SDS R3.1:

System Description Standard:

Semantic Specification

Author: Philippe Spaas

Document History

Document Location

This is a snapshot of an on-line document. Paper copies are valid only on the day they are printed. Refer to the author if you are in any doubt about the currency of this document.

This document can be found in the following Community: System Description Standard - SDS.

Revision History

Date of this revision: 23/07/12	Date of next revision *(date)*

Revision Number	Revision Date	Author	Summary of Changes	Changes marked
R1	July 13, 1998	Philippe Spaas	First publication of ADS R1	N
R2	March 26, 2002	Philippe Spaas	Publication of ADS R2	N
R3	December 18, 2008	Philippe Spaas	Publication of SDS R3, ADS R3	N
R3.1	July 23rd, 2012	Philippe Spaas	Publication of SDS R3.1, ADS R3.1	N
	Aug 6th, 2012		Classified as '© IBM'	N

Contributors

Current Release

SDS R3 was revised to incorporate the feedback from the SA4TeamSD project. This project uses SDS R3 as the basis for the development of an integrated solution design environment built within Rational System Architect. The SA4TeamSD project team, led by Mark Palmer and Tom Seevers offered a unique opportunity to validate the SDS model and engage in a dialogue about useful extensions. The contribution of Ian Charters was a direct influence on the evolution of the modeling support for the Requirements Viewpoint (see [3]).

Since the last release, a number of suggestions were received from among others Jim Amsden and Guy Hillier which were taken into account as part of this revision. Helpful suggestions were received as part of the review process from Tom Seevers, Gisbert Kruesemann and Ian Charters. We would also like to explicitly thank David Brown for his thorough review of the proposed changes.

Past Releases

The initial discussions about the R3 revision of ADS were initiated during the GSM (Global Services Method) and RUP (Rational Unified Process) integration project undertaken within the method development program aiming to deliver IBM's Unified Method Framework (UMF).

As part of the integration project, a specific work stream on Foundational Semantics, sponsored by Ed Kahan, focused on the further evolution of ADS, and defined the overall objectives of the 3rd revision.

The core team of the Foundational Semantics work stream consisted of Dave Brown, Ian Charters, and Philippe Spaas as work stream lead. This core group benefited from the comments and insights offered by the wider team involved in the various UMF related work streams.

Many people invested time and energy to provide constructive criticisms and point out areas for improvement. This document benefitted from the review comments received from: Pete Cripps, Dave Brown, Bran Selic, Eric Nelson, Ian Charters, Paul Making, Mathew Kaplan, Jeremy Caine and others.

The 3rd revision leans heavily on the insights obtained from the two earlier revisions, and consequently still benefits from the earlier contributions from Luba Cherbakov, Ed Kahan, Deborah Le Monde, Ian Charters, Karin Duermeyer, Steve J Cook, Nick Ababurko, Pete Cripps, Bertus Eggen, Trevor Hopkins, Chris Kirby, Gisbert Kruesemann, Tim Lloyd, Marcel Schlatter, John Black, Paul Fertig, George Galambos, Ralf Geyer, Jim Whitmore and Peter Wittmann.

Contents

Table of Figures

1. System Description Standard: Semantic Specification

1.1 Preface

The System Description Standard R3 (SDS R3) project was initiated within the context of the creation of the Unified Method Framework (UMF) and more particularly to support the creation of the artifacts related to the Unified Application Development Framework (UADF).

Working within the scope of the UADF project, provided the opportunity to:

- Broaden the original remit of the Architecture Description Standard (ADS):
    - ADS was originally defined for an IT system context, and taking on board the concepts and vision of the Rational Unified Process – System Engineering (RUP-SE) [6], allowed us to reposition ADS as the architecture description standard for systems in general. Hence the names change from ADS to SDS.
    - ADS R2 focused on the functional and operational aspects of the architecture of an IT system, R3 extends this to include all of the basic viewpoints [5], i.e. it adds the requirements and validation aspects.
- Cater for additional engineering techniques, in particular those offered by RUP-SE, in conjunction with those already supported by ADS R2
- Extend the existing concepts by incorporating additional ones offered by RUP-SE and UML2

To reflect the change in scope of the description standard, we will use SDS R3 to refer to the System Description Standard, and use ADS R3 to refer to the IT System Description Standard.

The current revision ensures that the refinements identified during the SA4TeamSD project are incorporated into the description standard.

1.2 What is the purpose of the document?

The SDS Semantic Specification defines, together with the UML2 Superstructure Specification [2], all concepts and their basic relationships required to describe systems. And more specifically the SDS Semantic Specification aims to provide the clear and unambiguous definition of the concepts involved in modeling the requirements, architecture and validation of systems and extends, where necessary, the concepts defined in UML2.

