient : Claim
C20 Use cases are adequately tested in simulation physical settings : Claim
: AssertedEvidence
: AssertedInference
AR11 Argument over simulation and physical testing : ArgumentReasoning
IE19 Confirmation of Use Cases : InformationElement
C21 Use cases are adequately simulated : Claim
C22 Use cases are adequately simulated : Claim
: AssertedEvidence
: AssertedEvidence
IE20 Simulation results : InformationElement
IE21 Physical testing results : InformationElement

Figure 8.9 - Dynamic Dependability Analysis Argument part of the DAC template for Evolutionary Development Argument Structure

Most of the dependability analysis can be done with the conventional system assurance methodology in the Static Dependability Analysis Argument. In order to enhance the dependability, the Dynamic Dependability Analysis Argument needs to be done with physical testing to emulate real use cases for system validation. Given the fact that all the use cases cannot be fully identified because of the nature of SSCDs (Claim C19), the simulation and physical testing have to be repeatedly run to identify as many use case as possible to validate the dependability requirements (Claims C21 and C22).

So, the argumentation structure in Figure 8.9 is necessary to confirm both the sufficiency of the scope of use cases and the sufficiency of the simulation and physical testing.

The XMI file for Dynamic Dependability Analysis Argument part is DynamicDependabilityAnalysisArgument.xmi (normative).

8.5 Standard Compliance Argument

The DAC template for Standard Compliance Argument (Figure 8.10) requires to list up all other standards st1 ,…, stN needed to be complied in the system domain of the SSCD architecture (Information Element C1). For each standard sti (1 <= i <= N), a sub claim is stated as “System S adequately satisfies sti.” Note that in Figure 8.10, only two sub claims for st1 and stN are shown. The number of sub claims is dependent on the number of standards needed to be complied.

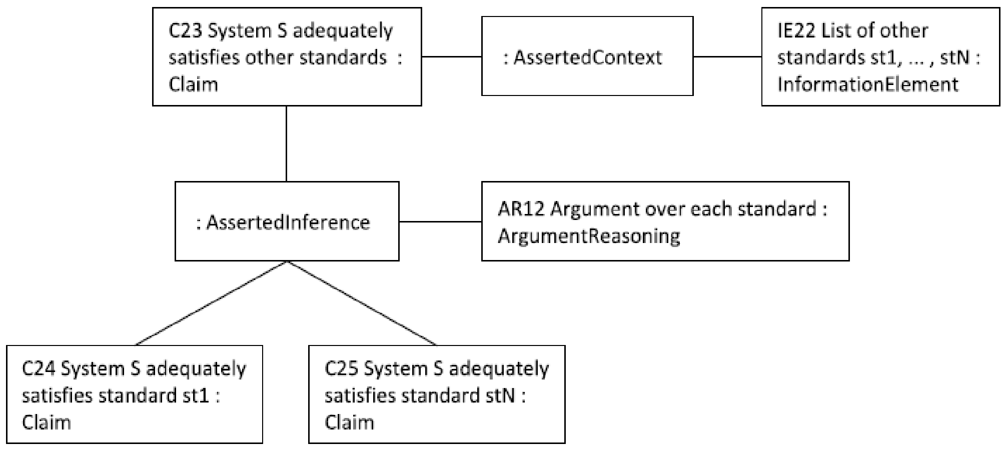

C23 System S adequately satisfies other standards : Claim
: AssertedContext
IE22 List of other standards st1, ... , stN : InformationElement
: AssertedInference
AR12 Argument over each standard : ArgumentReasoning
C24 System S adequately satisfies standard st1 : Claim
C25 System S adequately satisfies standard stN : Claim

Figure 8.10 - The DAC Template for Standard Compliance Argument Structure

The XMI file for the DAC template for Standard Compliance Argument Structure is StandardComplianceArgument.xmi (normative).

9 Dependability Process Model

9.1 General

This clause specifies the Dependability Process Model (DPM) using the DPM conceptual-model of Clause 7.4. DPM defines a process or Activities of dependability assurance for consumer devices based on the conventional Systems Engineering processes with a notion of the iterative and rapid process. The dependability assurance is developed in parallel with the normal product development process and cannot be discussed separately from the corresponding product development process.

