Date: Sep, 2015

OBJECT MANAGEMENT GROUP

Dependability Assurance Framework for Safety-Sensitive Consumer Devices (SSCD) Specification

Version Beta 2~~1~~

OMG Document Number: ptc/2015-09-11~~8-01~~~~sysa/2015-08-24~~

Normative reference: http://www.omg.org/spec/DAF~~SSCD~~
Normative Machine ~~readable~~ Consumable files:
ptc/2015-08-03: SSCD DAC Template SACM xmi machine readable files
~~sysa/2014-11-02: http://www.omg.org/spec/SSCD/20141101/SSCD-DAC-Templaes-SACM-XMIfiles.zip~~
ptc/2015-08-04: SSCD DCM xmi file
~~sysa/2014-11-03: http://www.omg.org/spec/SSCD/20141101/SSCD-DCM.xmi~~
ptc/2015-08-05: SSCD DPM.xmi file
~~sysa/2014-11-04: http://www.omg.org/spec/SSCD/20141101/SSCD-DPM.xmi~~

Non-normative Machine Consumable~~readable~~ files:
ptc/2015-08-06: SSCD DCM Astah UML file
~~sysa/2014-11-05: http://www.omg.org/spec/SSCD/20141101//SSCD_DCM.asta~~

Submitters:

National Institute of Advanced Industrial Science and Technology (AIST)

Fujitsu Limited

Information-technology Promotion Agency, Japan (IPA)

Toyota Motor Corporation

Supporter:

The University of Electro-Communications

Formatted: Font: (Default) +Headings (Ari
Formatted: Font: (Default) +Headings (Ari
Formatted: Font: (Default) +Headings (Ari
Formatted: Font color: Red
Formatted: Font: (Default) +Headings (Ari
Font color: Red
Formatted: Font: (Default) +Headings (Ari
Font color: Red
Formatted: Font: (Default) +Headings (Ari
Formatted: No underline, Font color: Auto
Formatted: Font: (Default) +Headings (Ari
Formatted: Font color: Auto
Formatted: Font: (Default) +Headings (Ari
Formatted: Font: (Default) +Headings (Ari
Formatted: Font color: Red
Formatted: Font: (Default) +Headings (Ari
Font color: Red
Formatted: Font color: Red
Formatted: Font: (Default) +Headings (Ari

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights Clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 109 Highland Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

IMM®, MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™ , OMG Interface Definition Language (IDL)™ , and OMG SysML™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG'S ISSUE REPORTING PROCEDURE

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue (http://www.omg.org/report_issue.htm.)

Table of Contents

Formatted: Default Paragraph Font, Font: (Default) Century, Bold, Check spelling and grammar

Formatted: Default Paragraph Font, Font: Bold, Check spelling and grammar

Formatted: Default Paragraph Font, Font: Bold, Check spelling and grammar

Formatted

Formatted

Formatted

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

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 Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
109 Highland Ave
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 clause 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 to http://www.omg.org/report_issue.htm.

1 Scope

The objective of this document is to provide a new system assurance methodology for the dependability argumentation~~methodology of the dependability argumentation~~ for consumer devices, which is achieved by integrating conventional system assurance approaches~~integrating the conventional system assurance approach~~ 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~~as we are not experts in of other systems~~ rather than consumer devices.

Comment [A1]: DAF#18

Comment [A2]: DAF#18

Comment [A3]: DAF#18

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 ~~the~~ 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.

~~In this Dependability Assurance Framework for Safety-Sensitive Consumer Devices (SSCD) Specification, suggested conformance points are Dependability Conceptual Model, Dependability Assurance Case, and Dependability Process Model. DAC templates shall confirm to SACM.~~

~~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 the clause 8.~~

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

Comment [A4]: DAF#18

Comment [A5]: DAF#18

Comment [A6]: DAF#10

Name of Model	Clause Number	Requirement for conformance
Dependability Conceptual Model	Clause 7	Each class defined in ~~the~~ 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 a 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 contains 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/)

Comment [A7]: DAF#18

Comment [A8]: DAF#18

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~~Additional Information~~

Comment [A9]: DAF#11

6.1 Introduction~~Background and Rationale~~

Comment [A10]: DAF#11

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~~General Introduction to Dependability Assurance Framework (DAF)~~

Comment [A11]: DAF#11

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~~Now, given the fact that SSCD is a system to work for certain function, which aggregates systems to individually work for certain different function~~, 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.

Comment [A12]: DAF#26

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.

~~6.3~~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.

~~6.4~~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 a SSCD according to the dependability process.

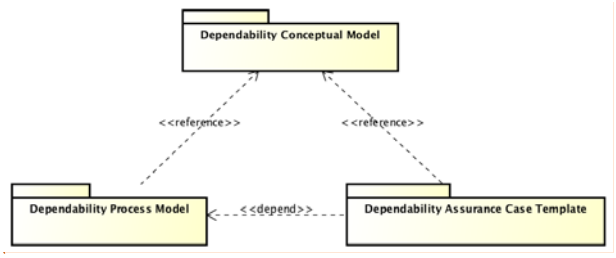


Figure 6-1 – Dependability Assurance Framework

~~6.5~~ Changes to Adapted OMG Specifications

~~No changes to adopted OMG Specifications are made in this specification.~~

Comment [A13]: DAF#11
Formatted: Heading 3
Comment [A14]: DAF#11
Formatted: Heading 3
Formatted: Centered
Comment [A15]: DAF#9
Comment [A16]: DAF#11

~~6.6~~6.2.3 How to Read this Specifications

Comment [A17]: DAF#11

Formatted: Heading 3

Users who want to apply this specification to their own products may want to read the procedure in Clause 6.4 first.

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.

7 Dependability Conceptual Model (DCM)

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, our semantic models are provided at an abstract level where several specifications/standards can be accommodated and harmonized.

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.

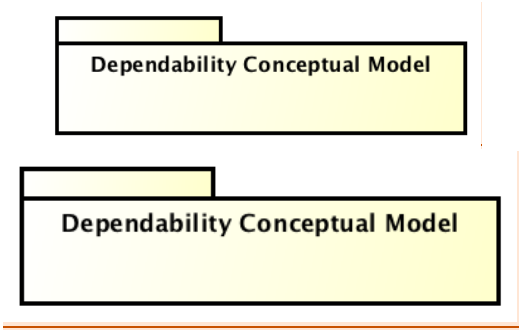


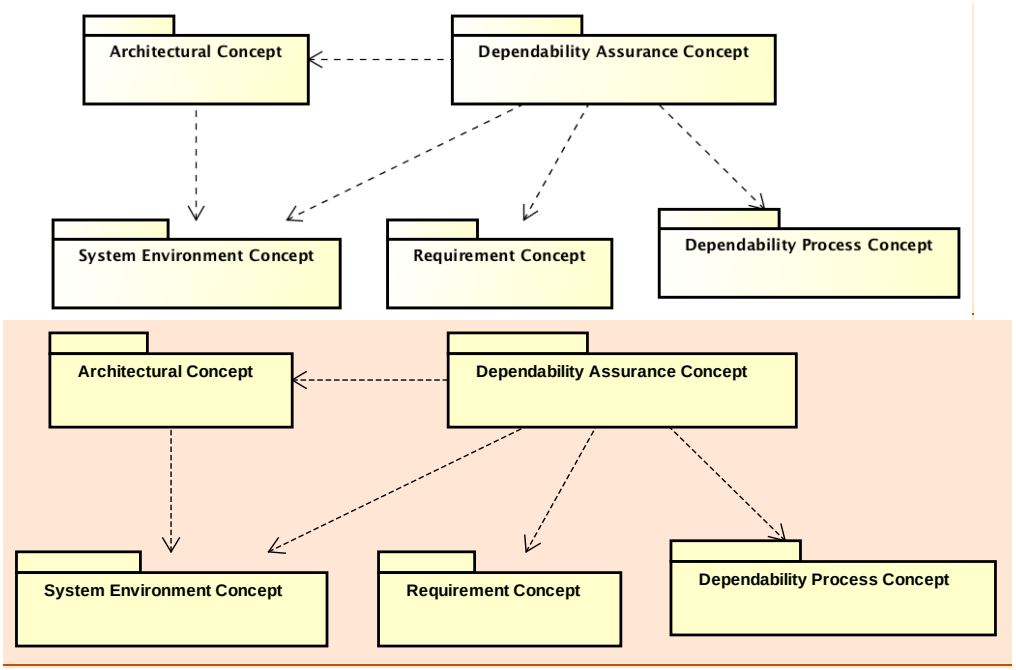
Figure 7-1 – Dependability Conceptual Model

Comment [A18]: DAF#9

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.



Formatted: Centered

Figure 7-2 –Dependability Conceptual Model package

Comment [A19]: DAF#9

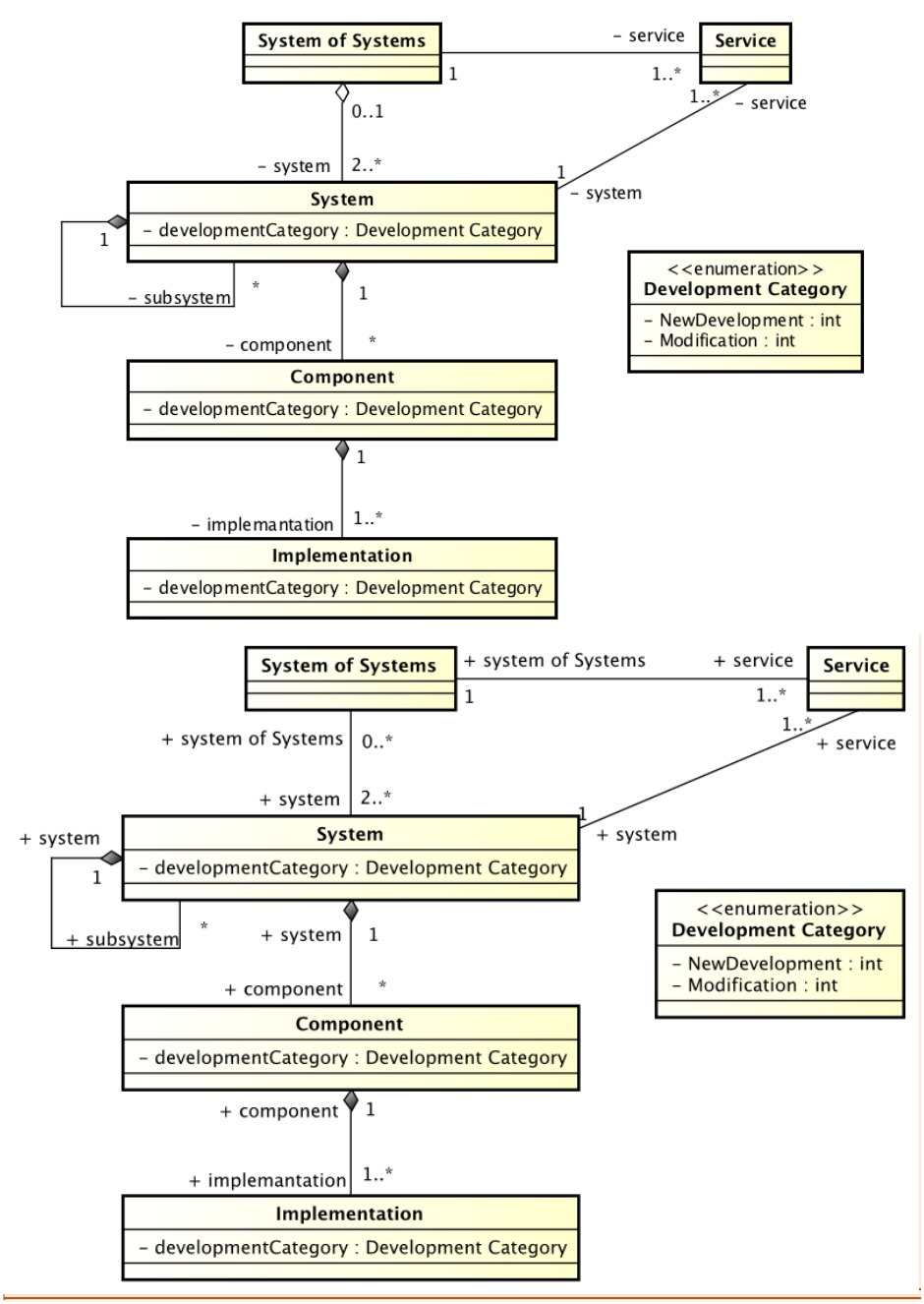
The UML (Unified Modeling Language) class diagrams are used throughout in this specification. Concepts in the Dependability Conceptual Model are modularized topically to separate concerns and to facilitate understanding by developers. These packages are interrelated, but independently specified in this 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.

7.1 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.

Formatted: Centered



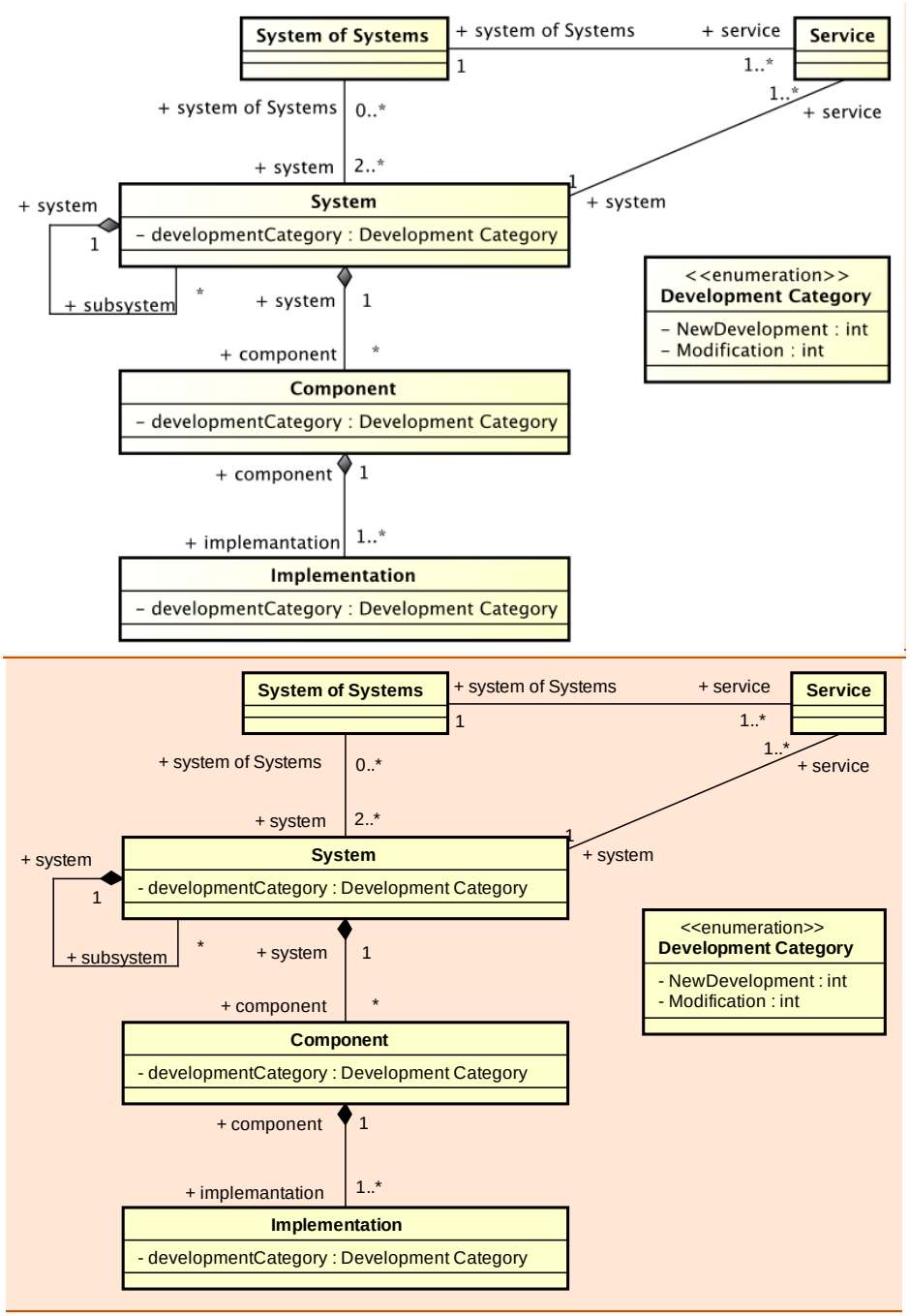


Figure 7-3 – Architectural Concept

Comment [A20]: DAF#3, DAF#9

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.

7.1.1 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.

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

Comment [A21]: DAF#4

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

system: System[2..*]

System of Systems is composed of two or more systems.~~two more systems.~~

Comment [A22]: DAF#4

service: Service[1..*]

System of Systems can provide one or~~e~~ more services.

7.1.2 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 by~~composed of subsystems. If subsystems are necessary, the System is ~~composed by~~composed of subsystems, and the subsystem is ~~composed by~~composed of components.

Comment [A23]: DAF#18

Comment [A24]: DAF#18

~~Combination of interacting elements organized to achieve one or more stated purposes. (From ISO/IEC15288)~~

Comment [A25]: DAF#18

Comment [A26]: DAF#18

Generalizations

No additional generalizations

Attributes

developmentCategory: DevelopmentCategory

It is distinguished by new development or modification.

Associations

subsystem: System[*]

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

Comment [A27]: DAF#18

component: Component[*]

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

Comment [A28]: DAF#18

service: Service[*]

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

7.1.3 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.~~It is not almost a single Hardware and/or Software item, but compose several items of Hardware and/or Software.~~

Comment [A29]: DAF#12

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

implementation: Implementation[1..*]

The Component is ~~composed by~~composed of one or more implementations.