The semantic specification is based on a metamodel and on supporting descriptions, which explain what each of the metamodel concepts means and how they are related. For insight in how the concepts are actually used, we refer to the artifacts and guidance papers of the UADF, and more particularly to those associated with the Architecture Domain for which SDS provides the supporting language.

Note:

- This is **not** an introductory document - it assumes familiarity with the SDS Overview ([9]) and Glossary ([10]), and general systems modeling experience.
- Any tooling initiative, based on the SDS metamodel, providing support to practitioners will further extend the model in order to support the activities undertaken by practitioners, by e.g. incorporating additional attributes, documentation, versioning and other features.

1.3 Who should read this document?

The primary intended audiences for this document are method authors, tool builders and practitioners in general who are interested in the precise semantic definitions of the concepts underlying the artifacts and guidance of the architecture domain of the UADF and the solution design tooling implemented in Rational System Architect (SA4TeamSD).

We encourage method authors and others to comment on these documents and augment their content.

Please address any queries, comments, and suggestions to: Philippe Spaas1/Belgium/IBM@IBMBE.

1.4 When should this document be used?

This document assumes the reader is already familiar with the SDS Overview ([9]) and Glossary ([10]), so should only be read after these documents. It should be read either when authoring method changes or artifact descriptions, to ensure that additions and changes to the method content of the Unified Method Framework use consistent concepts, or when analyzing the requirements for a tool directly supporting the SDS concepts, such as SA4TeamSD, or as input to the definition of a UML2 or SysML based profile.

1.5 What accompanies this document?

Besides the current document, the Semantic Specification, the System Description Standard R3.1 consists of the following documents:

- SDS R3.1 Overview presentation: positions the SDS and presents the core concepts ([9]).
- SDS R3.1 Glossary: an informal definition of the concepts required to describe systems ([10]).
- SDS notation documents: define the notations to be used when modeling from either the functional [4] or the operational [8] viewpoint.

2. Summary of Changes

R3.1 is a minor update resulting from the refinements introduced during the development of solution design tooling (SA4TeamSD). The update includes a limited number of new concepts, and relies mainly on the use of already established concepts to support the required refinements.

The models supporting the (IT) System Requirements viewpoint evolved to support the distinction between functional and operational views of the system context. This evolution allowed us to clarify in which specific way ‘actors’, ‘users’ and ‘systems’ are used with the standard. Also, specific stereotypes were introduced for cross system boundary connections and conversations, namely ‘access mechanism’ and ‘exchange’. Furthermore, the notion of ‘requirement’ was introduced to further generalize the already existing relationships for the non-functional requirements. Non-Functional characteristics were repositioned as a ‘Non-Functional Requirement’ at a specific position (‘obtained’) in one of the engineering perspectives (see section 4.2.4).

The System Functional Viewpoint was extended to cover the notion of ‘Service’.

The SDS document itself was reorganized by combining and rewriting the sections dealing with the ‘Organization of Concepts’ and ‘Concept Qualification’, resulting in the ‘Applicability of the System Description Standard’ chapter. As part of this restructuring, we also further developed the notion of ‘Engineering Perspectives’ which has proven particularly useful to limit the introduction of ad hoc concepts within the many different contexts in which systems are modeled.

The distinction between ‘core’ and ‘engineering’ packages was dropped leading to the integration of the System Operational Engineering package into the System Operational package.

3. Introduction

3.1 Principles

3.1.1 The meta-modeling principles

SDS adopts a commonly accepted structure for meta-modeling, based on a four layer scheme. This four layer metamodel hierarchy is illustrated on Figure 1.

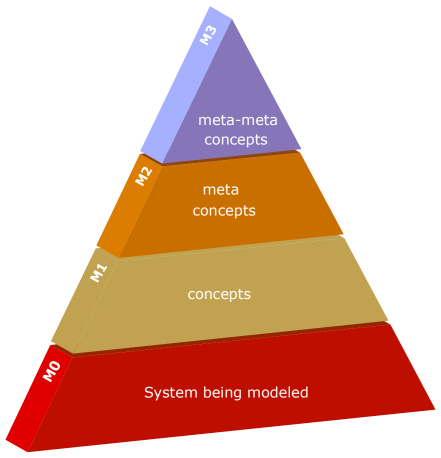


Figure 1: Four layer metamodel hierarchy

Within this hierarchy, the following layers are defined:

- The system being modeled. This layer is sometimes referred to as M0.
- A model layer: this layer contains the concepts that represent things from the system being modeled. This layer is referred to as M1.
- A metamodel layer: this layer contains the meta-concepts. All models, positioned at the model layer, are built from instances of these meta-concepts. This layer is referred to as M2.
- A meta-metamodel layer: this layer contains the meta-meta concepts. All meta-models, positioned at the meta-model layer, are built from instances of these meta-meta concepts. This layer is referred to as M3.