9.2 Overview of Iterative and Rapid Process

The V-model (a “V” shaped process model to describe the engineering process from the requirements definition phase, specifications development phase, implementation phase, verification and validation phase) has been well known to describe the engineering process for automotive as a part of systems engineering. The role of V-model is essential to making our development process further efficient so that it is incorporated into the safety development process in ISO26262. The automotive OEMs who follow the ISO26262 have to clearly define the safety process on their own for ISO26262.

One of the challenges, though, to roll out the V-model into organizations is what level of granularity for each sub-process in the V-model is expected to be defined. Obviously, the V-model illustrates only a fraction of the entire engineering process where engineers repeatedly create and modify their products with a heuristic approach on a daily basis. It indicates that small and large V-models need to be addressed at the same time if real engineering process are required to be defined. However, it is not realistic to fully lay out all the V models all at once because of the size of processes for SSCD development.

In order to balance the process definition between the top-down governance such as the V-model and the bottom-up individual processes, our proposal is illustrated in Figure 9.1.

Two circles are supplementary illustrated in Figure 9.1, which implicitly describe the concept of iteration. Also, the iterations are quick and engineers run the iteration many times per development on their own. That is what we call rapid iteration. For the left circle, engineers start off with the requirements engineering to gather requirements for a particular system that they are going to develop. After that, specifications for the system are supposed to be created in line with the requirements defined. After modeling or coding the specifications, they are going to be verified and validated with simulation or testing with physical parts. If (or I should say every time) engineers find something wrong on their control system, they go back to requirements to find out where the failure comes from. Spotting the cause of the failure, engineers modify the corresponding specifications and control models for further calibration and V&V.

In the right circle for the implementation process, once the control models are well matured, engineers are going to find out how efficiently it should be implemented into an ECU (Electronic Control Unit) within available ROM/RAM resources. Engineers need to find a way to reduce the size of the model or code by simplifying or optimizing them. Likewise, automated code generation, followed by the calibration and V&V process, is carried out to identify the most efficient way of implementation by trial and error.

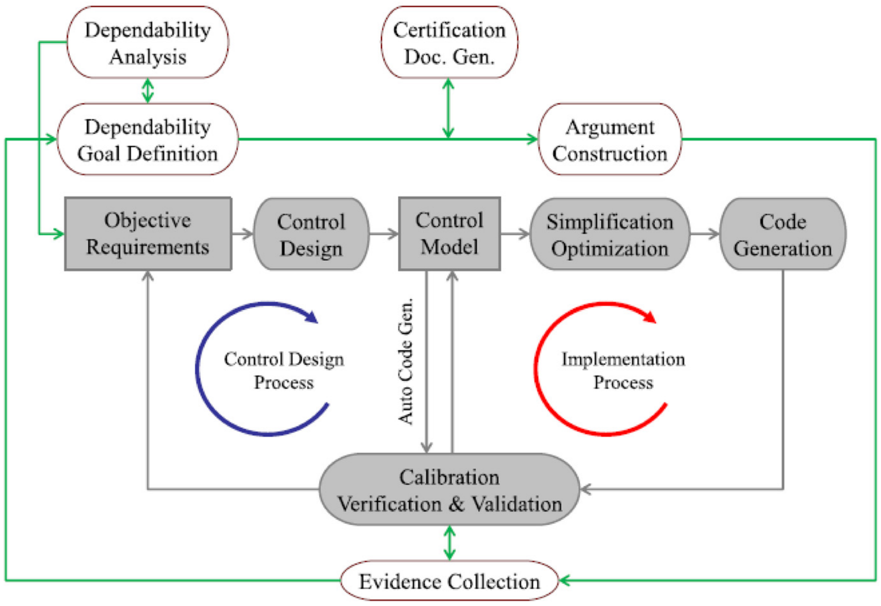


Figure 9.1 - Example of Rapid Iterative Process

This abstract process can be divided into a collection of Activities. In the following clauses, we define a three-layered Activity model. It consists of processes, activities, and tasks. Processes are the Dependability Process, the System Engineering Process, the Evolutionary Development Process, and the Etcetera Process, Each process is sub-divided into activities, and, in turn, each activity is sub-divided into tasks.