Comment [A30]: DAF#18

7.1.4 Implementation

Description

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

Comment [A31]: DAF#18

Generalizations

No additional generalizations

Attributes

No additional attributes
Associations
No additional associations

7.1.5 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

7.1.6 Development Category

Description
The Development Category provides the distinction between new development of a system and modification-~~development~~ 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.

Comment [A32]: DAF#18

Comment [A33]: DAF#18

7.2 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~~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

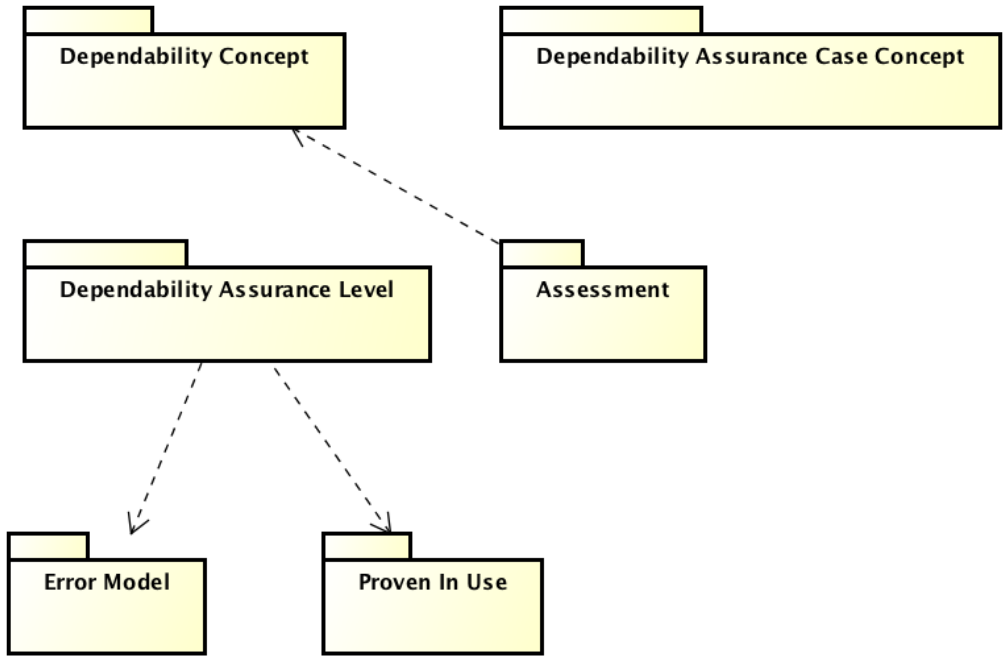


Figure 7-4 – Dependability Assurance Concept package

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.

7.2.1 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

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: Centered

Comment [A34]: DAF#9

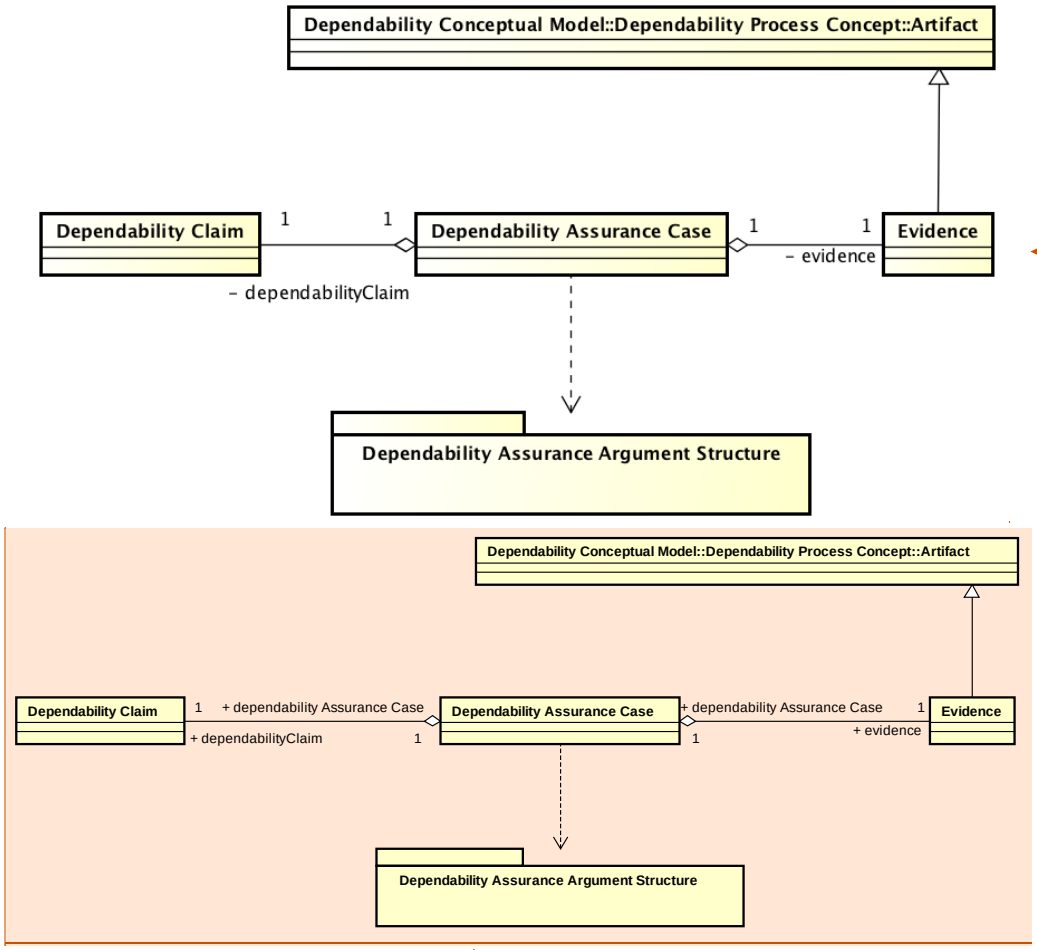


Figure 7-5 – Dependability Assurance Case Concept

7.2.1.1 Dependability Claim

Description

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

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.2.1.2 Evidence

Description

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

Generalizations

Formatted: Centered

Comment [A35]: DAF#9

Comment [A36]: DAF#18

Comment [A37]: DAF#18

No additional generalizations
Attributes
No additional attributes
Associations
No additional associations

7.2.1.3 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]

7.2.1.4 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

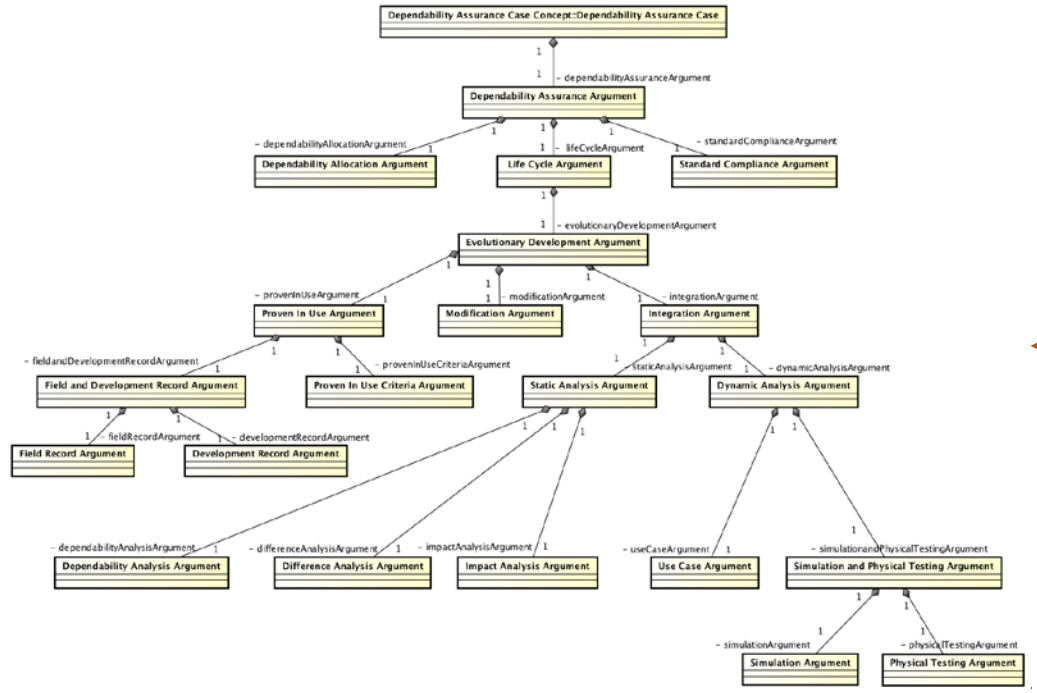


Figure 7-6 – Dependability Assurance Argument Structure

7.2.1.5 Dependability Assurance Argument Description

Formatted: Don't keep with next

Comment [A38]: DAF#9

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

7.2.1.6 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

7.2.1.7 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

7.2.1.8 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.

Comment [A39]: DAF#18

Comment [A40]: DAF#18

Comment [A41]: DAF#18

Comment [A42]: DAF#18

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

evolutionaryDevelopmentArgument: Evolutionary Development Argument[1]

7.2.1.9 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.

Comment [A43]: DAF#18

Comment [A44]: DAF#18

Comment [A45]: DAF#18

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

provenInUseArgument: Proven In Use Argument[1]
modificationArgument: Modification Argument[1]
integrationArgument: I-ntegration Argument[1]
nextGenerationArgument: Next Generation Argument[1]

7.2.1.10 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~~none~~ or more development argument structures for the modified and impacted parts.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.2.1.11 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]

7.2.1.12 Proven In Use Criteria Argument

Description

The Proven In Use Criteria Argument class represents the argument structure ~~for~~ 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

7.2.1.13 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]

Comment [A46]: DAF#18

Comment [A47]: DAF#18

Comment [A48]: DAF#18

Comment [A49]: DAF#18

Comment [A50]: DAF#18

Comment [A51]: DAF#18

Comment [A52]: DAF#18

7.2.1.14 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 adequately enough for the Proven In Use argument in the operation.

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

No additional associations

7.2.1.15 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

7.2.1.16 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]

7.2.1.17 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]

7.2.1.18 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

Comment [A53]: DAF#18

Comment [A54]: DAF#18

Comment [A55]: DAF#18

Comment [A56]: DAF#18

Attributes
No additional attributes
Associations
No additional associations

7.2.1.19 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

7.2.1.20 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

7.2.1.21 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]

7.2.1.22 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

7.2.1.23 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.

Comment [A57]: DAF#18
Comment [A58]: DAF#18
Comment [A59]: DAF#18
Comment [A60]: DAF#18
Comment [A61]: DAF#18

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.

7.2.1.24 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

Comment [A62]: DAF#18

7.2.1.25 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

7.2.2 Dependability Concept
As given in Figure 7-7~~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.

Formatted: Font: (Default) Times New Roman, 10 pt, Font color: Custom Color(RGB(34,34,34)), Pattern: Clear (White

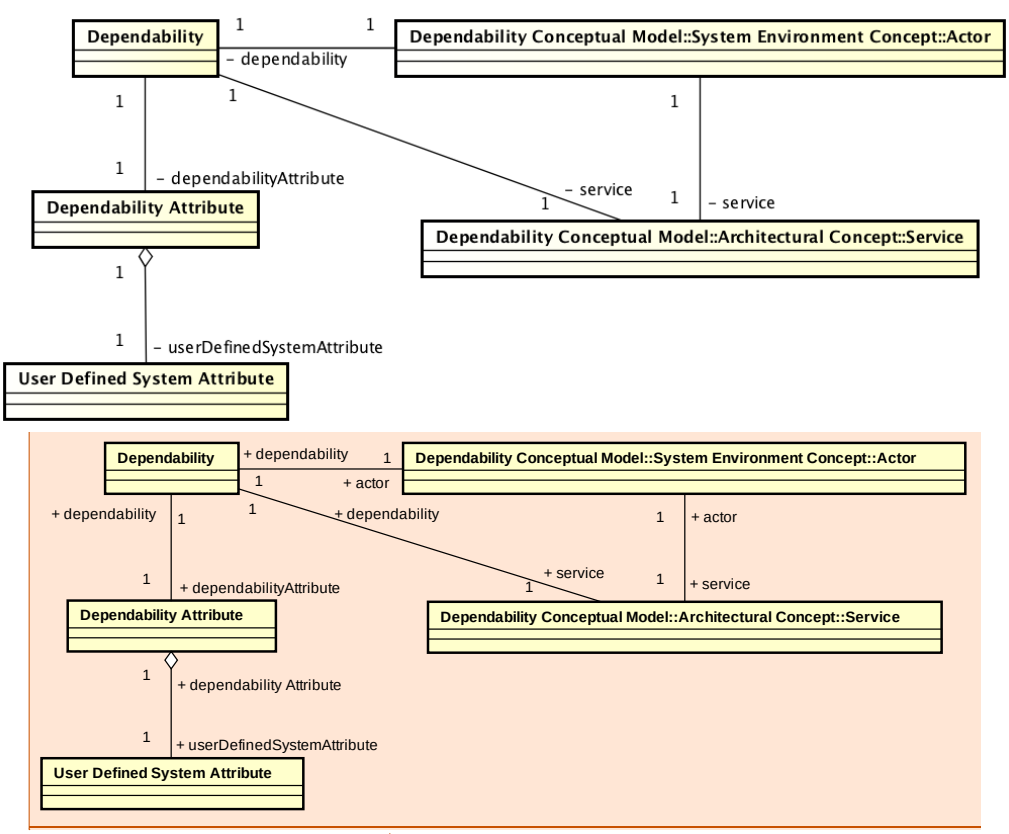


Figure 7-7 – Dependability Concept package

7.2.2.1 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.

7.2.2.2 Dependability Attribute

Description

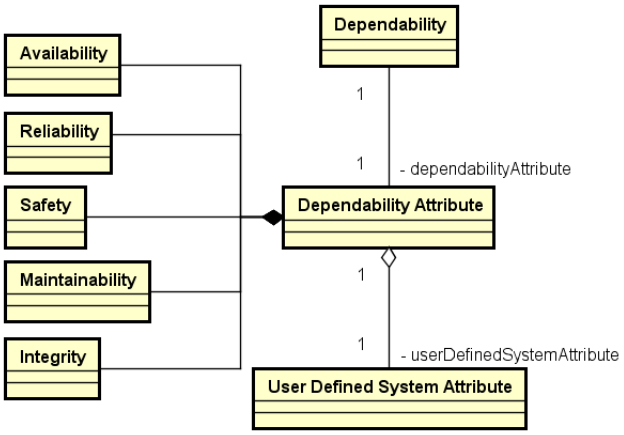
Dependability Attribute is an anchor point where any specific notion of dependability of a particular specification/standard under this SSCD specification could be defined particularly specifying system attributes by which the dependability in that specification/standard is defined.

Formatted: Centered

Comment [A63]: DAF#9

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 the Figure 7-8~~Figure 7-8~~ below where five system attributes; Availability, Reliability Safety, Maintainability and Integrity are defined as essential constituents of the dependability attribute.



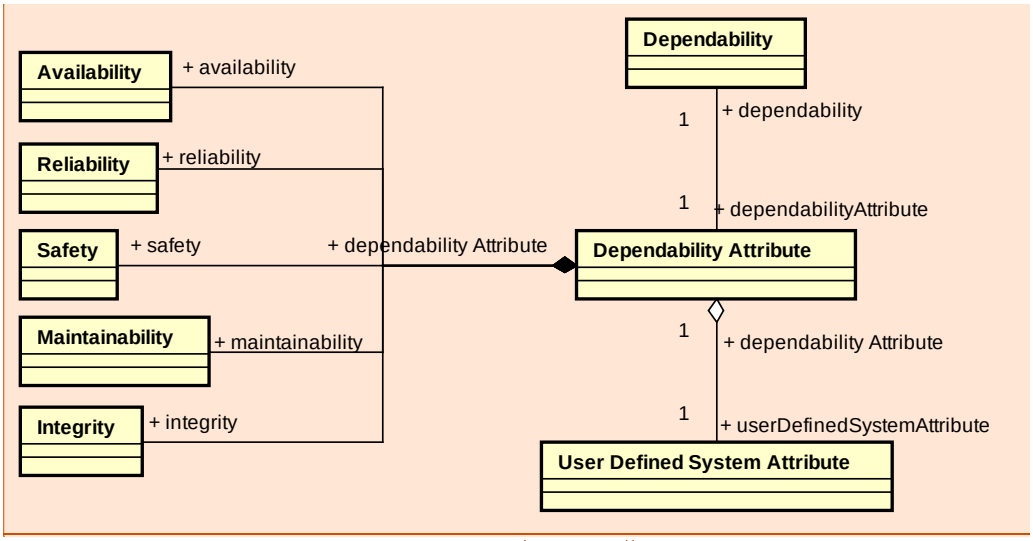


Figure 7-8 – Sample extension of Dependability Attribute (Informative)

7.2.2.3 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.

Formatted: Font: (Default) Times New Roman, 10 pt, Pattern: Clear (White)

Comment [A64]: DAF#13

Comment [A65]: DAF#18

Generalizations
No additional generalizations
Attributes
No additional attributes
Associations
No additional associations

NOTE: User Defined System Attribute is a part of the Dependability Attribute.

Comment [A66]: DAF#18

7.2.3 Dependability Assurance Level

The package in Figure 7-9~~Figure 7-9~~ accommodates all the notions as to how threat is assessed in the specification.