Figure 2 shows how SDS aligns to the four layer metamodel hierarchy.

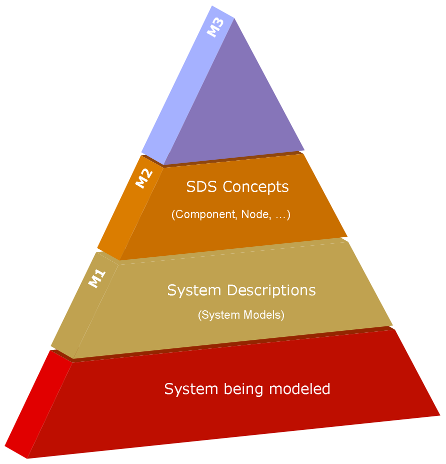


Figure 2: SDS' alignment with the four layer metamodel hierarchy

Figure 3 shows by means of an example how the concepts defined in the SDS are related to both the models built at the M1 level and to the 'System being modeled'.

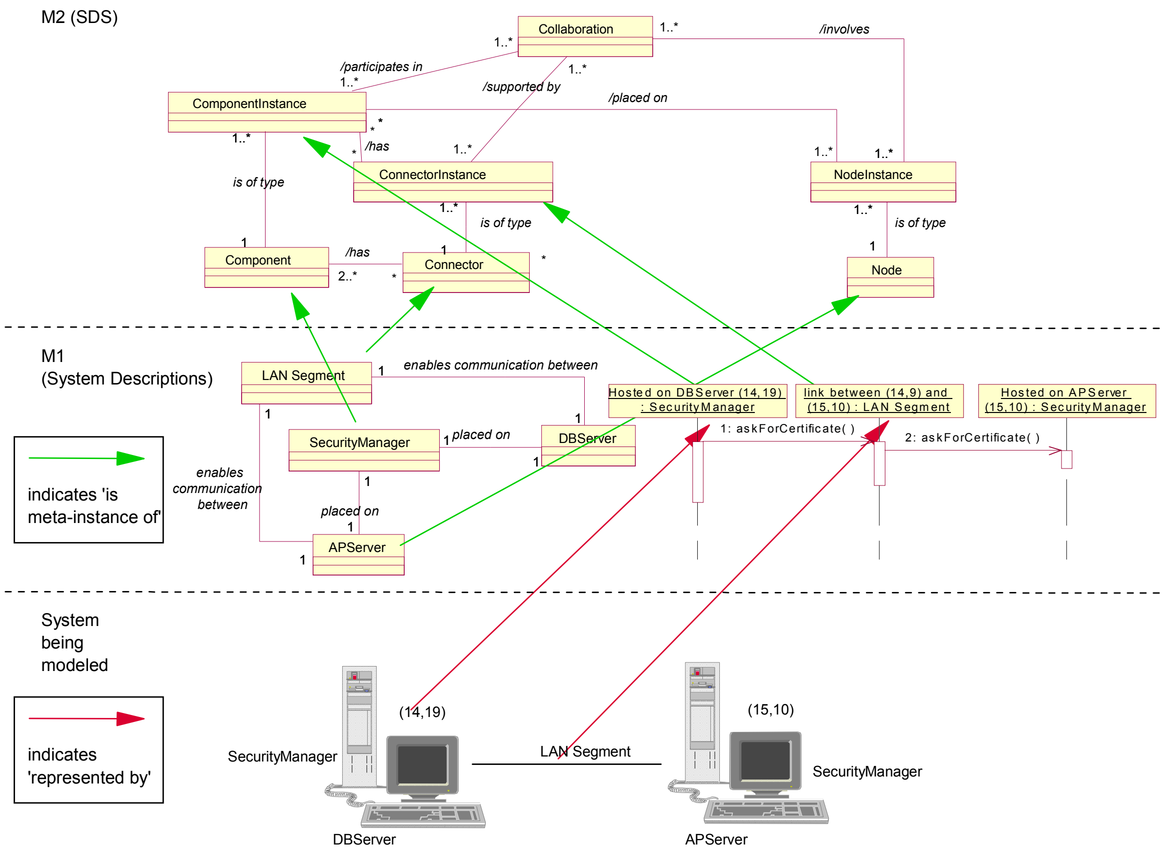


Figure 3: SDS vs. M2/M1/System being modeled

Being positioned at the metamodel layer, SDS defines the (meta) concepts from which system descriptions (models) can be built. These models will support the description of the various basic viewpoints or aspects of a system. For a discussion about the concepts 'basic viewpoints' and 'aspects' we refer to [5].