9.3 Dependability Process

The Dependability Process is a collection of activities for system development that utilizes systems engineering processes, and it contains following activities:

- Dependability Analysis
- Dependability Requirement Definition
- Construction of Dependability Assurance Cases through Dependability Argument Construction

These activities constitute the Dependability Process. The entire relationship of the related processes and activities are shown in Figure 9.2. As shown in Figure 9.2, the Systems Engineering Processes in the middle of the figure are performed along with the Dependability Requirements Definition activity and the Dependability Argument Construction activity. These activities defined here are a minimum set, and the set can be applied with necessary extensions for various consumer device developments.

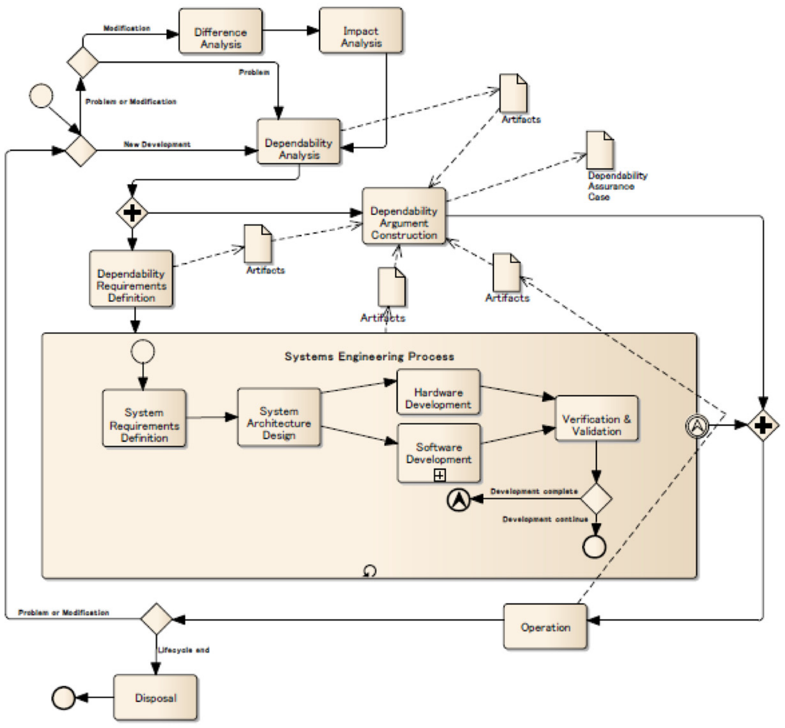


Figure 9.2 - BPMN for Dependability Process Model

9.3.1 Dependability Analysis

In the Dependability Analysis activity, threats and associated operational environments are identified. In this clause and following clauses, the term threat is a general term to mean not only threats in security but also hazards in safety so that the dependability analysis may include hazard identification (in safety analysis) as well as threat identification (in security analysis). Once threats and associated operational environments are identified, their levels are assessed based on dependability attribute assurance levels. This activity is very generic one and the minimum requirement which this specification mandates. The requirements of this clause may be extended to meet more specific system properties depending on the definition of dependability. The tasks mentioned above are a minimum set, and other tasks may be added if necessary.

As examples of inputs, they are the following documents:

- Product Plan provided by upper level processes including Planning.
- Incident Reports from the Problem flow.
- Development results from the Evolutionary Development Process consisting of the Difference Analysis activity and the Impact Analysis activity.

These inputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

The output of this activity is:

- Dependability Analysis Results including the results of Threat Analysis (Threat and operational environment List).

This output is a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

9.3.2 Dependability Requirements Definition

Dependability Requirements are defined based on the results of the Dependability Analysis. Primary inputs are the Threat and operational environment List and the Threat Assessment Results. Dependability Requirements are to mitigate each Threat. They are composed of Reliability Requirements, Availability Requirements, Maintainability Requirements, Safety Requirements, Security Requirements, and other necessary requirements based on the utilized Dependability concept. Various Requirements needed for the development will be selected and defined as a collection of Dependability Requirements.

When the existing Dependability Analysis has been found as insufficient in the course of the Dependability Requirements Definition, it is possible to rework the Dependability Analysis.

The Inputs of this activity are the outputs from the Dependability Analysis.

The Output of this activity are:

- Dependability Requirements including Reliability Requirements, Availability Requirements, Maintainability Requirements, Safety Requirements, Security Requirements, and other necessary additional requirements.

They shall be strictly managed including the additional requirements.

9.3.3 Dependability Argument Construction

In this activity, the Dependability Argument Construction based on the results of Dependability Analysis and Dependability Requirements Definition, the following tasks shall be performed to assure that the system is dependable.