Formatted: Font: (Default) Times New Roman, 10 pt, Pattern: Clear (White)

Formatted: Centered

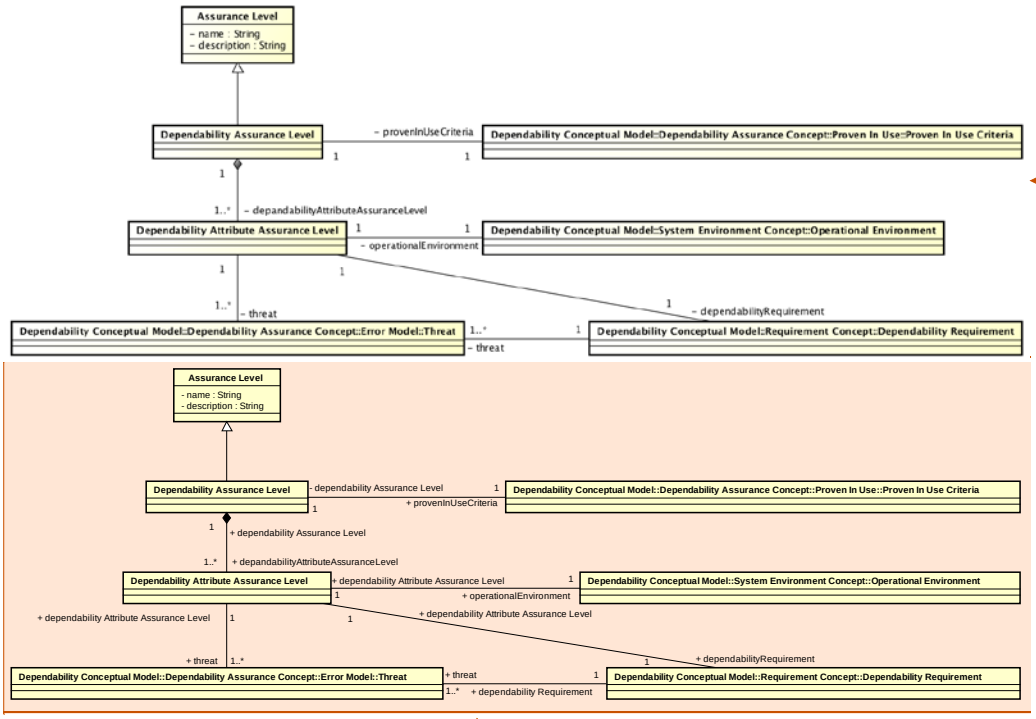


Figure 7-9 – Dependability Assurance Level Package

Comment [A67]: DAF#9

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~~is using~~ 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.

Comment [A68]: DAF#18

Dependability Assurance Level in Figure 7-9~~Figure 7-9~~ is the top concept for assessing threats in SSCDs (Please refer to the Error Model Package in Figure 7-12~~Figure 7-12~~ for the exact meaning of the Threat). Dependability Attribute Assurance Level corresponds to any particular system attribute which composes ~~of~~ the dependability concept of a target system. For instance, safety assurance level in functional safety standards is a sub-notion of Dependability Attribute

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: Font: (Default) Times New Roman, 10 pt

Comment [A69]: DAF#18

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.

Comment [A70]: DAF#18

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.

7.2.3.1 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

7.2.3.2 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.

7.2.3.3 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.

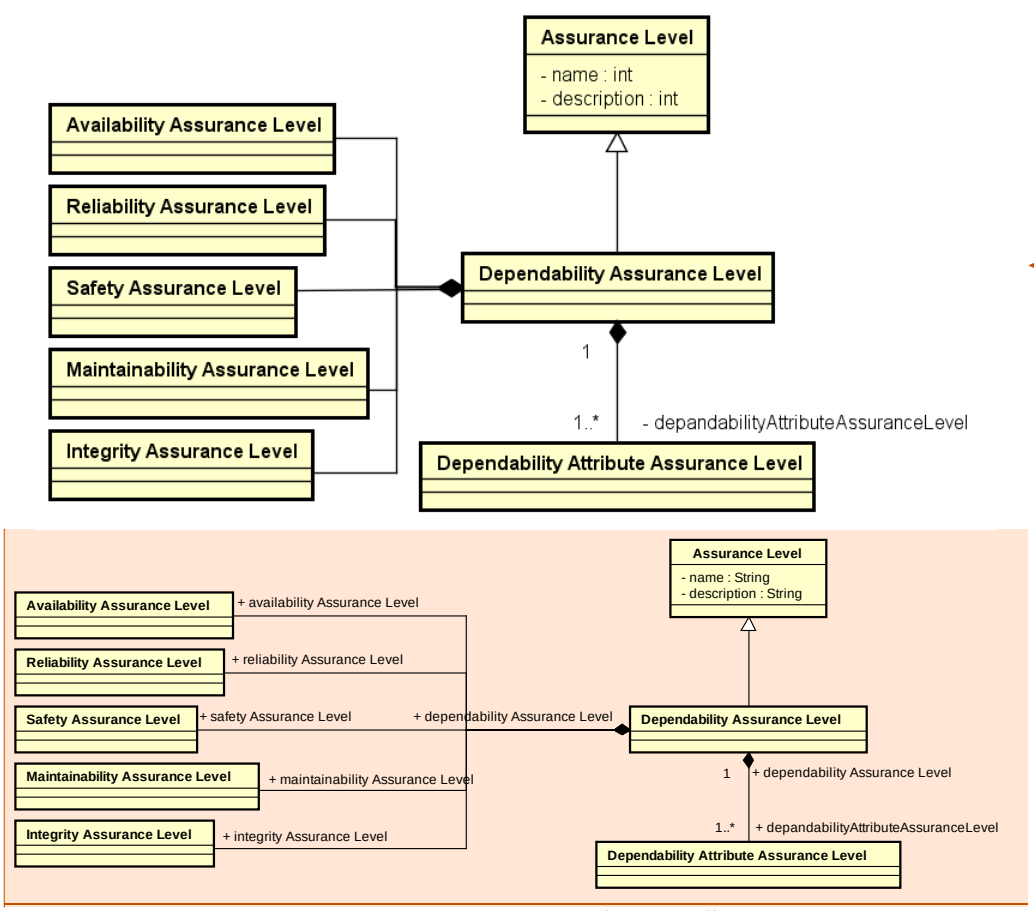


Figure 7-10 – Sample of Dependability Attribute Assurance Levels (Informative)

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~~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.

Formatted: Centered

Comment [A71]: DAF#13

Comment [A72]: DAF#9

Formatted: Font: (Default) Times New Roman, 10 pt, Font color: Custom Color(RGB(34,34,34)), Pattern: Clear (White

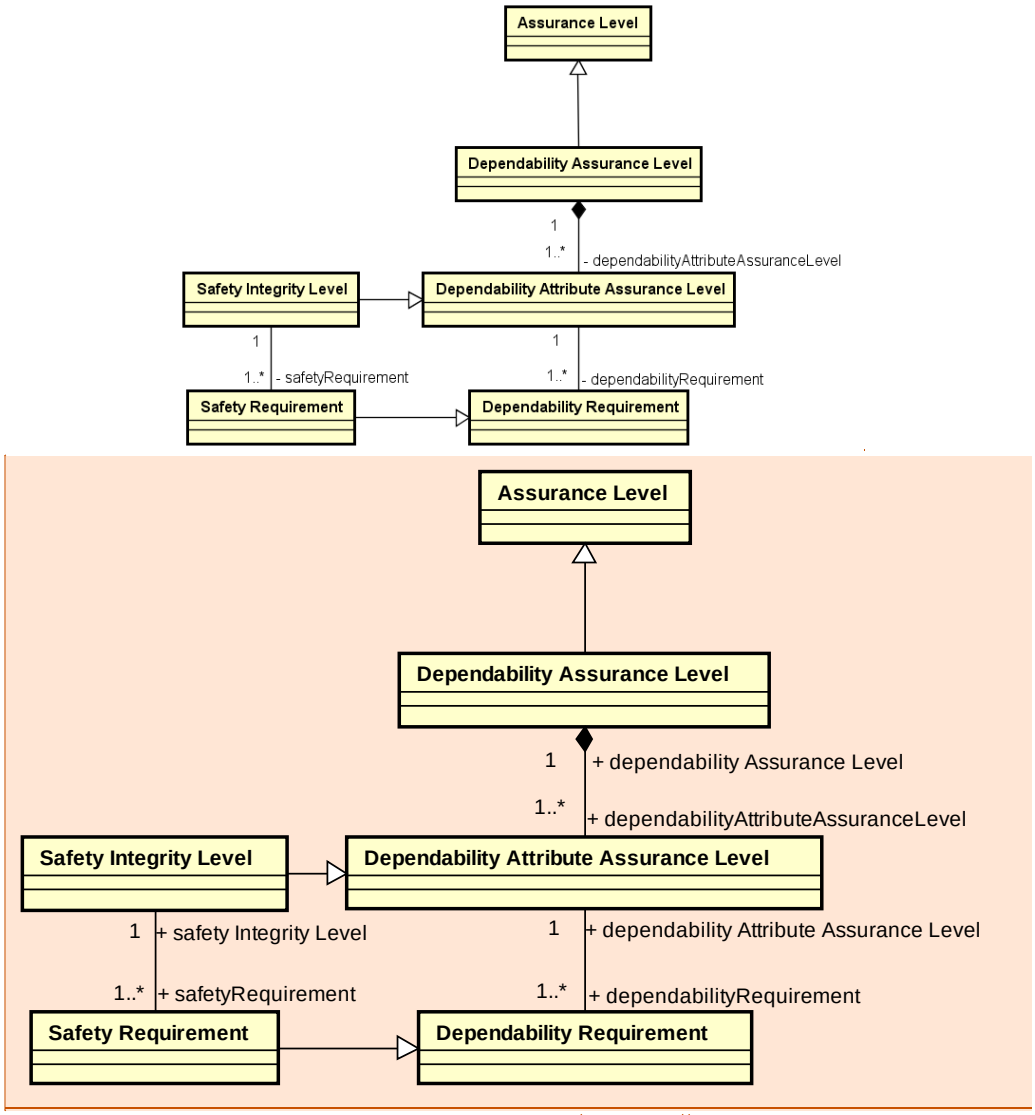


Figure 7-11 – Sample of SIL in IEC 61508 in SSCD specification (Informative)

Comment [A73]: DAF#13

Comment [A74]: DAF#9

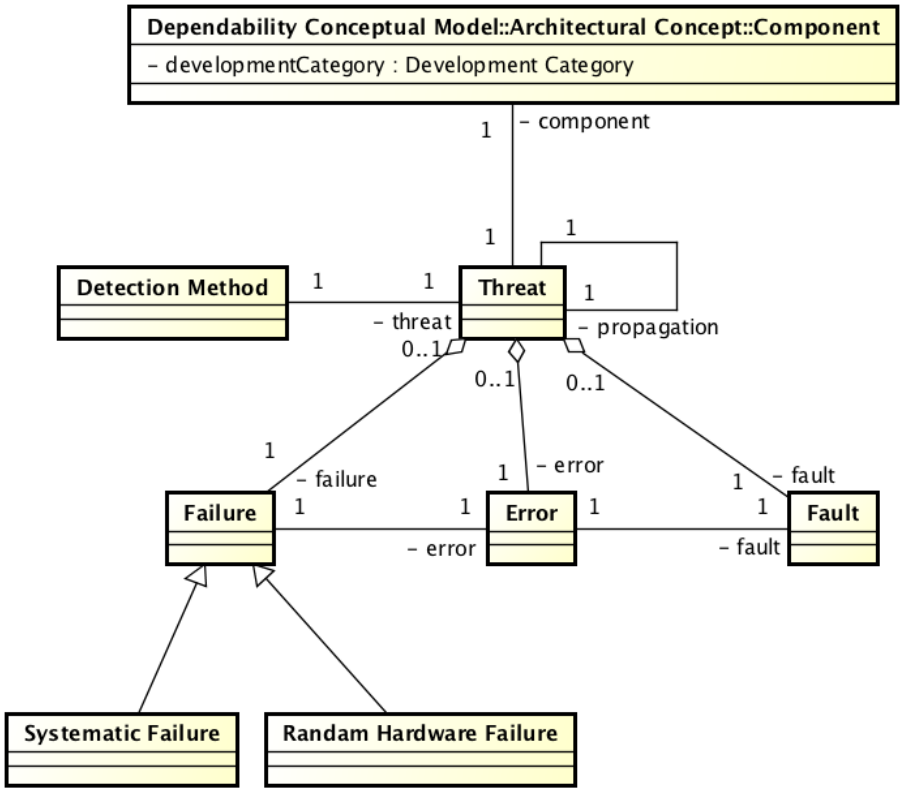
7.2.4 Error Model

This Clause specifies the semantic model for the Error Model. This model in Figure 7-12~~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~~bring the~~ consistency.

Formatted: Font: (Default) Times New Ro

Comment [A75]: DAF#18

Comment [A76]: DAF#18



Formatted: Centered, Don't keep with next

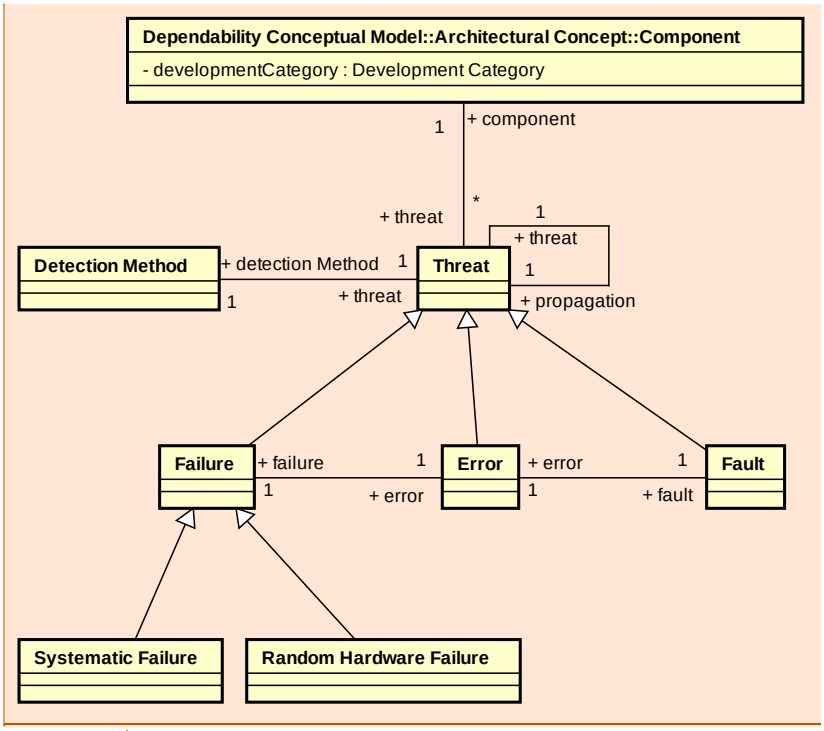


Figure 7-12 – Error Model

Comment [A77]: DAF#9, DAF#29

7.2.4.1 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].

~~failure:Failure~~

~~Specifies that threat is an abstracted notion of failure[1].~~

~~error:Error~~

~~Specifies that threat is an abstracted notion of error[1].~~

~~fault:Fault~~

~~Specifies that threat is an abstracted notion of fault[1].~~

Comment [A78]: DAF#29

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].

Comment [A79]: DAF#29

7.2.4.2 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].

7.2.4.3 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

7.2.4.4 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.

Comment [A80]: DAF#18

Generalizations

No additional generalization

Attributes

No additional attributes

Associations

No additional associations

7.2.4.5 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.

Comment [A81]: DAF#18

Comment [A82]: DAF#18

Generalizations

No additional generalization

Attributes

No additional attributes

Associations

fault: Fault

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

7.2.4.6 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

7.2.4.7 Detection Method

Description

Detection Method is a method to identify a Threat.

Generalizations
No additional generalization
Attributes
No additional attributes
Associations
No additional associations

7.2.5 Assessment

The Assessment package in Figure 7-13~~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~~d by~~ a Confirmation Measure to assess the degree of dependability to be achieved.

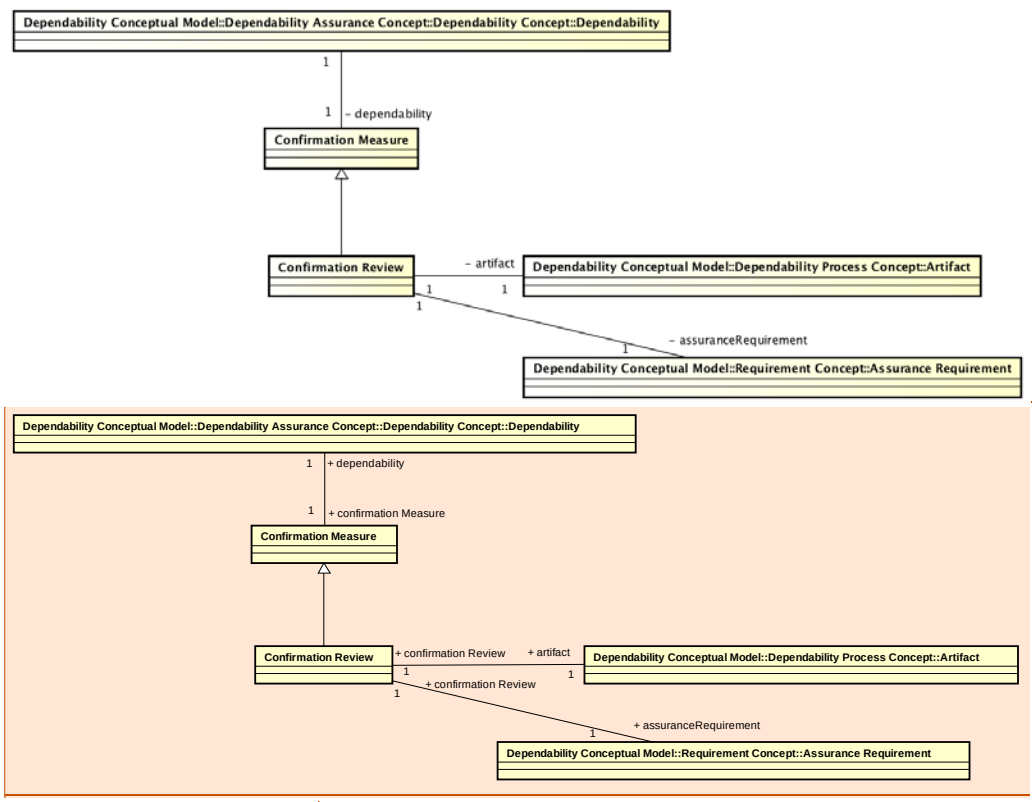


Figure 7-13 – Assessment package

7.2.5.1 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

Formatted: Font: (Default) Times New Roman, 10 pt

Comment [A83]: DAF#18

Comment [A84]: DAF#18

Comment [A85]: DAF#9, DAF#20

Associations

artifact: Artifact [1]

Specifies whether Artifact satisfies the Assurance Requirement.

assuranceRequirement: Assurance Requirement [1]

Specifies the~~at~~ Assurance Requirement is referenced by Confirmation Review.