Whenever an M1 model is built on the basis of the meta concepts defined in an M2 model, the meta concepts are instantiated at the M1 level. In order to draw the distinction between the 'instantiation' taking place between the different 'Mx' levels and the one taking place when a model includes an instance on the basis of a type (specification) whereby both are defined at the same metamodel level, we will label the 'Mx+1 to Mx' instantiation as a 'meta-instantiation' (e.g. the relationship between Node (an M2 concept) and APServer (an M1 concept)) and the latter 'type to instance instantiation' as 'is of type'.

So referring to 0, SecurityManager (positioned at the M1 layer) is the meta-instantiation of Component at the M2 layer, while the 'SecurityManager hosted on DBServer (14,19)' (positioned at the M1 layer) is the instantiation of the SecurityManager (positioned at the M1 layer) and the meta-instantiation of the ComponentInstance at the M2 layer.

3.1.2 Principles underlying SDS

During the development of the SDS, a number of principles have been adhered to:

- Wherever possible, the semantic specification relies on industry standard constructs, adopting the industry's well-defined meaning. In particular, the SDS specification is based on the UML2 specification (See 'Relationship with UML2' for more details about the relationship).
- A number of quality criteria have been applied to the concepts discussed in the semantic specification:
  - parsimony - economy of construct
  - harmony - the model expresses the language of the community (all the concepts practitioners use can either be found in or mapped to from the SDS)
  - adequacy - it serves its purposes, and no more
  - integrity – the model is correct, complete, self-consistent, etc.
  - precision - no ambiguities exist in the defined concepts or the relations amongst them
  - utility - it is useful, desired and above all used
- SDS will support the main concepts used in the artifacts and guidance documents of the IBM Unified Method Framework. The current scope is aligned with the scope of the architecture domain within UADF (UMF). However, SDS itself does not discuss any actual techniques that could be used in developing one or more aspects of a system, nor does it comment on the nature of 'good modeling' or 'correct modeling'. SDS will only provide a definition for the concepts useful during development, by describing how they relate to other concepts, or extend existing concepts.
- Concepts are independent of the use that will be made of the models built from them. Hence, the same concepts will support models established for engineering and for defining 'architecture building block' models (we also refer to the discussion in section 4.2.3).
- It should be recognized that the proposed terms are, at times, not fully consistent with similar industry terms, which are already overloaded. Consistent use of these terms according to their SDS definitions within IBM should allow us to achieve our goal of asset creation without undue confusion caused by these diverging definitions.

3.1.3 Relationship with UML2

During this revision, the current UML2 Superstructure Specification ([2]) was consulted to ensure that SDS R3.1 would align as much as possible with the UML2 standard. In some areas SDS and UML2 have become more closely aligned, in others SDS still goes beyond UML2.

As a consequence, a number of different approaches can be taken for constructing UML2 based profiles for SDS. Hence, the discussion of an actual profile has been removed from the Semantic Specification and will be addressed else where.

3.1.4 Interpreting models built from the concepts of SDS

Practitioners desire to build a number of views of a system model, from a variety of viewpoints, to support their understanding of the architecture, design and the development of a system. These models typically emphasize either static relationships (e.g. via a component relationship diagram) or dynamic behaviours (e.g. via a sequence diagram) between a wide variety of model elements.

For static relationships between system elements, two different interpretations exist about the nature of the elements shown on, for example, the component relationship diagram (which is a model of the static relationships between components):

- One view holds that types are represented on such diagrams, expressing that the relationships shown between these types would apply to all possible instances that could be created according to these type definitions.
- Another view holds that 'representative' instances are shown on such diagrams, expressing that the relationships shown between these representative instances would only apply to the instances in a particular context.

Common sense dictates that, when creating views or other representations (such as tables) of system models, diagrams and tables will always be drawn for either types or representative instances instead of actual instances, since the numbers of instances of a type in a model can be enormous – whether types or representative instances are modeled is therefore a question of practice.

For example, in the case of the component relationship diagram (see [4]), it is established practice to consider that types are represented on the diagrams. Indeed, the vast majority of views of the system model created to support the functional viewpoint will focus on the elements' behavior and information content independent of any particular context, i.e. the components' modeled behaviours will be independent of where these components are deployed in the target system - hence the appropriateness of using types to reflect that the observed behavior applies to all possible instances.