- Construction of Dependability Assurance Cases using templates
- Evaluation of artifacts on their validity of evidence

Also another task shall be performed to evaluate the validity of Dependability Assurance Cases. These are a minimum task set, and the set can be extended if needed. When the existing Dependability Requirements Definition has been found as insufficient in the course of Dependability Argument Construction, it is possible to rework the Dependability Requirements Definition.

The inputs are:

- Dependability Analysis Results.
- Dependability Requirements including the Risk List, the Risk Mitigation Plan, and Dependability Requirements It shall also include a rough System Architecture Model.

These inputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications. The outputs are:

- Dependability Assurance Cases.
- The result of assurance, that is, the revaluation result of the Evidence.

These outputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

9.4 Systems Engineering Process

Basic activities of the Systems Engineering Process are a process to develop systems that are defined in this clause.

The Systems Engineering Process is located in the middle of DPM (Figure 9.2). Its outcome artifacts are used as Evidence to assure dependability of the system in the Dependability Argument Construction. Figure 9.3 illustrates the Systems Engineering Process diagram extracted from Figure 9.2.

In this process, the following activities are performed:

- System Requirements Definition.
- System Architecture Design.
- Concurrent Hardware Development and Software Development which has a sub-process to illustrate the control software development process under it (Figure 9.3).
- Verification & Validation for the integrated outcome of the Hardware Development and the Software Development.

The activities defined here are a minimum set, and the set can be applied combined with necessary extensions for various consumer device developments.

This process may be performed iteratively to develop a system. Several iteration loops may be applied step-by-step to develop the system. Problems found in the previous development loop may be solved in the next development loop.

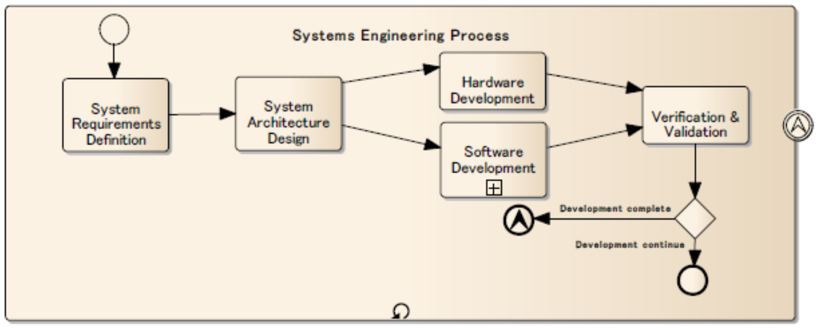

Systems Engineering Process
System Requirements Definition
System Architecture Design
Hardware Development
Software Development
Verification & Validation
Development complete
Development continue

Figure 9.3 - BPMN for Systems Engineering Process

There are a lot more activities in the systems engineering process, but not all are always necessary. These are the required subset.

9.4.1 System Requirements Definition

This activity is the first activity of the Systems Engineering Process. It shall define requirements for the system. This activity corresponds to ‘Stakeholder Requirements Definition Process’ and ‘Requirements Analysis Process’ of ISO15288.

Detailed tasks of the activity are as follows.

Basic requirements shall be clarified. In this task, Dependability Requirements shall be clarified for categories of Reliability Requirements, Availability Requirements, Maintainability Requirements, Safety Requirements and Security Requirements.

As the second task, Use Cases and their Scenarios of associated system behaviors shall be defined to realize the requirements. These are a minimum task set, and the set can be extended if needed. When the existing Dependability Requirements Definition has been found as insufficient in the course of System Requirements Definition, it is possible to rework the Dependability Requirements Definition.

As examples of inputs, they are the following documents:

- Needs from upper layer processes
- Development Plan
- Dependability Requirements

These inputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

As examples of outputs, they are the following documents:

- Requirement Specifications
- Use Case Specifications (Use Cases and Use Case Scenarios)

These outputs are a minimum set, and the set can be applied with necessary extensions and/or modifications.

9.4.2 System Architecture Design

System Architecture shall be designed to realize requirements defined in the System Requirements Definition activity. This activity corresponds to 'Architectural Design' Process of ISO15288.

This activity shall clarify the structure and behavior of the system and subsystems, and identification of components, where subsystems compose the system and components compose the subsystems.