Comment [A86]: DAF#18

Comment [A87]: DAF#18

7.2.5.2 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

7.2.6 Proven In Use

This package depicted in Figure 7-14~~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.

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: Centered

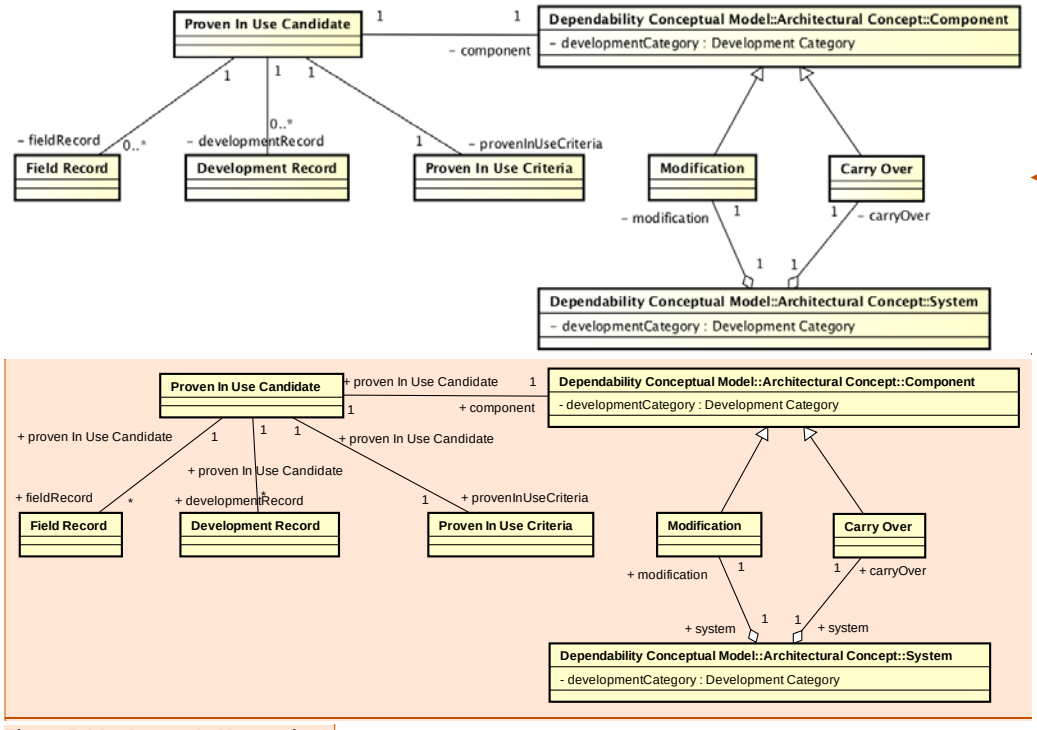


Figure 7-14 – Proven In Use package

Comment [A88]: DAF#9

7.2.6.1 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.

7.2.6.2 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

7.2.6.3 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.

Formatted: Indent: Left: 0", First line: 0"

7.2.6.4 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

~~No additional 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.

Comment [A89]: DAF#20

7.2.6.5 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

~~No additional 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.

7.2.6.6 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

~~No additional 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.

Comment [A90]: DAF#20

Comment [A91]: DAF#20

Formatted: Indent: Left: 0.58", First line: 0.58"

Formatted: Font: 11 pt

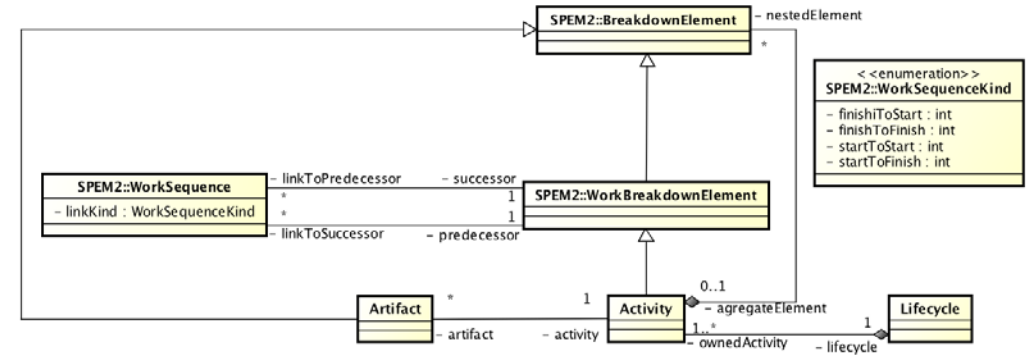
7.3 Dependability Process Concept

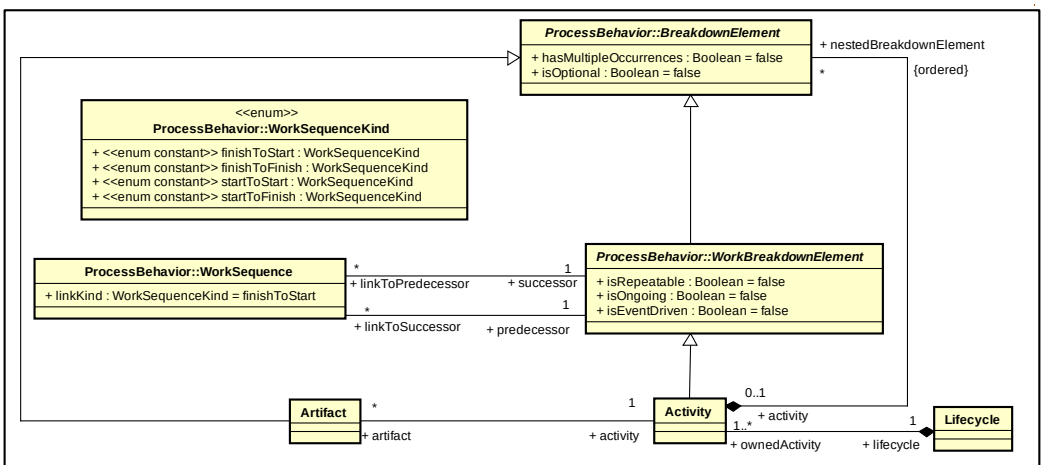
This clause specifies the semantic model for the Dependability Process Model. This model contains the basic structural elements for defining the Dependability Process. The Dependability process is defined in iterations of each Dependability specific process, that is, design systems, simulation and operation, 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.

- Clause 7.3.1 indicates the conceptual model for the Dependability Process.
- The following Clause after Clause 7.3.1 illustrates each constituent element represented in the conceptual model.

7.3.1 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, ~~SPEM2~~SPEM 2.0 is introduced. BreakdownElement, WorkBreakDown Element and Work Sequence are imported from ~~SPEM2~~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 – Dependability Process Model~~Figure 7-15 – Dependability Process Model~~.





Comment [A92]: DAF#18

Comment [A93]: DAF#18

Formatted: Font: (Default) Times New Roman, 10 pt

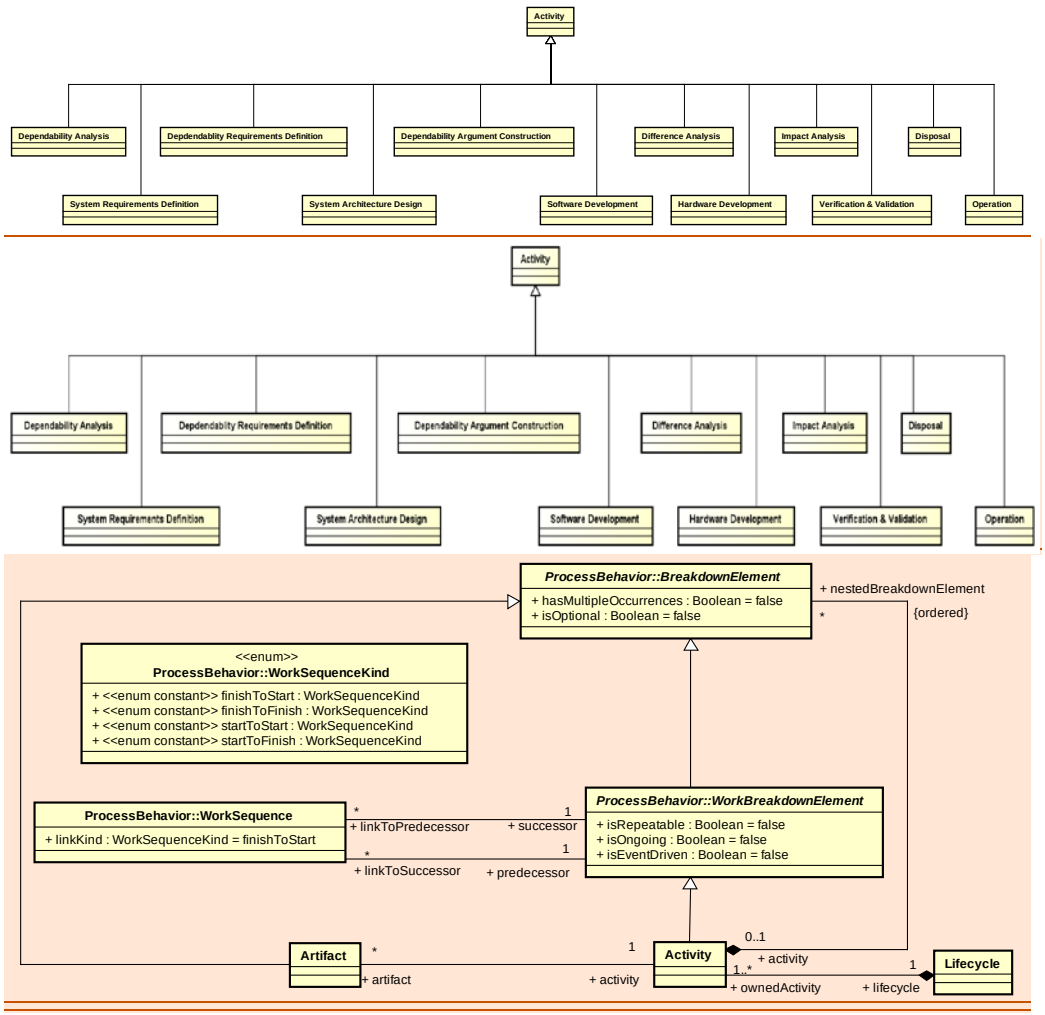


Figure 7-15 – Dependability Process Model

Comment [A94]: DAF#9, DAF#8

7.3.2 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.

Comment [A95]: DAF#18

Comment [A96]: DAF#18

Comment [A97]: DAF#18

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 conceptioal model of process follows SPEM2.0.

Comment [A98]: DAF#31

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.

7.3.3 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.

7.3.4 BreakdownElement (from ~~SPEM2~~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.

Comment [A99]: DAF#18

Comment [A100]: DAF#18

Comment [A101]: DAF#18

Comment [A102]: DAF#18

7.3.5 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

Comment [A103]: DAF#18

7.3.6 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

Comment [A104]: DAF#18

7.3.7 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

7.3.8 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

7.3.9 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

7.3.10 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

7.3.11 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

7.3.12 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.

7.3.13 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

7.3.14 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

7.3.15 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

7.3.16 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

7.3.17 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

7.3.18 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

7.3.19 WorkBreakdownElement (from ~~SPEM2~~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~~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.

7.3.20 WorkSequence (from ~~SPEM2~~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~~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.

7.3.21 WorkSequenceKind (from ~~SPEM2~~SPEM 2.0)

Description

Work Sequence represents a relationship between two Work Breakdown Element 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~~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 finish. For example, if you have two WorkBreakdownElements, "Construct fence" and "Paint

Comment [A105]: DAF#18

Comment [A106]: DAF#18

Comment [A107]: DAF#18

Comment [A108]: DAF#18

Comment [A109]: DAF#18

Comment [A110]: DAF#18

	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.
finshToFinsish	a WorkBreadownwonElement cannot finish until another WorkBreakdownElement finishes. For example, if you have two WorkBreakdownElement, "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 if doesn't matter exactly when and you don't want a late finish to affect the just-in-time WorkBreakdownElement, you can create an the dependency between the WorkBreakdownElement you want scheduled just in time (the predecessor) and its related WorkBreakdownElement (thesuccessor). Then, if you update progress on the successor WorkBreakdownElement, it won't affect the scheduled dates of the predecessor WorkBreakdownElement.

Comment [A111]: DAF#18

Comment [A112]: DAF#18

Comment [A113]: DAF#18

Comment [A114]: DAF#18

7.4 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 paragraph.

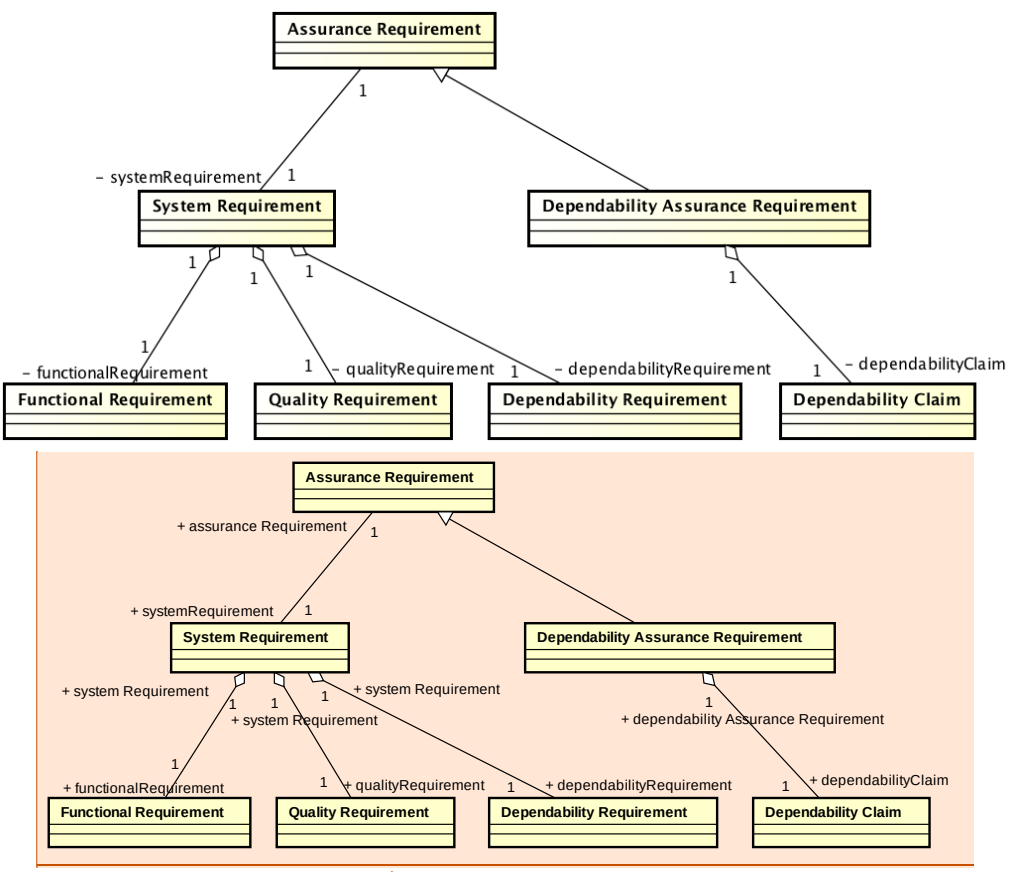


Figure 7-16 – Requirement Concept package

7.4.1 Assurance Requirement

Description

The Assurance Requirement specifies requirements related to system assurance.

Generalizations

No additional generalizations

Attributes

No additional attributes

Formatted: Centered

Comment [A115]: DAF#9

Associations

systemRequirement: System Requirement [1]

Specifies that System Requirement is used in Assurance Requirement.

7.4.2 System Requirement

Description

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

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

functionalRequirement: Functiona 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.

7.4.3 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

7.4.4 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

7.4.5 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.

7.4.6 Dependability Assurance Requirement

Description

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

Generalizations

Assurance Requirement specified in 7.4.1.

Attributes

No additional attributes

Associations

dependbilityClaim: Dependability Claim[1]

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

7.4.7 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.

Comment [A116]: DAF#20

7.5 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.