In the case of the operational viewpoint (relying among others on the operational model's relationship diagram), it is much more useful and natural to model representative instances, i.e. a number of instances in a particular context (such as a representative node in a representative location). Effectively, the operational model's relationship diagram needs to show the 'same' elements in different contexts. The need for 'the same' 'Application Services' node in a retail branch office location as well as a warehouse location can only be met by considering that representative instances are placed within these different locations, ensuring that we are able to record the fact that the node's instances have the same responsibilities in both locations, but e.g. have different connections in those locations. Using types to represent this situation would lead to the creation of an additional type every time the same node is used differently in a particular context. Alternatively using a single instance of a node would make any associations made within a particular context immediately applicable everywhere.

This reasoning applies to all model elements shown in views defined by the operational viewpoint on the system model. For example, the locations shown on an operational model are usually treated as being representative for a range of similar locations (such as 'retail branch') so that placement decisions can be made for a representative location instance instead of having to consider each individual location separately. Typically, the number of location instances being represented will be indicated on such models. This custom allows us to limit the number of model elements that need to be documented (given that they are representative of a number of instances in that particular context). These representative instances can be further constrained (i.e. become less representative) when required. For example, a large retail branch representative instance vs. a small retail branch representative instance can be introduced where previously everything was documented in terms of a single representative retail branch instance. Consequently, the diagrams drawn to document the operational viewpoint will be interpreted as showing representative instances.

Dynamic relationships between model elements represented on all diagrams of the system model (both functional and operational) will always be interpreted as relationships between representative instances.

Note that the notion of 'representative' instance lends itself to the following kind of expression: 'a given instance is less representative than another'. This implies that a representative instance can be more constrained (i.e. more of its attributes have been bound to actual values) and therefore represent fewer actual instances than another one (i.e. with fewer of its attributes bound to actual values). Consequently, on a component collaboration diagram we could witness the exchanges between representative component instances (i.e. exchanges which could take place between any instances); while on a walkthrough diagram we could witness fully constrained instances, i.e. representations of the exchanges between actual components hosted on actual nodes located in actual locations.

3.2 Notational conventions

3.2.1 Layout of the meta-model diagrams

When labeling associations in the meta-model, we use the following conventions:

- Labels on associations are to be read left to right, top to bottom
- Associations between concepts do not lead to the corresponding definition of attributes
- Any resolvers of many-to-many associations have been omitted from the model to simplify representation
- Associations are only documented in the concept from which they originate (according to the labeling convention explained above)
- A role played by a concept in a particular association is indicated by a '-' followed by the role name
- Whenever associations between types redefine the associations defined at the level of their respective super types, the association will be redrawn at the level of the types

For reasons of clarity:

- The following concepts from the Model Core package from which elements of SDS inherit, are not repeated on the diagrams: Model Element, Non-Connectable Model Element, Non-Locatable Model Element, Type
- All diagrams show type-level constructs, the names of the corresponding instance level constructs can be easily derived.

3.2.2 Package definition template

Packages are described according to the following template:

- Scope: defines the scope of the package
- Import relationships: defines the concepts which have been imported from other packages
- Overview diagram: shows the relationships between the concepts (both local and imported) in this package
- Concepts: defines the concepts or the extensions made to a concept imported from another package according to the concept definition template.

3.2.3 Concept definition template

In each package, concepts are described according to the following template:

- Definition: defines the concept
    a. this definition is first stated in plain English (1st paragraph)

   b. a formal definition is given in terms of the concepts defined in the metamodel (2nd paragraph) - whenever a reference is made to a concept defined in the metamodel, the name of the concept will be capitalized.

   c. (optional) additional explanation in plain English

- Associations: lists only the associations originating from the concept together with any association classes that might be present. Whenever a reference is made to a concept defined in the metamodel, the name of the concept will be capitalized. Associations are documented in a standard fashion:

   a. Associations without a name are documented as 'associated with'

   b. Aggregations (having a white diamond at one association end) are documented as 'aggregates'

   c. Compositions (having a black diamond at one association end) are documented as 'composes'

- Derived associations (optional): lists the derived associations originating from the concept

- Attributes (optional): lists the attributes

- Related Concepts (optional):

   - stereotypes[1]: lists the concept's stereotypes used in SDS

   - synonyms: well-established concepts that can be directly related to SDS concepts

   - related via an engineering perspective: see section 4.2.4 for a discussion of this topic

Note that when a (stereotyped) concept is extended in another package, the concept is prefixed with the name of the originating package and only those sections of the concept definition template will be present containing the actual extensions.

[1] Throughout SDS, stereotypes will be used to refine (not redefine) the meaning of a concept, for example to emphasize the connection that connects two locations, we use the «Border» stereotype to refine the general notion of a 'connection'. If we want to define a concept which is more specific than its (parent) concept (and thus redefine it), we will define a 'generalization' relationship from the child to the parent (inheritance), for example a component is a specific kind of resource container.