These are a minimum task set, and the set can be extended if needed.

As examples of inputs, they are the following documents:

- Outputs from the System Requirements Definition activity

These inputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications

As examples of outputs, they are the following documents:

- The System Architecture design including the structure and behavior of the system and subsystems, and identification of components.

This output is a minimum set, and the set can be applied with necessary extensions and/or modifications.

9.4.3 Hardware Development

After the results of System Architecture Design, hardware will be developed. Basic tasks shall be design of the hardware, simulation, hardware prototype production and test. This activity was referring to the part of the ‘5 Part of ISO26262: Product development at the hardware level.’

Design of the hardware includes the mechanical design and the circuit design, and it clarifies the specification of the hardware. Simulation verifies correctness of the design. Hardware prototypes are manufactured based on the verified design, and they are tested.

These are a minimum task set, and the set can be extended if needed.

As examples of inputs, they are the following documents:

- System Architecture including components (Hardware candidates)

This input is a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

As examples of outputs, they are the following documents:

- Hardware Specification
- Hardware Prototype
- Test Results

These outputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

9.4.4 Software Development

In this clause we use the terminology of Control Software Development process instead of Control Software Development activity, and also use the terminology of activity instead of task and the terminology of task instead of sub-task, for simplicity.

The process of Control Software Development is a part of the Systems Engineering Process. Control Software Development process is divided into two parts component processes of Control Design Process and Implementation Process.

The Software Development process is carried out in parallel with the Hardware Development process as illustrated in Figure 9.2. In addition, the Software Development process contains the Control Design Process composed of the activities of Requirements Definition, Control Design, Control Modeling, Auto-Coding, and Software Calibration & V&V. It also contains the Implementation Process composed of the activities of Auto Coding, Simplification Optimization, and Code Generation. It is necessary to perform rapidly and iteratively both the Control Design Process and the Implementation Process, improving the accuracy and quality of the control strategy and, at the same time, solving the problem of processing time and memory capacity. The entire Control Software Development process is illustrated in Figure 9.4.

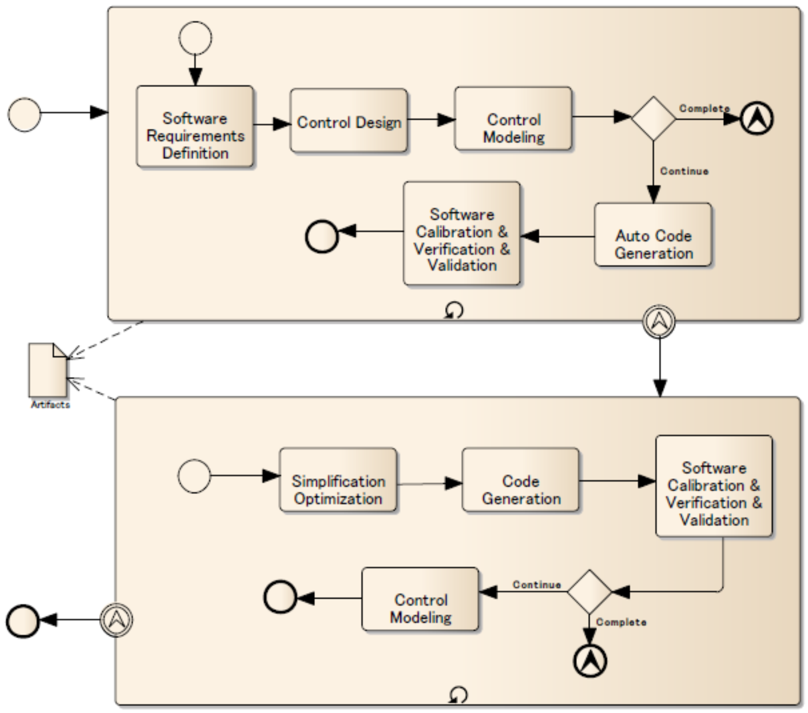

Software Requirements Definition
Control Design
Control Modeling
Complete
Continue
Software Calibration & Verification & Validation
Auto Code Generation
Artifacts
Simplification Optimization
Code Generation
Software Calibration & Verification & Validation
Control Modeling
Continue
Complete

Figure 9.4 - Software Development Process

9.4.4.1 Software Requirements Definition

Software Requirements Definition is an activity to clarify the control software requirement specification. It uses outputs from the System Architecture Design activity.