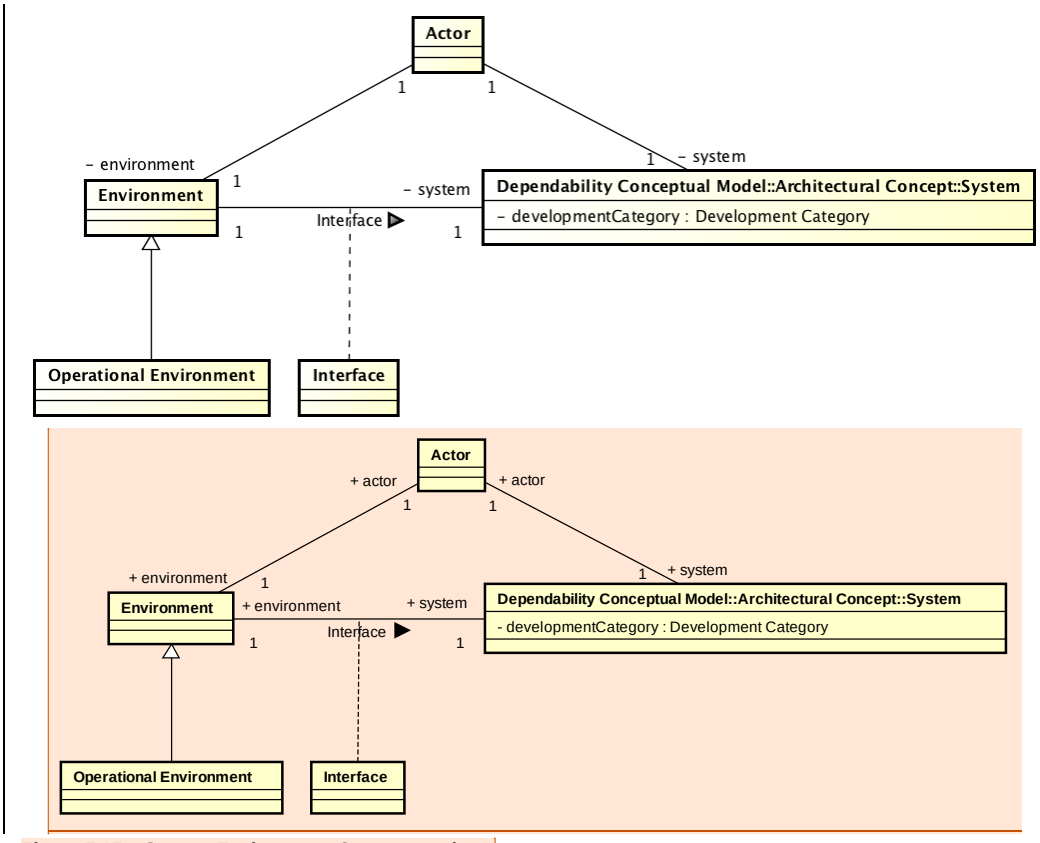


Figure 7-17 – System Environment Concept package

Formatted: Centered

Comment [A117]: DAF#9

7.5.1 Actor

Description

The Actor may be a stakeholder or a user of the system. ~~The System is operated by the Actor.~~

Comment [A118]: DAF#20

Generalizations

No additional generalizations

Attributes

No additional attributes

Associations

environment: Environment[1]~~: Actor [1..*]~~

Specifies that an Environment influences an Actor. ~~a System is operated by one or more Actors.~~

Comment [A119]: DAF#20

system:System [1]

Specifies that an Actor interacts with a System.

Comment [A120]: DAF#20

7.5.2 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.

7.5.3 Operational Environment

Description

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

Generalizations

Environment specified in 7.5.2.

Attributes

No additional attributes

Associations

No additional associations

7.5.4 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)

Comment [A121]: DAF#14

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~~Recently,~~ the USA FDA (Food and Drug administration) requires safety cases for introducing infusion pump.

Comment [A122]: DAF#15

The structure of DAC templates is defined in Dependability Assurance Argument Structure of DCM in Clause 7.2.1 through 7.2.26. DAC templates are represented by instance diagrams of SACM 1.0 classes.

Dependability Claim in Clause 7.2.1.1 for the target safety-sensitive consumer device is about the dependability requirements. In Clause 7.1, 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).~~ In Clause 8.3.~~

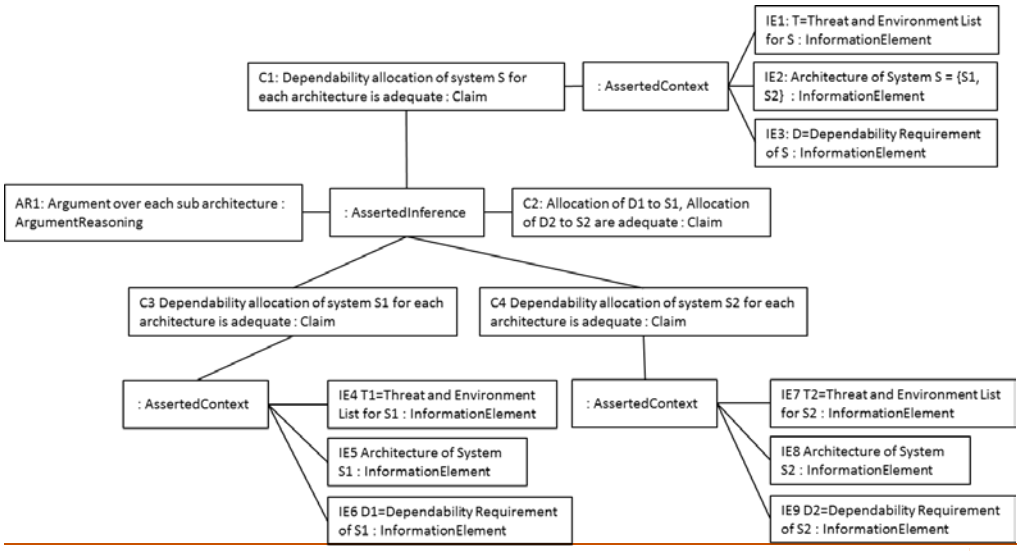
Comment [A123]: DAF#15

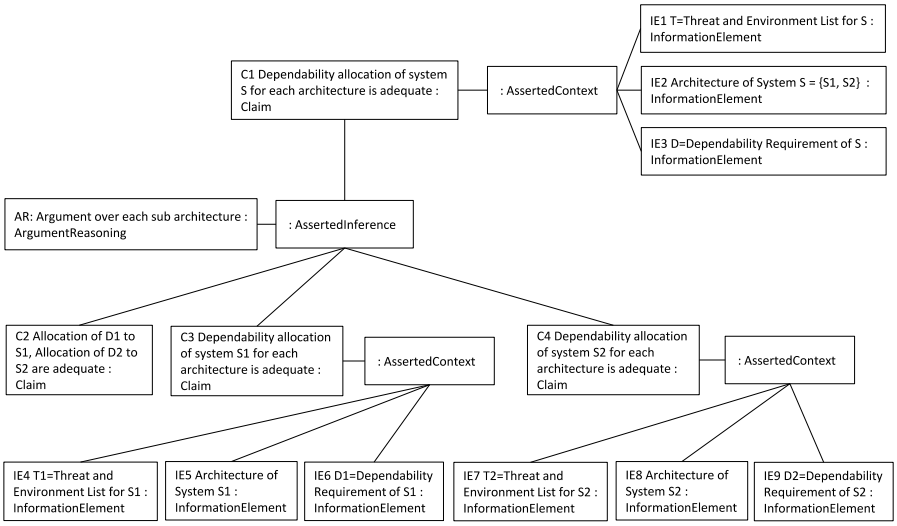
8.2 Representation of DAC Template by SACM Instance Diagram

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


C1: Dependability allocation of system S for each architecture is adequate : Claim
: AssertedContext
IE1: T=Threat and Environment List for S : InformationElement
IE2: Architecture of System S = {S1, S2} : InformationElement
IE3: D=Dependability Requirement of S : InformationElement
AR1: 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
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
: AssertedContext
IE7 T2=Threat and Environment List for S2 : InformationElement
IE8 Architecture of System S2 : InformationElement
IE9 D2=Dependability Requirement of S2 : InformationElement


C1 Dependability allocation of system S for each architecture is adequate : Claim
: AssertedContext
IE1 T=Threat and Environment List for S : InformationElement
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

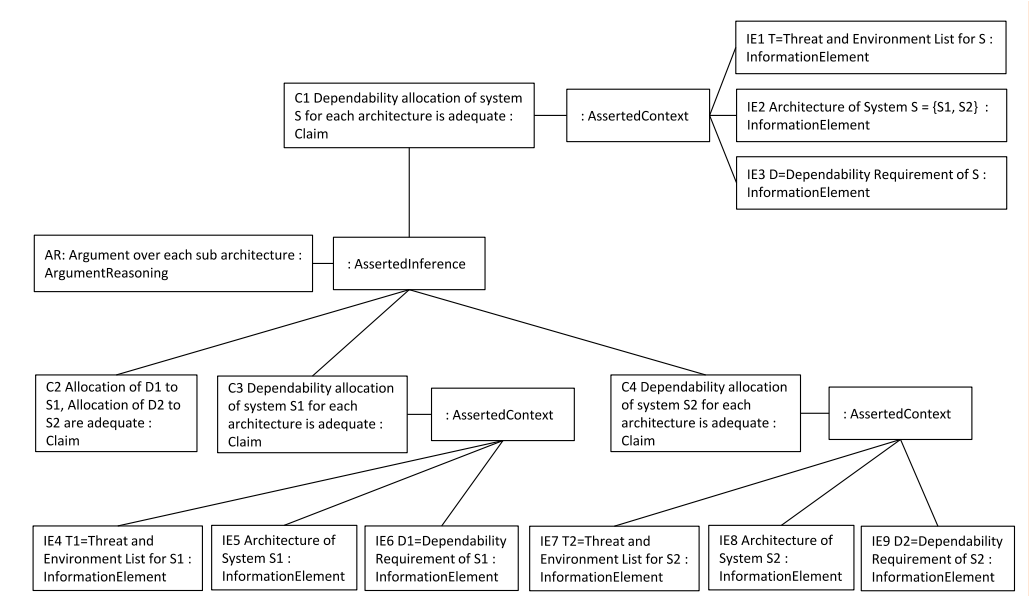


Figure 8-1 – DAC template for Dependability Allocation Argument

Comment [A124]: DAF#9

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 Clause 7.1). 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 ~~as h1,…, hn~~, 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 ~~for S: T=h1,…,hn~~ 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).

~~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 Clause 7.1). For example, if a system S consists of sub systems S1 and S2, and the threat and environmental list for S is derived as h1,…, hn, 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 for S: T=h1,…,hn 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).~~

Comment [A125]: DAF#16

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~~Figure 8-2~~. The DAC template is divided into 4 sub parts: "Top Structure", "Modification Argument", "Proven In Use Argument", and "Integration Argument" parts.

Formatted: Font: (Default) Times New Roman, 10 pt

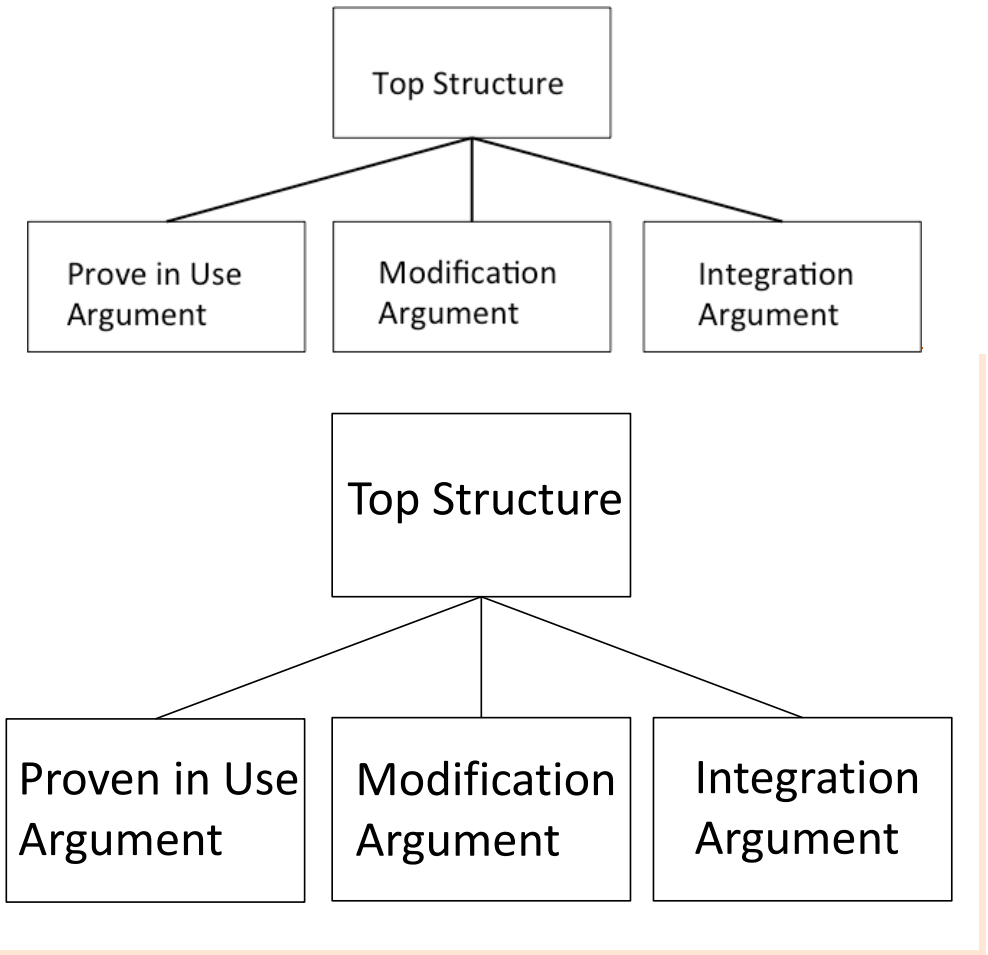


Figure 8-2 – Overview of Evolutionary Development Argument Template (Informative)

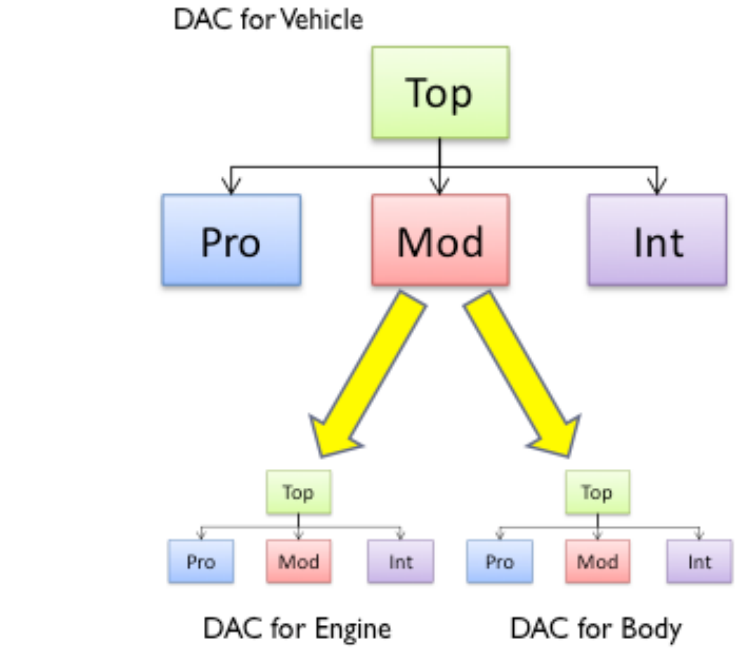
Comment [A126]: DAF#13

Comment [A127]: DAF#9

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 7.1). 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~~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.

Formatted: Font: (Default) Times New Roman, 10 pt



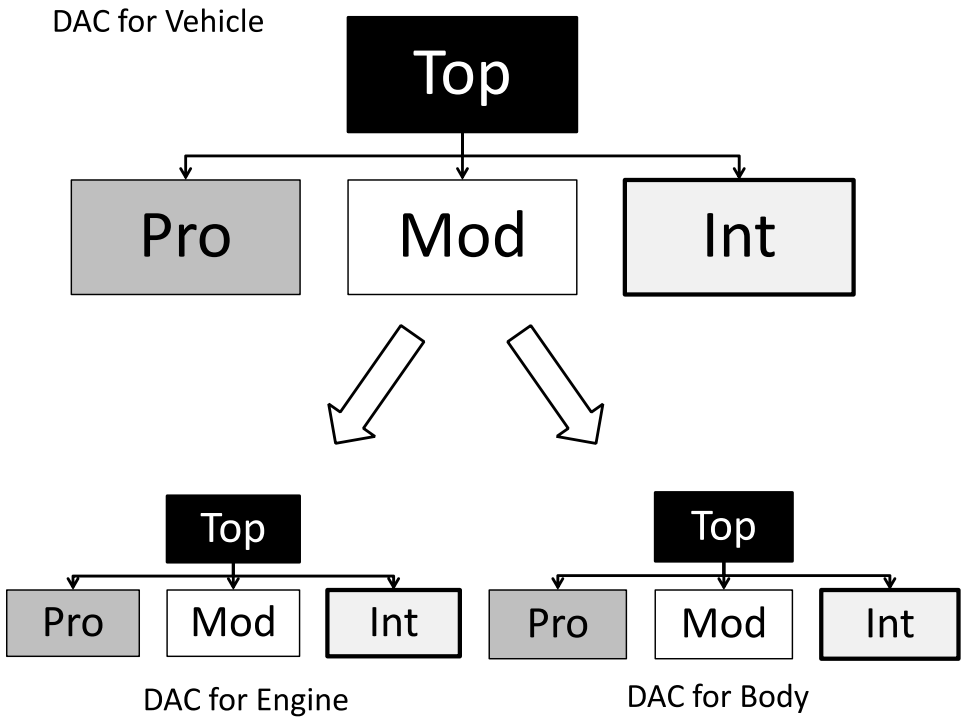


Figure 8-3 – An example of the use of DAC template for automobile (Informative)

Comment [A128]: DAF#13

Comment [A129]: DAF#9

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 Clause 7.2.2.2.