4. Applicability of the System Description Standard

4.1 Introduction

As already indicated by the design principles stated in section 3.1.2, SDS has the objective of defining a concise language for the description of systems addressing the needs of all involved stakeholders. Furthermore, SDS' intended scope also covers models supporting enterprise architecture and solution architecture.

This chapter discusses how SDS tries to reconcile the two potentially conflicting objectives of conciseness and broad coverage. The objective of 'broad coverage' results from our desire to support a wide range of stakeholders and a rich set of models. SDS tackles this challenge via a flexible use of its concepts and the notion of a single system model to support the required stakeholder views.

4.2 Ensuring the flexible use of concepts

4.2.1 Introduction

The SDS modeling language supports the description of architecture in many different contexts and circumstances in line with stakeholders' expectations. By making the concepts useable in many different contexts, we can avoid the introduction of ad hoc concepts to cater for such a wide range of requirements.

This section discusses how the SDS concepts can be used across 3 specific contexts.

4.2.2 Supporting different kinds of systems

The SDS modeling language supports the description of the architecture of systems, irrespective of the actual type of system, i.e. it supports the description of a (general) system, an IT system,… Although the specific interpretation of the concepts will vary slightly across contexts (e.g. the notion of a component in a business system refers to the combination of human, IT and other resources, while in an IT system it refers to a combination of function and data), these concepts nevertheless express the same basic idea (in this example a grouping of resources delivering services) which remains relevant and useful across different contexts.

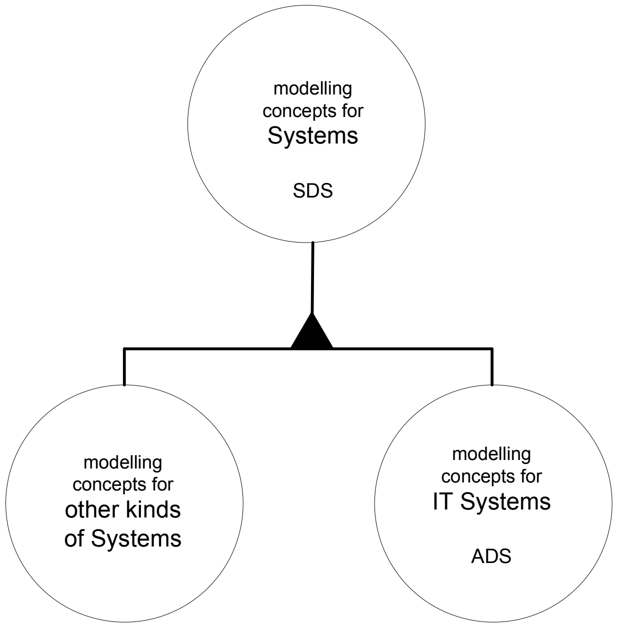

modelling concepts for Systems
SDS
modelling concepts for other kinds of Systems
modelling concepts for IT Systems
ADS

Figure 4: Modeling concepts for describing multiple kinds of systems

Figure 4 illustrates how the SDS concepts are organized to support the notion of generalized modeling concepts (for systems), while allowing for a richer set of meanings and relationships within a specific context, e.g. within the IT system context. 'SDS' is used to refer to the generalized modeling language, while ADS is used to refer to the IT System specific modeling language.

All concepts applicable to systems in general will be positioned at the level of the modeling concepts for systems (e.g. 'component'), those specific to the context of IT systems will either be suitably refined from the more generalized concept, or introduced specifically within that context (e.g. deployment unit).

This approach allows us to balance the need for generality, i.e. defining the concepts at the most general level (the system level), vs. the need for having meaningful relationships between the concepts within a specific context, hence their refinement within that context. Moreover, it also supports strong relationships between system models created in different but inter-related contexts, such as IT components in an IT system that support the business components in a business system.

Note that the current version of the SDS only focuses on the concepts within the system and IT system contexts and where required, the context will be made explicit via the use of the 'IT' qualifier.

4.2.3 Supporting different modeling purposes

Models of systems can be created for many different purposes, among which we can distinguish between models with a focus on the engineering of a particular system (a system solution model) and those with a focus on the description of a 'catalogue of parts' documenting the parts from which systems and their models can be built (architecture building block models).

It is important that the same underlying modeling language is used for building these models. The use of the same language ensures a straightforward incorporation of the architecture building block models into the models supporting the engineering of a single system (system solution models).

The present document organizes the concepts and relationships from the perspective of system solution models; however it should be clearly noted that the same concepts support the architecture building block models, but subtle differences in the relationships between these concepts may exist.