As examples of inputs, they are the following documents:

- System Requirements
- Dependability Requirements

These inputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

As examples of outputs, they are the following documents:

- Software Requirements

These outputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

9.4.4.2 Control Design

Control Design is an activity to develop Control software Specifications or design. In this activity, details of Software Requirements are analyzed and concrete Control Specifications are described. Primarily it designs functional features.

As examples of inputs, they are the following documents:

- Software Requirements

These inputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

As examples of outputs, they are the following documents:

- Control Specifications

These outputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

9.4.4.3 Control Modeling

Control Modeling is an activity to develop a control model and to put it in a simulation-ready status. Control Specifications that are outputs of Control Design are usually described in natural language, but it is necessary to perform rapid and iterative development for improved control accuracy by utilizing MBD (Model Based Development) simulation. Therefore, in this activity, specifications should be described using a modeling language or mathematical models (e.g., Simulink).

As examples of inputs, they are the following documents:

- Control Specifications

These inputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

As examples of outputs, they are the following documents:

- Control Software Models

These outputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

9.4.4.4 Auto Code Generation

The Auto Code Generation activity is to generate implementation code using an automatic implementation code generation system. Control models developed by the Control Modeling activity are converted into program code (C-code for example) and also into executable code to be implemented (installed) in the target CPU and memory, using a software build environment.

As examples of inputs, they are the following documents:

- Control Software Models

These inputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

As examples of outputs, they are the following documents:

- Implementation Codes that is generated automatically

These outputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

9.4.4.5 Simplification Optimization

The Simplification Optimization activity is to simplify and/or optimize the control logic and model of the Control Software. The Control Model implemented in Auto Code Generation may be generated as a Control Model with redundancy and functions containing constraints to be improved. Therefore, an implementation to achieve equivalent quality of the control model is deployed with simplified and/or optimized approach for codes, considering constraints of implementation size and cost. This activity is to achieve final implementable Control Models using mathematical methods to tune simplicity and optimization.

As examples of inputs, they are the following documents:

- Control Software Models

These inputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

As examples of outputs, they are the following documents:

- Simplified and /or optimized Control Software Models

These outputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

9.4.4.6 Code Generation

The Code Generation activity is to generate Implementation Code. In this activity source code to be implemented in the target CPU is generated. It generates actually implemented Code as opposed to Auto-Coding results which sometimes cannot be flashed into the target CPU because CPU size constraints and poor failure mode implementation. The auto-coded code is going to be further manually optimized for size or further manually enhanced with additional failure mode implementation. It is necessary to equip dedicated compilers and/or build- environments appropriately and to manage code quality and code size precisely for generating code conforming to the target CPU and memory specification.

As examples of inputs, they are the following documents:

- Simplified and /or optimized Control Software Models

These inputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

As examples of outputs, they are the following documents:

- Implementation Code based on simplified and /or optimized Control Software Models

These outputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

9.4.4.7 Software Calibration & Verification & Validation

The Software Calibration & Verification & Validation activity is to tune up software parameters and to verify the correctness of developed Control software.

The final Implementation Code, i.e., the combination of Control Software logic and tuned parameters, is verified in order to achieve required control.

Problems and defects found in the course of the Calibration & V & V activity require cause analysis. If the cause exists in the Control Requirements, the Requirements will be modified, and the Control Software Development process will be performed again. Further, the Implementation sub-process will be performed again. Accuracy of the Implementation Code is improved, and when the results of Verification are judged to be good, the Implementation Code becomes the final Implementation Code.

As examples of inputs, they are the following documents:

- Implementation Code based on simplified and /or optimized Control Software Models

These inputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

As examples of outputs, they are the following documents:

- Validated parameters

- Verification results
- Final Implementation Code

These inputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

9.4.4.8 Rapid and Iterative Loops

The entire Control Software Development process has three Loops. The first Loop is the Control Design Loop. The Loop activities are Requirements Definition, Control Design, Control Modeling, Auto-Coding, and Software Calibration & V&V.

The second Loop is the Implementation Loop. The Loop activities are Auto Coding, Simplification Optimization, and Code Generation.

The third and last Loop is the Entire Loop. The Loop activities are Software Requirements Definition, Control Design, Control Modeling, Simplification Optimization, Code Generation, and Software Calibration & Verification & Validation.