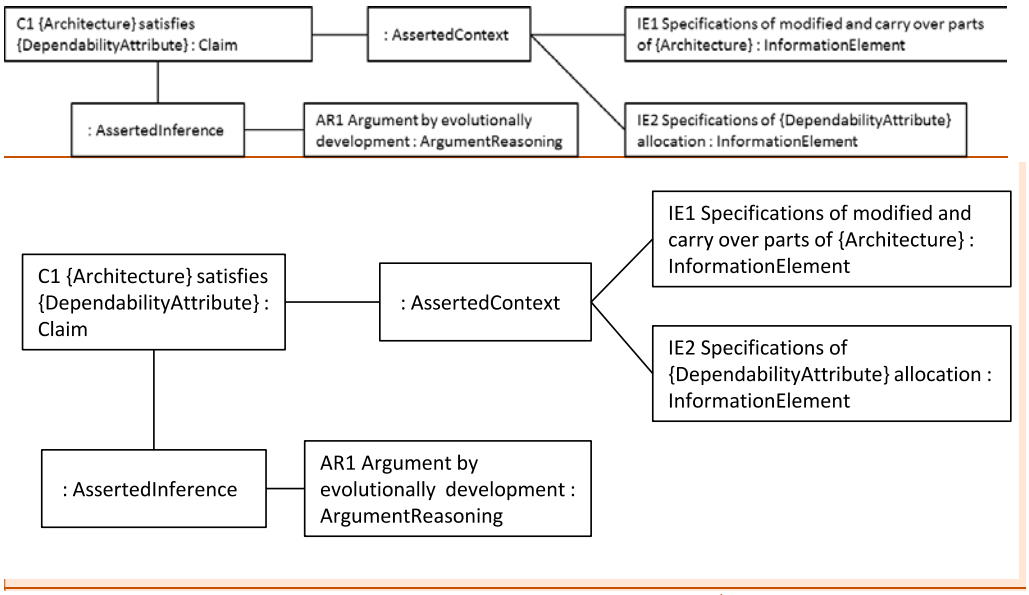


Figure 8-4 – Top Structure of the DAC for Evolutionary Development Argument

Comment [A130]: DAF#9

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

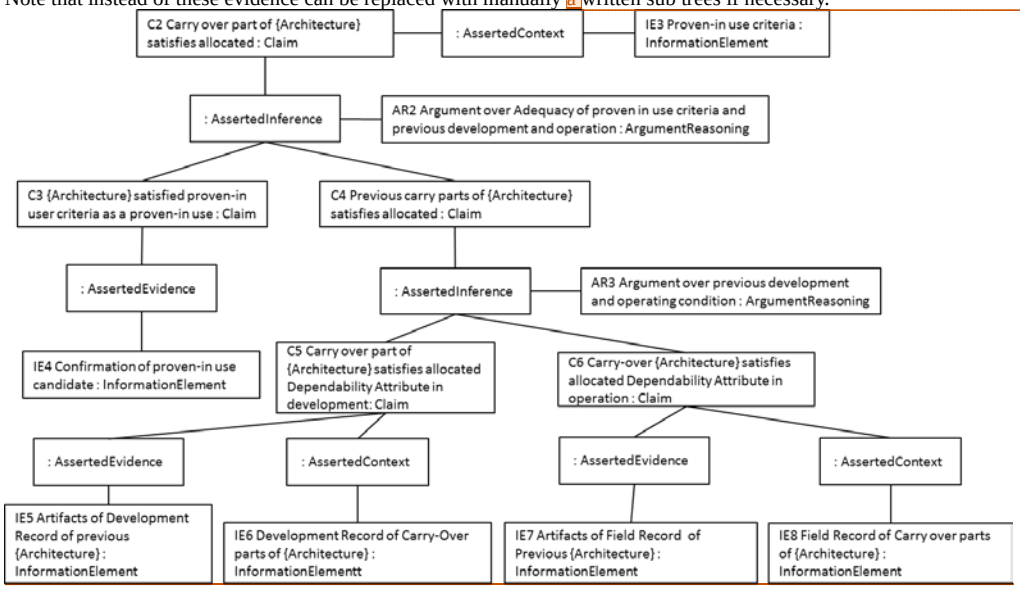
8.4.3 Proven In Use Argument

In DCM, Proven In Use Argument is defined in Clause 7.2.6. The Proven In Use Argument corresponds to the Proven In Use package (Figure 7-14~~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.

Formatted: Font: (Default) Times New Roman, 10 pt

The SACM instance diagram for Proven In Use Argument DAC template is shown in Figure 8-5~~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~~for~~ the dependability of the architecture.

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



Formatted: Font: (Default) Times New Roman, 10 pt

Comment [A131]: DAF#18

Comment [A132]: DAF#18

Comment [A133]: DAF#18

Comment [A134]: DAF#18

Comment [A135]: DAF#18

Comment [A136]: DAF#18

Comment [A137]: DAF#18

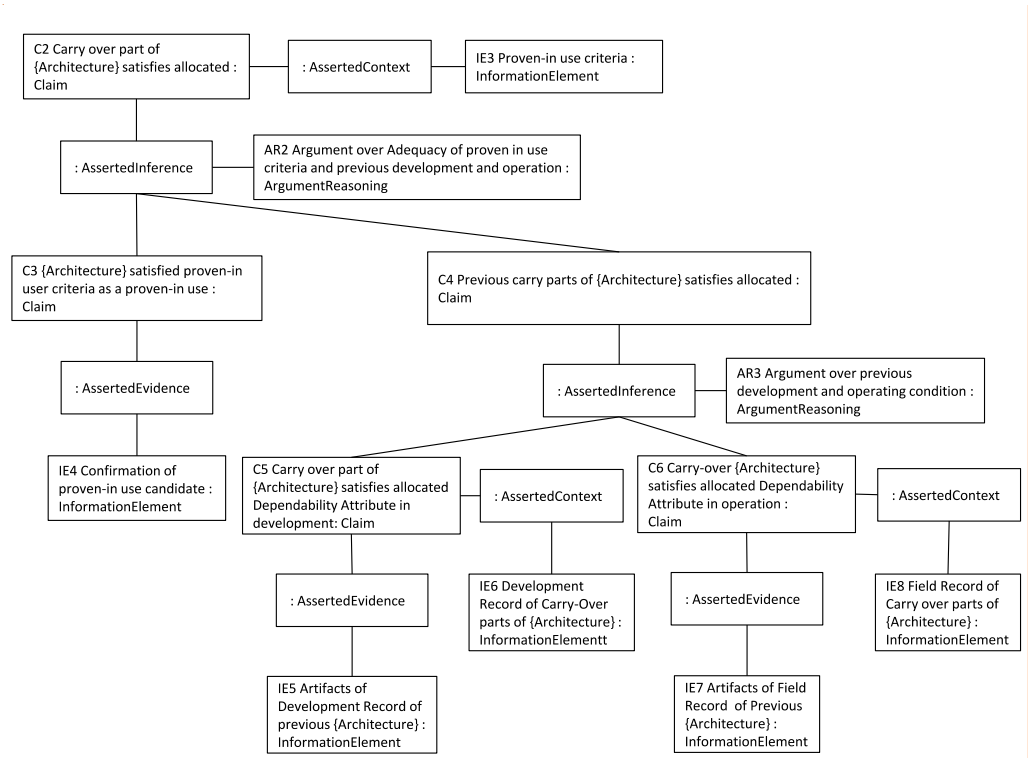


Figure 8-5 – Proven In Use Argument part of the Evolutionary Development Argument Structure

Comment [A138]: DAF#9

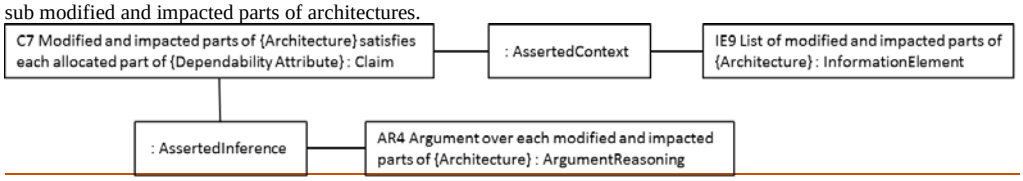
The XMI file for Proven In Use Argument part of the Evolutionary Development Argument Structure is ProvenInUseArgument.xmi (normative).

8.4.4 Modification Argument

Figure 8-6~~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. Argument Reasoning 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.

Formatted: Font: (Default) Times New Roman, 10 pt

Comment [A139]: DAF#18



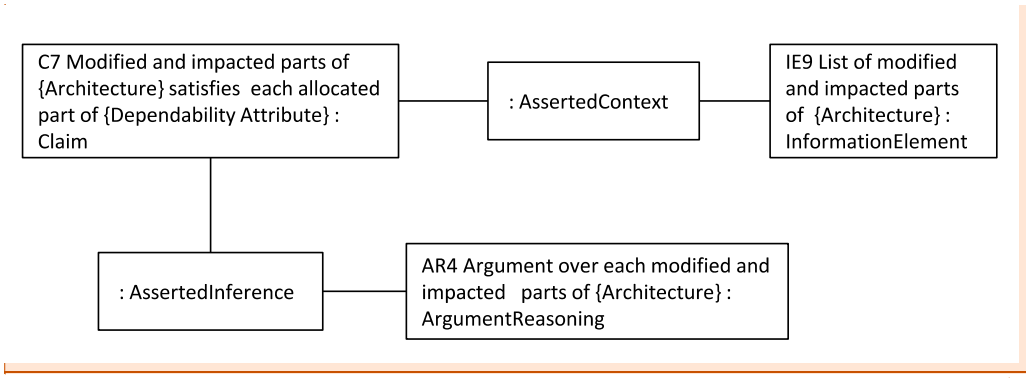


Figure 8-6 – Modification Argument Part of the DAC template for Evolutionary Development Argument Structure

Comment [A140]: DAF#9

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~~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.

Comment [A141]: DAF#18

Comment [A142]: DAF#18

Comment [A143]: DAF#18

Formatted: Font: (Default) Times New Roman, 10 pt

Comment [A144]: DAF#18

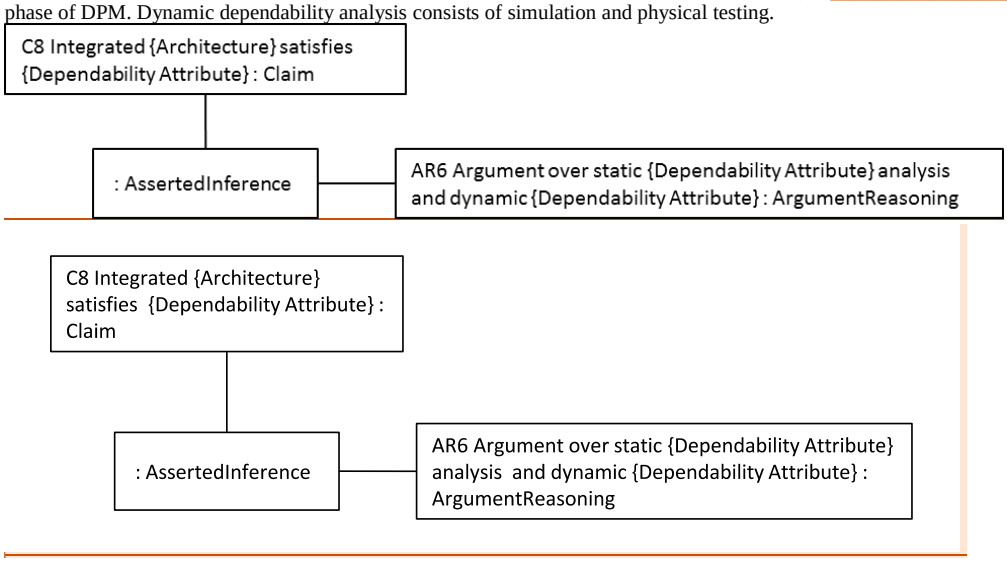


Figure 8-7 – Top Structure of Integration Argument part of the DAC template for Evolutionary Development Argument Structure

Comment [A145]: DAF#9

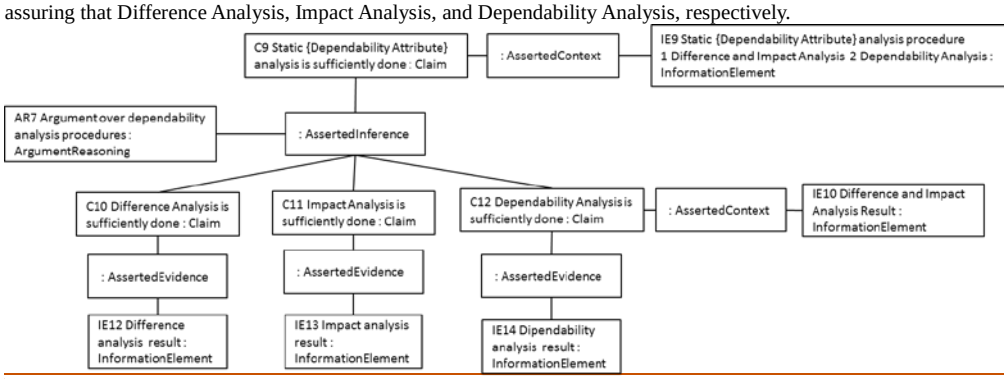
The XMI file for the top structure of Integration Argument part is IntegrationArgument.xmi (normative).

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~~Figure 8-8~~. InformationElement IE9 specifies Static Dependability Attribute analysis procedures

Formatted: Font: (Default) Times New Roman, 10 pt

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.



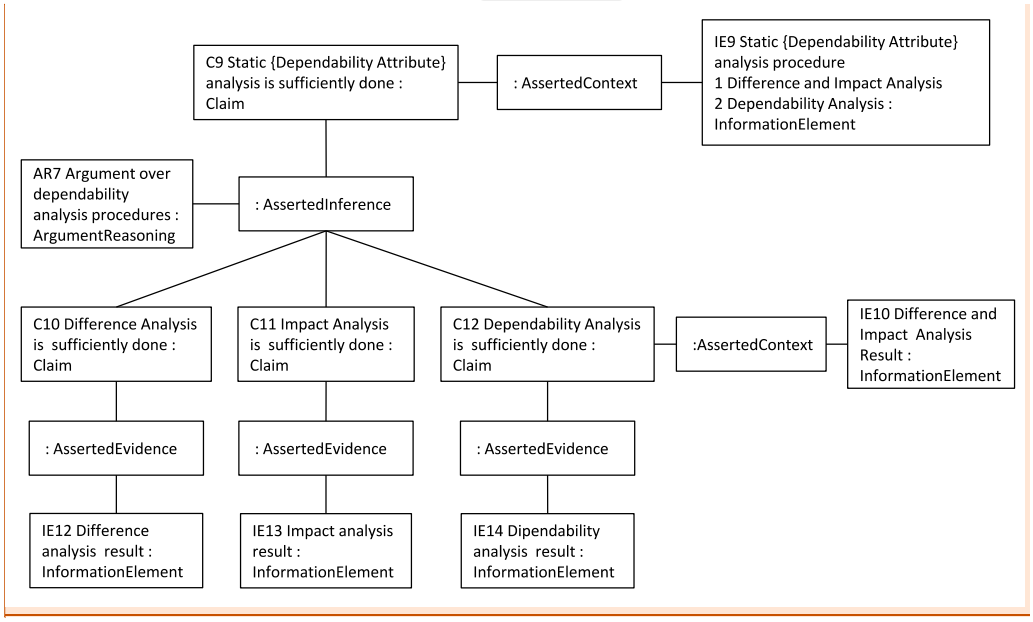


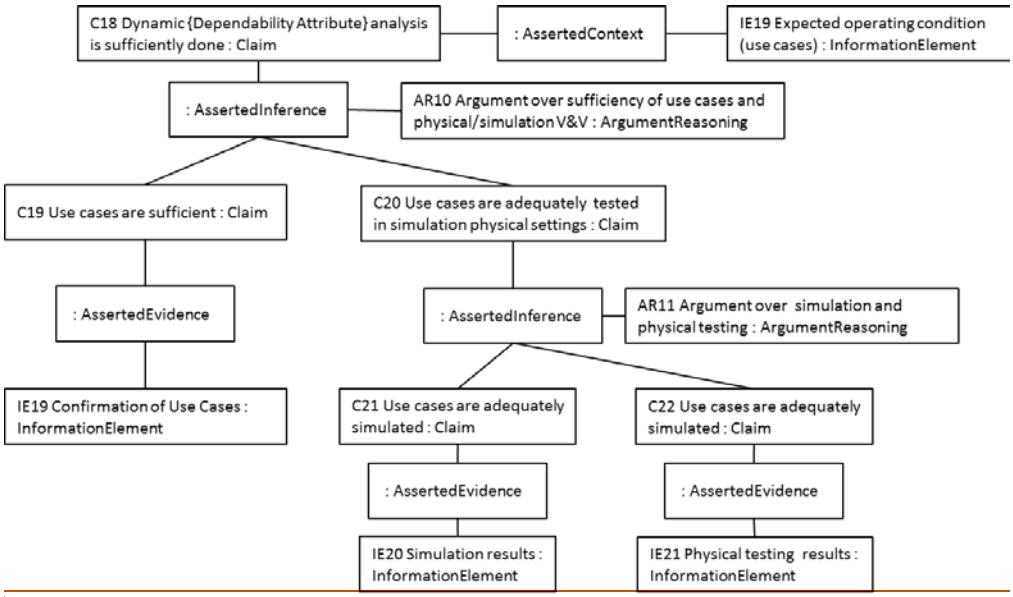
Figure 8-8 – Static Dependability Analysis Argument part of the DAC template for Evolutionary Development Argument Structure

Comment [A146]: DAF#9

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.