4.2.4 Supporting different engineering perspectives

During the course of the engineering of a system model, elements from the system model evolve as they reflect the results of the engineering effort, e.g. product & technology selections are made, model elements are progressively detailed, …. We use 'engineering perspective' to limit our focus to (a) specific (set of) attributes of a model element e.g. level of detail of documentation, product/technology decision taken, … To be able to differentiate between (the same) concepts having different values for these attributes (e.g. 'as-is' vs. 'to-be'), concepts will be explicitly qualified by these values. Specific relationships can exist between concepts at different positions (i.e. having different values for the relevant attribute) along the same engineering perspective. It should also be clear that multiple engineering perspectives can be applicable to any given model element and this at the same moment in time, e.g. a component can be considered along the logical-physical perspective, while at the same time also along a time perspective (as-is vs. to-be), leading to the notion of e.g. a 'to-be physical component'.

In line with the overall principle of keeping the modeling language concise, the same modeling concepts are used along an engineering perspective.

This section will identify a number of engineering perspectives which are applicable to all model elements and model element relationships defined in the SDS.

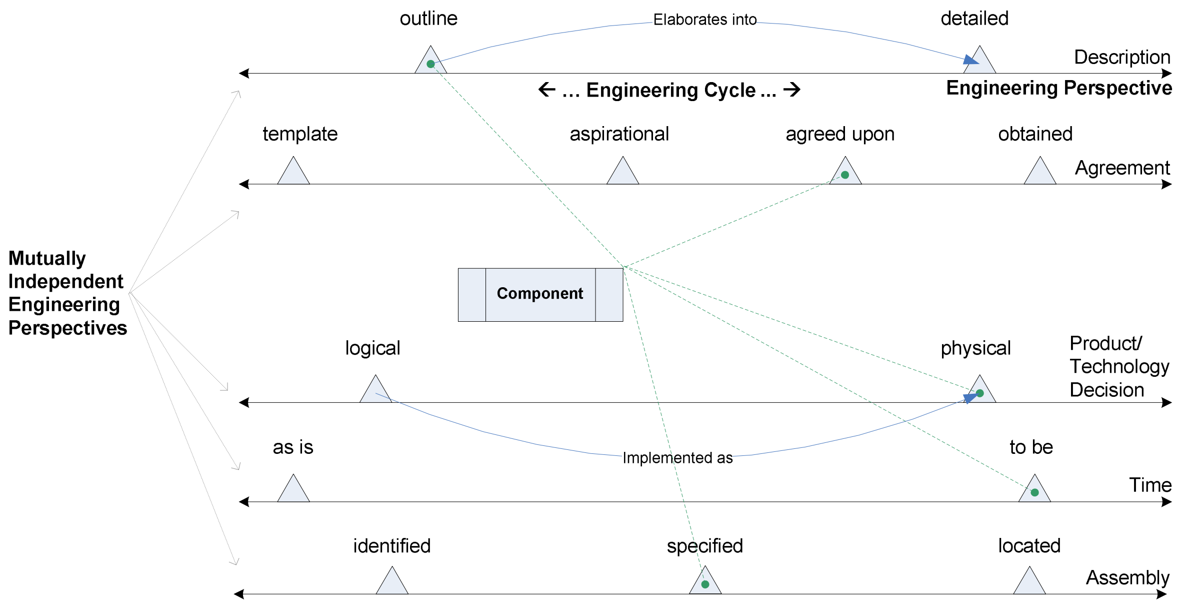


Figure 5: Engineering Perspectives

The following is a non-exhaustive list of engineering perspectives:

Engineering Perspective: Product/Technology decision

All model elements and model element relationships can be presented in their logical form ('what they do'), as well as in their physical form ('what they are').

The relationship between a logical and physical model element or between a requirement and a specification meeting that requirement, is denoted as 'implemented'.

Engineering Perspective: Description

All model elements and model element relationships get better defined over time - they elaborate. Hence, if helpful, words like 'draft', 'outline', 'revised', 'refined' and 'final' may be used to qualify the model elements or model element relationships.

The relationship between model elements/model element relationships at an increasing level of detailing/coverage is denoted as 'elaborates into'.

The relationship between these elements at a decreasing level of detail/coverage is denoted as ‘abstracts into’.

The relationship between these elements at an increasing level of confidence is denoted as ‘improves into’.

The relationship between these elements at a decreasing level of confidence is denoted as ‘approximates to’.

Engineering Perspective: Time

All model elements and model element relationships can be positioned explicitly in time, leading to a qualification by a time indicator (for example: ‘as-is’, ‘to-be (point in time 1)’, ‘to-be (point in time 2)’, …) of model elements or model element relationships.

Engineering Perspective: Agreement

All model elements and model element relationships can be positioned in relation to the level of agreement reached between the involved parties, leading to the use of qualifiers like: ‘desired’, ‘proposed’, ‘standard’, ‘agreed upon’, ‘obtained’, ….