9.4.5 Verification & Validation

After the results of the Hardware Development and the Software Development, integration of Hardware items and Software items, and the Verification and Validation activity shall be performed. This activity corresponds to 'Integration Process,' 'Verification Process,' 'Transition Process,' and 'Validation Process' of ISO15288.

Firstly, the integrated system will be verified if it meets with the system specification. Secondly, validity of the system is evaluated apart from the fact that it is verified successfully. These verifications and validations shall be performed at the component level, the subsystem level, and the system level. This is a minimum task set, and the set can be extended if needed.

As examples of inputs, they are the following documents:

- System Specification (System Architecture)
- Hardware products
- Software code

These inputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

As examples of outputs, they are the following documents:

- Verification and Validation results for each level.
- The system as the output of the corresponding development loop.

These outputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

9.5 Evolutionary Development Process

This clause defines activities corresponding to the derivational development or the Evolutionary Development Process.

When functions are added and/or modified for the system in the Operation phase after the initial development, these activities shall be performed.

Firstly, the Difference Analysis activity shall be performed and secondly the Impact Analysis activity shall be performed.

9.5.1 Difference Analysis

This activity is to analyze what is to be changed. It contains tasks to clarify change requests, to identify subsystems and/or components related to these change requests, and to define necessary system modifications.

This is a minimum task set, and the set can be extended if needed.

As examples of inputs, they are the following documents:

- Upper level plans such as the Product Plan

These inputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications. An example of such extensions may be a document describing abstract instructions for new functions and/or modifications.

As examples of outputs, they are the following documents:

- Difference Analysis Results describing subsystems and components to be modified

These outputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

9.5.2 Impact Analysis

This activity is to analyze impacts of the planned changes. It shall contain tasks to clarify which subsystems and/or components are impacted by the planned changes, and how they are impacted. This is a minimum task set, and the set can be extended if needed.

As examples of inputs, they are the following documents:

- Difference Analysis Results

These inputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

As examples of outputs, they are the following documents:

- Impact Analysis Results describing impact details and scope of affected subsystems and components.

These outputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

9.6 Etcetera Process

In this clause, activities which are not contained in the Dependability Process, the Systems Engineering Process and the Evolutionary Development Process are defined as Etcetera Process activities. When the Systems Engineering Process has been completed, the system shall go into the Operation phase, and when the Operation has been stopped and the lifecycle of the product is to be closed, the system shall go into the Disposal phase (Figure 9.2).

9.6.1 Operation

When the system development has been completed, the system goes into the Operation phase. If system problems are found in the Operation phase, they are reported and require necessary modification. Minor problems do not require stoppage of the Operation of the system, and the modified system will continue its operation under the new conditions.

This activity corresponds to ‘Operation Process’ and ‘Maintenance Process’ of ISO15288. This is a minimum task set, and the set can be extended if needed. When derivational development is applied and a new product model has been developed, several product models may be operated concurrently before the closure of the lifecycle of the old product model.

As examples of inputs, they are the following documents:

- Verification and Validation Results
- The system to be operated and maintained

These inputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

As examples of outputs, they are the following documents:

- Incident Reports for found problems

These outputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

9.6.2 Disposal

When the product lifecycle has been closed, the system will go into the Disposal phase. The legally required disposal procedures shall be performed as defined legally. Reusable resources should be processed to be reused properly. This activity corresponds to ‘Disposal Process’ of ISO15288. These are a minimum task set, and the set can be extended if needed.

As examples of inputs, they are the following documents:

- The system for which the disposal has been planned

This input is a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

As examples of outputs, they are the following documents:

- Reusable resources if they exist

These outputs are a minimum set, and the set can be applied combined with necessary extensions and/or modifications.

This page intentionally left blank.

Annex A: Bibliography

[1] ISO26262: Road vehicles-Functional safety-2011

[2] A. Avizienis, J.-C. Laprie, B. Randell and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, vol. 1, pp. 11-33, 2004

[3] IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems - 2010

[4] ISO/IEC 15408: Common Criteria for Information Technology Security Evaluation 2012

[5] ISA-62443-3-3: Security for industrial automation and control systems, Part 3-3: System security requirements and security levels Draft 4, Jan 2013

[6] SWEBOK: Guide to the Software Engineering Body of Knowledge, IEEE, 2004

This page intentionally left blank.