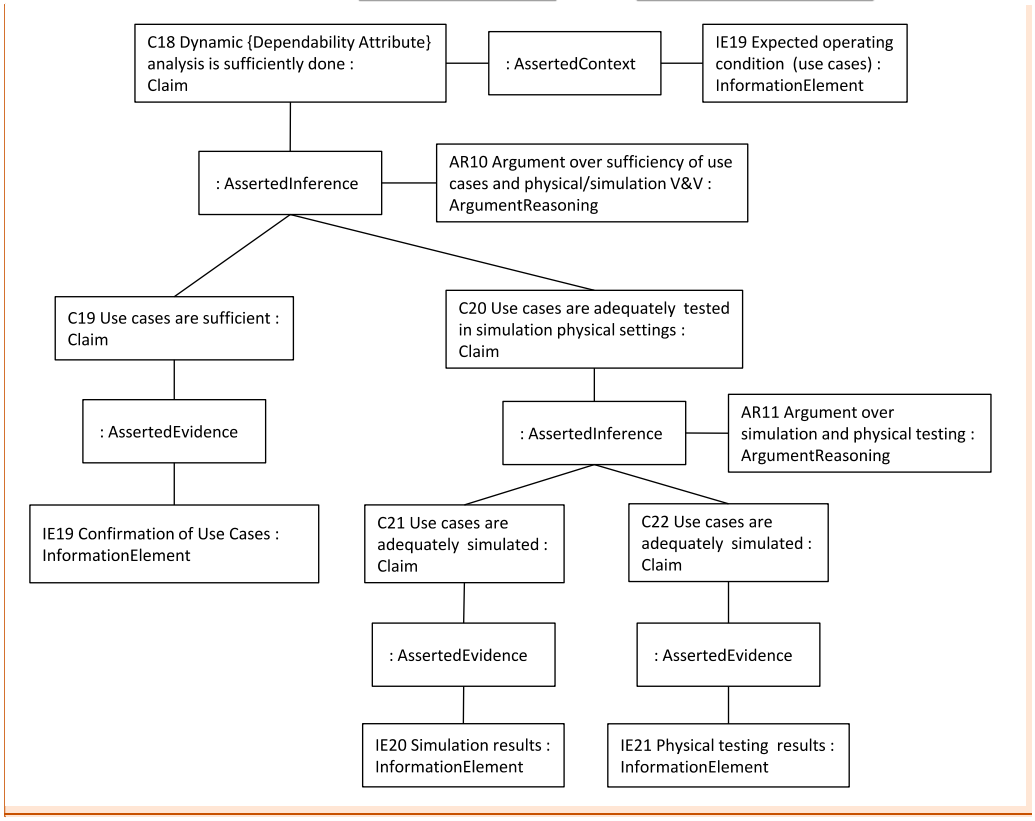


Figure 8-9 – Dynamic Dependability Analysis Argument part of the DAC template for Evolutionary Development Argument Structure

Comment [A147]: DAF#9

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~~much~~ use case as possible to validate the dependability requirements (Claims C21 and C22).

So, the argumentation structure in Figure 8-9~~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~~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.

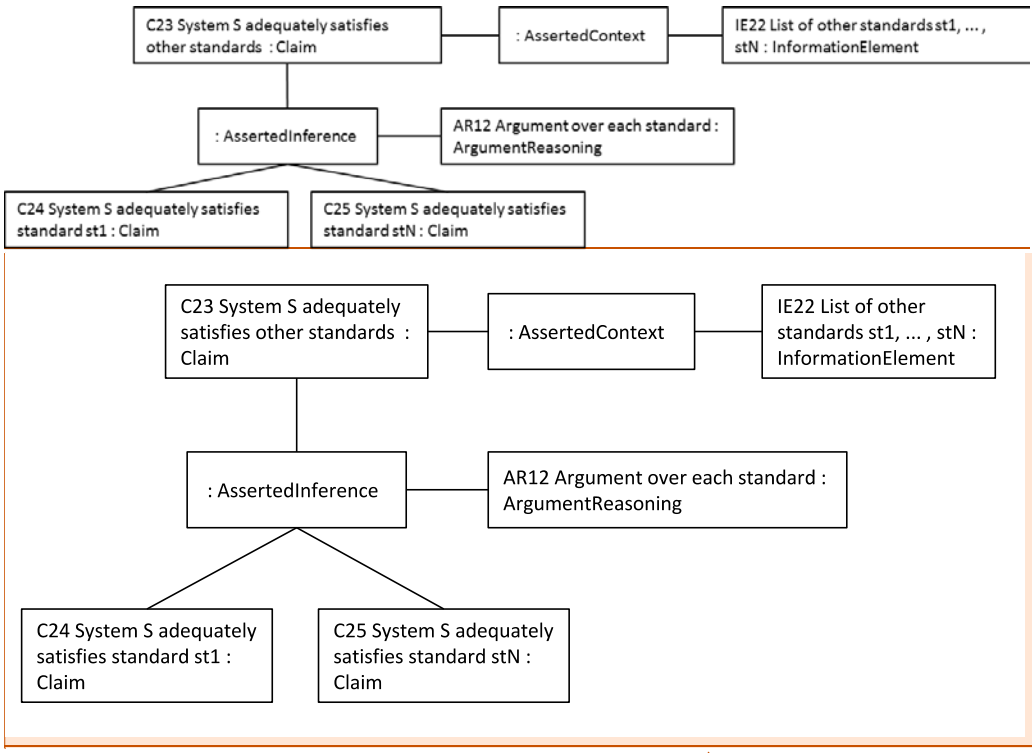


Figure 8-10 – The DAC Template for Standard Compliance Argument Structure

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

Comment [A148]: DAF#18

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: Font: (Default) Times New Roman, 10 pt

Comment [A149]: DAF#18

Comment [A150]: DAF#9

9 Dependability Process Model

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.

9.1 Overview of Iterative and Rapid Process~~Iterative and Rapid Process~~

Comment [A151]: DAF#17

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~~is~~ required to be defined. However, it is not realistic to fully lay out all the~~each single~~ V models all at once because of the size of processes for SSCD development.

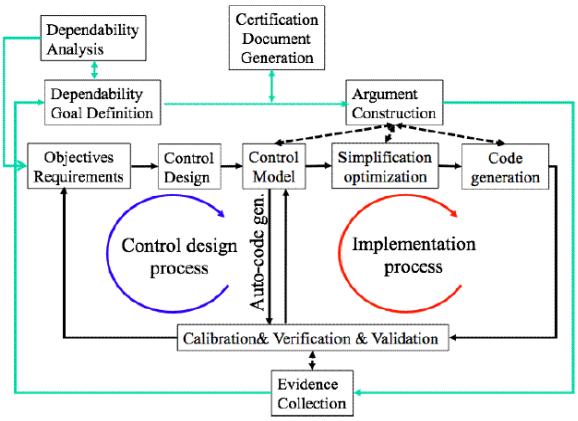
Comment [A152]: DAF#18

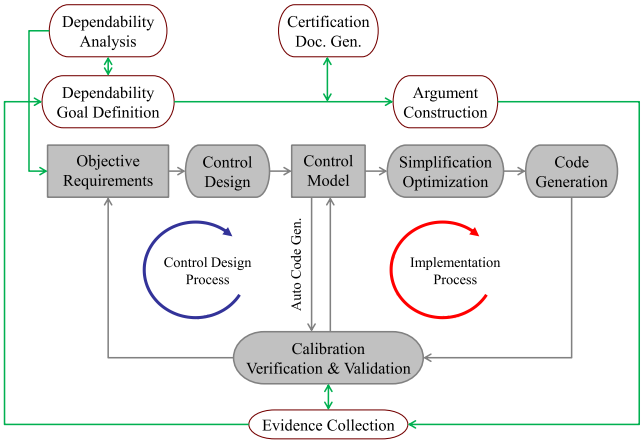
Comment [A153]: DAF#18

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 the following Figure 9-1 – Example of Rapid Iterative Process~~Figure 9-1 – Example of Rapid Iterative Process~~.

Two circles are supplementary illustrated in Figure 9-1~~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.


Dependability Analysis
Certification Document Generation
Dependability Goal Definition
Argument Construction
Objectives Requirements
Control Design
Control Model
Simplification optimization
Code generation
Control design process
Auto-code gen.
Implementation process
Calibration& Verification & Validation
Evidence Collection



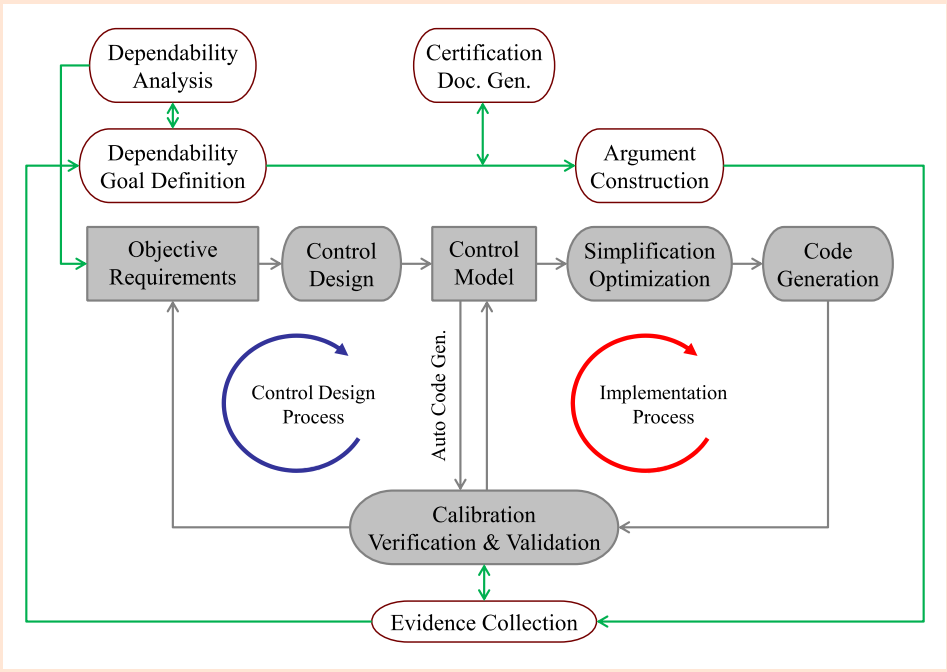


Figure 9-1 – Example of Rapid Iterative Process

Comment [A154]: DAF#9

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.

9.2 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

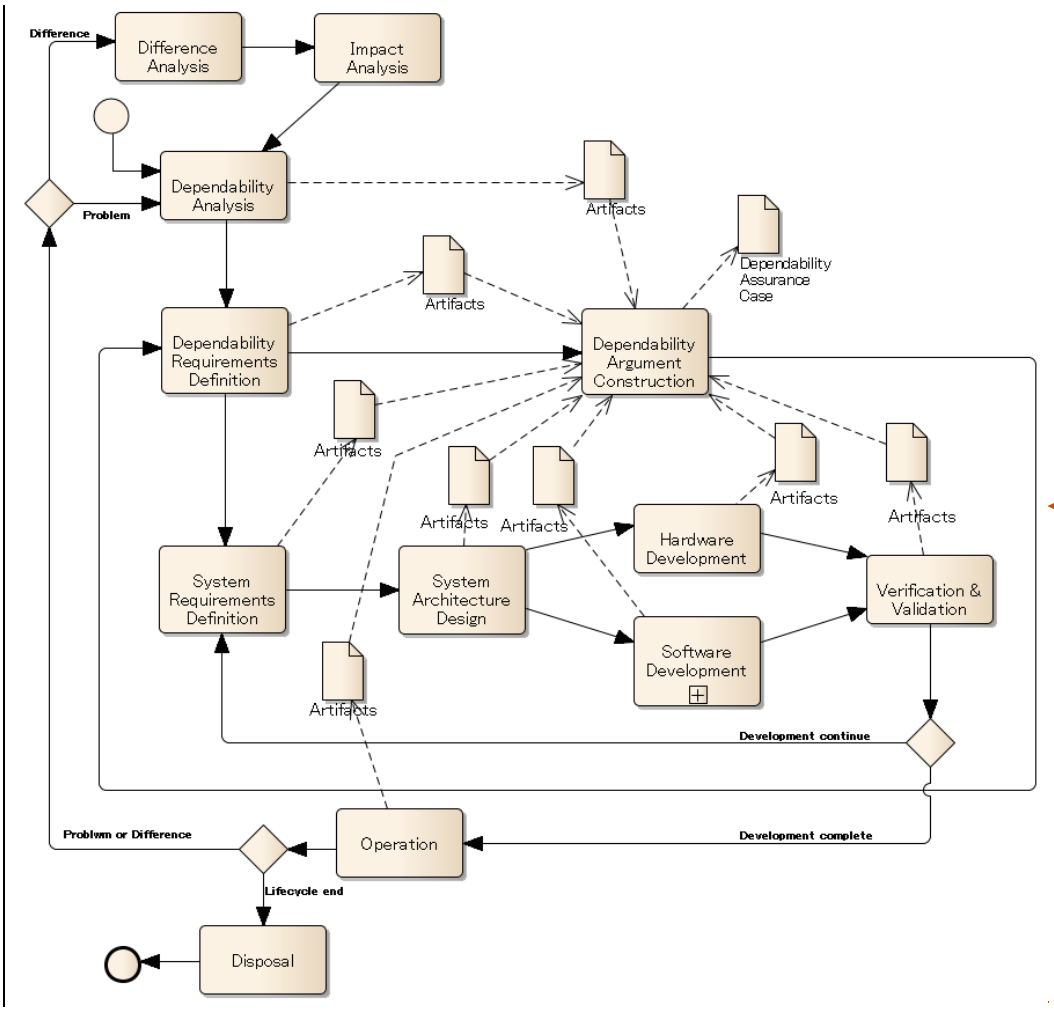
These activities constitute the Dependability Process. The entire relationship of the related processes and activities are shown in Figure 9-2 – BPMN for Dependability Process Model~~Figure 9-2 – BPMN for Dependability Process Model~~. As shown in Figure 9-2~~Figure 9-2~~, the Systems Engineering Processes in the middle of the Figure 9-2~~Figure 9-2~~ are performed along with the Dependability Requirements Definition activity and the Dependability Argument Construction activity. These activities defined here are~~is~~ a minimum set, and the set can be applied with necessary extensions for various consumer device developments.

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: Font: (Default) Times New Roman, 10 pt

Comment [A155]: DAF#18



Formatted: Centered, Don't keep with next

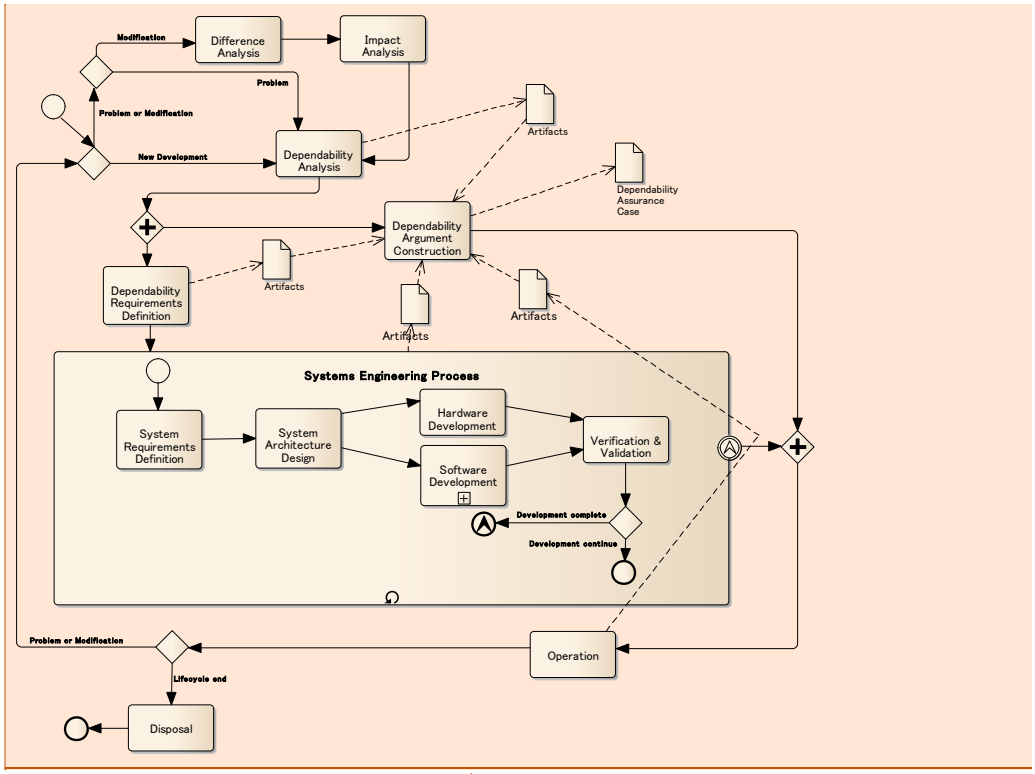


Figure 9-2 – BPMN for Dependability Process Model

Comment [A156]: DAF#5, DAF#9, DAF#

9.2.1 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, and
- Development results from the Evolutionary Development Process consisting of the Difference Analysis activity and the Impact Analysis activity.

~~The inputs of this activity are~~
~~- Product Plan provided by upper level processes including Planning,~~
~~- Incident Reports from the Problem flow, and~~
~~- 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),

Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Bulleted + Leve 1 + Aligned at: 0" + Indent at: 0.29"
Formatted: List Paragraph
Comment [A157]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"
Formatted: Font: (Default) Times New Roman, 10 pt

~~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.

9.2.2 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~~is~~

- 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.

9.2.3 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, and
- 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.

9.3 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~~Figure 9-2~~). Its outcome artifacts are used as Evidence to assure dependability of the system in the Dependability Argument Construction. Figure 9-3~~Figure 9-3~~ illustrates the Systems Engineering Process diagram extracted from Figure 9-2~~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)

Comment [A158]: DAF#22

Formatted: Font: (Default) Times New Roman, 10 pt

Comment [A159]: DAF#22

Comment [A160]: DAF#18

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"

- –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.

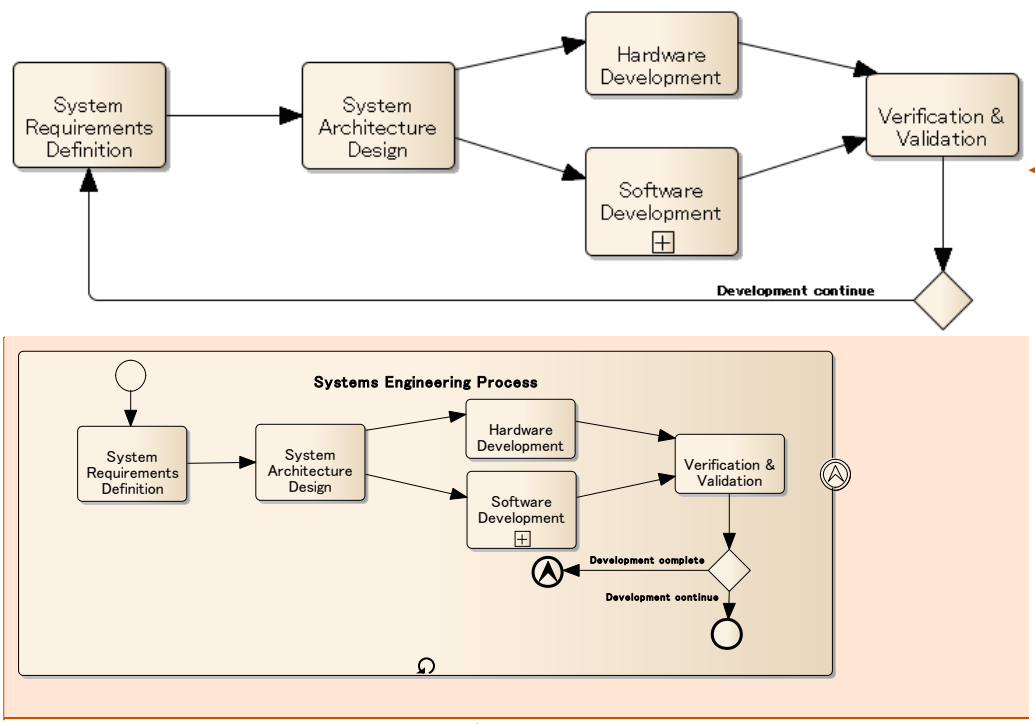


Figure 9-3 – BPMN for Systems Engineering Process

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

9.3.1 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;
~~The inputs are~~
- –Needs from upper layer processes,

Formatted: Don't keep with next

Comment [A161]: DAF#5, DAF#9

Formatted: Font: 10 pt, Not Bold

Comment [A162]: DAF#30

Formatted: Font: 10 pt, Not Bold

Comment [A163]: DAF#22

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"

- –Development Plan, and
- –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;
~~The outputs are~~
- –Requirement Specifications, and
- –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.

9.3.2 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.
~~When the existing System Requirements Definition has been found as insufficient in the course of the System Architecture Design, it is possible to rework the System Requirements Definition as an exceptional treatment.~~

As examples of inputs, they are the following documents;
~~The inputs are~~
- –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 output is~~
- –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.

9.3.3 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. ~~When the existing System Architecture Design has been found as insufficient in the course of Hardware Development, it is possible to rework the System Architecture Design as an exceptional treatment.~~

As examples of inputs, they are the following documents;
~~The input is~~
- –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;~~The outputs are~~
- –Hardware Specification,
- –Hardware Prototype, and
- –Test Results.

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

Comment [A164]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"
Comment [A165]: DAF#19
Comment [A166]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"
Comment [A167]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"
Comment [A168]: DAF#19
Comment [A169]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"
Comment [A170]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"

9.3.4 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 the 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 the Figure 9-4 – Software Development Process~~Figure 9-4 – The Correct Control Software models~~.

Formatted: Font: (Default) Times New Roman, 10 pt

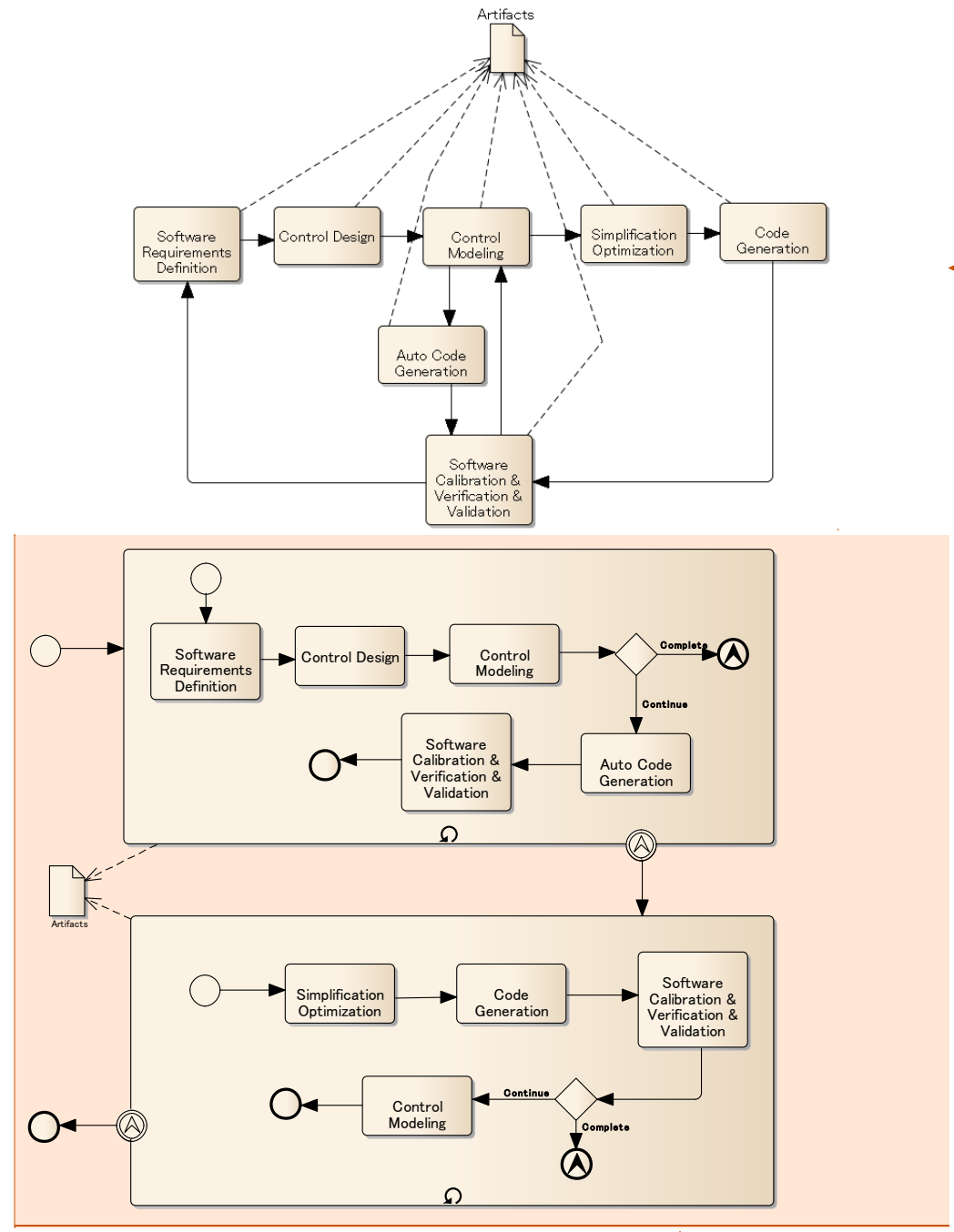


Figure 9-4 – Software Development Process~~The Correct Control Software models~~

Formatted: Normal, Centered, Indent: Left 0", Don't keep with next

Comment [A171]: DAF#5, DAF#9, DAF#

9.3.4.1 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;~~The inputs are~~
- System Requirements, and
- 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;~~The outputs are~~
- Software Requirements.

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

9.3.4.2 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;~~The inputs are~~
- 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;~~The outputs are~~
- Control Specifications.

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

9.3.4.3 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;~~The inputs are~~
- 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;~~The outputs are~~
- Control Software Models.

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

9.3.4.4 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;~~The Inputs are~~
- 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;~~The outputs are~~
- Implementation Codes that is~~are~~ generated automatically.

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

Comment [A172]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"
Comment [A173]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"
Comment [A174]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Comment [A175]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"
Comment [A176]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"
Formatted: Font: (Default) Times New Ron
Comment [A177]: DAF#22
Comment [A178]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"
Comment [A179]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted
Formatted: Font: (Default) Times New Ron
Comment [A180]: DAF#22
Comment [A181]: DAF#22
Formatted
Formatted
Comment [A182]: DAF#22
Comment [A183]: DAF#18
Formatted
Formatted
Formatted
Formatted
Formatted: Font: (Default) Times New Ron
Comment [A184]: DAF#22

9.3.4.5 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. ~~Therefore investigation to achieve equivalent quality control by simplified and/or optimized approach considering constraints of implementation size, cost and so on.~~ 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; ~~The inputs are~~

- ~~s~~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; ~~The outputs are~~

- ~~s~~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.

9.3.4.6 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 ~~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 Codes contrary to Auto-Coding results which could not be flashed into the target CPU because of the size in CPU resources and poor 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; ~~The inputs are~~

- ~~s~~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; ~~The outputs are~~

- ~~s~~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.

9.3.4.7 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.

~~Various frameworks to standardize and generalize software and to use them in various products universally are developed in order to reduce control software development cost and to improve efficiency of the process. Calibration is a process to tune parameters implemented in software according to the specific product requirements and target market environment. It is based on the possibility of using the same software for various products with varying parameters.~~ 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; ~~The inputs are~~

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

Comment [A185]: DAF#18

Comment [A186]: DAF#34

Comment [A187]: DAF#22

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"

Comment [A188]: DAF#22

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: Font: (Default) Times New Roman, 9 pt

Comment [A189]: DAF#22

Formatted: Font: (Default) Times New Ron

Comment [A190]: DAF#33

Comment [A191]: DAF#22

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"

Formatted: Font: (Default) Times New Roman, 10 pt

Comment [A192]: DAF#22

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"

Comment [A193]: DAF#22

Formatted: Font: (Default) Times New Roman, 10 pt

Comment [A194]: DAF#34

Comment [A195]: DAF#22

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 outputs are~~

~~- Validated parameters,~~

- Validated parameters,
- Verification results, and
- Final Implementation Code.

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

9.3.4.8 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.

9.3.5 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. ~~When the Hardware Development results and/or the Software Development results have been found as insufficient in the course of Verification and Validation, it is possible to rework the Hardware Development and/or the Software Development as an exceptional treatment. Major rework may be treated as an issue that should be treated in the next development loop.~~

As examples of inputs, they are the following documents;~~The inputs are~~

- System Specification (System Architecture),
- Hardware products, and
- Software code~~s~~.

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 outputs are~~

- Verification and Validation results for each level, and
- 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.

9.4 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.

Comment [A196]: DAF#22

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: Indent: Left: -0.29", Bulleted + Level: 1 + Aligned at: 0" + Indent at: 0.29

Formatted: Font: (Default) Times New Roman, 10 pt

Comment [A197]: DAF#22

Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: Font: (Default) Times New Ron

Comment [A198]: DAF#22

Comment [A199]: DAF#19

Comment [A200]: DAF#22

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"

Comment [A201]: DAF#22

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"

9.4.1 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:~~The inputs are~~

- 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:~~The outputs are~~

- 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.

9.4.2 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. ~~When the Difference Analysis result has been found as insufficient in the course of the Impact Analysis, it is possible to rework the Difference Analysis as an exceptional treatment.~~

As examples of inputs, they are the following documents:~~The inputs are~~

- 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:~~The outputs are~~

- 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.

9.5 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~~Figure 9-2~~).

9.5.1 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:~~The inputs are~~

- Verification and Validation Results, and
- 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:~~The outputs are~~

- Incident Reports for found problems.

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

Comment [A202]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"
Comment [A203]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"
Comment [A204]: DAF#19
Comment [A205]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"
Comment [A206]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"
Formatted: Font: (Default) Times New Roman, 10 pt
Comment [A207]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"
Comment [A208]: DAF#22
Formatted: Font: (Default) Times New Roman, 10 pt
Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"

9.5.2 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 input is~~

- 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;~~The outputs are~~

- Reusable resources if they exist.

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

Comment [A209]: DAF#22

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"

Comment [A210]: DAF#22

Formatted: Font: (Default) Times New Roman, 10 pt

Formatted: List Paragraph, Indent: Left: 0 Hanging: 0.1", Bulleted + Level: 1 + Aligne at: 0" + Indent at: 0.29"

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

Annex B
Acknowledgements
(Informative)

The following companies, organizations and people have contributed significantly to this specification either directly or indirectly through discussions and feedback:

- Consumer Device Safety Standardization Working Group, Information-technology Promotion Agency,Japan (IPA,Japan)
  - Seiichi Shin, Chair, The University of Electro-Communications, SICE
  - Masamichi Nakagawa, Panasonic Corporation
  - Makoto Sekiya, HONDA
  - Hiroo Kanamaru, Mitsubishi Electric
  - Nobuyasu Kanekawa, Hitachi, Ltd.
  - Yoshihito Sakamoto, IBM Japan
  - Susumu Akiyama, DENSO
  - Seigo Kotani, TCG
  - Daisuke Soma, CAV Technologies, Ltd.
  - Seiko Shirasaka, Keio University
  - Kenji Hiranabe, Change Vision, Inc.
  - Hitoshi Arima, dSPACE Japan
- Hiroshi Miyazaki, Fujitsu Limited
- Kenji Taguchi, AIST
- Geoffrey Biggs, AIST
- Tetsuo Kotoku, AIST
- Yoshihiro Nakabo, AIST
- Isashi Uchida, IPA,Japan
- Hiroyuki Haruyama, IPA,Japan
- Akira Ohata, TOYOTA Motor Corporation
- Naoya Ishizaki, TOYOTA Motor Corporation
- Satoru Watanabe, TOYOTA Motor Corporation
- Yutaka Matsuno, The University of Electro-Communications
- Djenana Campara, Nicholai Mansourov, KDM Analytics
- Safe Automotive soFtware architEcture Consortium (SAFE)

Formatted: Indent: Hanging: 0.39"

Annex C
Experiment of DAF
(Informative)

This annex is a brief report of an experiment for the applicability of the OMG DAF Standard. The experiment was conducted, assuming to develop a ABS(Antilock Braking System) as a system, to see how the DAF Standard can enhance existing dependability argumentations as well as how the construction of argumentation can be made effective.

In this scenario of experiment, the ABS is assumed to be modified according to new requirements based on a carry-over ABS system which is already in markets. The new ABS was designed to add a function which enables a vehicle to turn at a corner more safely. At the same time, the dependability argumentation of the new ABS was also constructed on the course of the development according to the DPM in the DAF Standard.

In this sample, each attribute of Dependability was not discussed because of simplicity. Instead, the functional safety was only discussed as an attribute of Dependability. The following is the procedure to apply the DAF Standard for the new ABS development.

First, SysML models of the ABS system are created according to the Systems Engineering process in the DPM, during which the enhancement of the ABS system is discussed based on the DAC to construct the Dependability argumentation of the new ABS system. Then, both of the SysML models and the DAC are evaluated by experts in the functional safety certification in terms of the validity of the Dependability argumentation. Finally, the assessment of applicability of the DAF Standard is carried out to describe how the DAF is confirmed effective.

1. DCM

The DCM was utilized to define all the things to construct the argumentation. Each class in the DCM was tailored to one suitable for the functional safety, instead of the Dependability in general. The DCM was found effective when defining classes for the functional safety with a sense of mutual exclusiveness and collective exhaustiveness.

2. DAC

The DAC template was utilized to construct the argumentation of the functional safety of the new ABS system. Based on the template, carry-over parts of the ABS system were firstly discussed to see if the notion of Proven-In-Use can be applicable for argumentation of the functional safety. Then, newly designed parts of the ABS system were validated and argued over to specify its validity for functional safety. Finally, the integration of the carry-over parts and the newly designed parts as a whole was discussed in the course of comprehensive evaluations.

3. DPM

The DPM was referenced to develop the new ABS system as well as to construct the dependability argumentation. The process was successfully applied to develop both SysML models and argumentation at the same time.

Assessment of Applicability for DAF Standard

The applicability of DAF Standard was qualitatively assessed in terms of the following two aspects to clarify the benefits of DAF Standard.

■ Robustness of Dependability argumentation (Functional Safety in this case)

Constructing the argumentation of the functional safety for the new ABS system with DAF became much easier than without DAF because the following three reasons;

① DCM helped define aspects to consider to construct the argumentation.
② DAC enabled robustly argue over the development of new products by dividing the discussion in argumentation into three pillars, Proven-In-Use, Modification and Integration.
③ DPM guided us to easily follow the process for argumentation as well as helped us understand that the DPM with rapid iterative process can enhance the conventional V process in order to develop more dependable products.

■ Efficiency in Argumentation construction

The efficiency in the argumentation construction became much higher than without DAF because of the following two reasons;

① The DAC template worked well to quickly construct the argumentation. Also, the argumentation based on the DAC template helped communicate with others for consistency in peer reviews of argumentation.
② The artifacts of the argumentation based on the DAC template were confirmed reusable for future reference or even a basis for new development.

Formatted: Bullet_1, Numbered + Level: 1 Numbering Style: ①, ②, ③ … + Start at: 1 Alignment: Left + Aligned at: 0" + Indent a 0.29", Tab stops: 0.35", Left

Comment [A211]: DAF#25

Formatted: Font: 10 pt, English (U.S.)