Engineering Perspective: Assembly

All model elements and model element relationships can be tracked along the engineering cycle in terms of their relation with respect to the target system, namely: ‘identified’, ‘specified’, ‘located’ (i.e. assigned to a location within the target system), …

4.3 Supporting stakeholders via a single model

4.3.1 Introduction

Besides offering flexible concepts remaining meaningful in many different contexts, SDS also needs to address the requirements of all involved stakeholders.

This section focuses firstly on the notion of a system model, underlying the stakeholder views of a system. Subsequently, some additional relationships between models (or views onto the system model) are discussed together with their influence on SDS.

4.3.2 Different views on a single model of the system

As described in [5], the overall description of a system, via its corresponding system model, can be further organized via a number of basic viewpoints, referred to as ‘viewpoints’ in the remainder of this document. This is shown in the figure below.

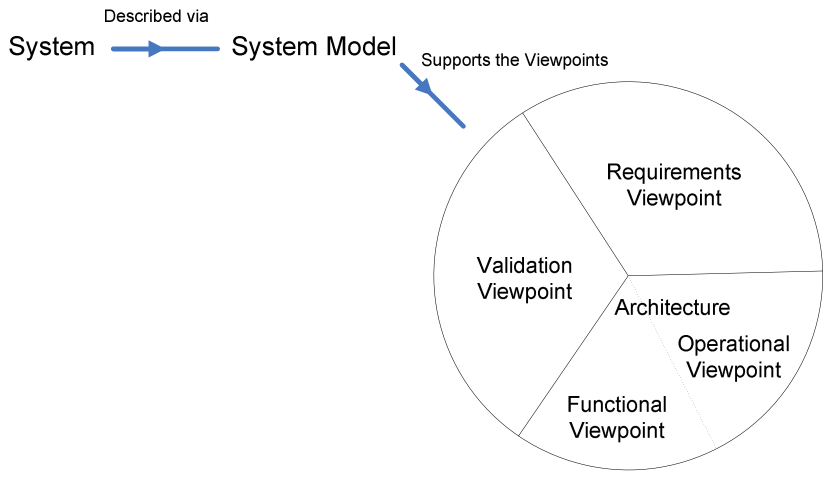


Figure 6: Supporting the system description via viewpoints

It should be noted that each of the different viewpoints defines which selection of model elements/model element relationships from the system model is required to deliver its associated view. Besides selecting specific kinds of model elements/model element relationships, basic viewpoints can also apply additional filtering criteria to the selected elements, e.g. a distinction can be made between application level elements (i.e. elements of direct interest to the user) and technical level elements (i.e. elements of no direct interest to the user), leading to application/technical models.

SDS/ADS is the language via which the system is described through the system model, emphasizing the intent of having a single set of concepts underlying not only the system model but also all the views derived from it.

The description of the SDS/ADS concepts has been organized on the basis of these standard viewpoints. Consequently, model elements will be discussed in the context of the viewpoint with which they have become most closely associated through custom and practice. As an example, ‘Resource Container’ is present in the requirements and functional viewpoint related models; however, the concept will be defined in the System Functional package, and imported by the System Requirements package.

4.3.3 Black box vs. white box views

One of the proven ‘divide and conquer’ strategies to control complexity consists of defining black box/white box models. As part of this approach a given (black box) model is further detailed by a so-called white box model which models the internal structure of the black box. In this way, different models are related to one another (the black box model relates to its corresponding white box model) whereby both models are expressed using the same language.

The Model Elements/Model Element Relationships belonging to the white box (internal) view ‘realize’ those belonging to the black box (external) view. Note that the Model Elements/Model Element Relationships realizing the Model Elements/Model Element Relationships associated with the ‘external’ view can either be of the same type or of different types, consider e.g. the model of a target system (black box model) and the corresponding operational model, established according to the operational viewpoint, where many more concepts are used to describe this white box model, like nodes, connections, zones, …

4.3.4 Accurate vs. approximate modeling

Models can be built with different degrees of accuracy[2], modelers can focus on documenting all the individual parts of direct interest to the user (the application level) which make up a system, each with an explicit definition of non-functional requirements/characteristics, together with the support required to make these parts work together, i.e. the parts not directly of interest to the user (the technical level), again

[2] Note for non-native English readers: ‘accurate’ is defined as ‘conform to the truth’ and should be distinguished from ‘precise’ which is defined as ‘sharply defined’.

each with their associated non-functional requirements/characteristics. In everyday practice, systems will only be partially documented at this level of accuracy, and the majority of models will document systems by using ‘placeholders’, implicitly aggregating the individual parts and the support they require to work together. As a consequence, these placeholders become associated with the properties of the aggregated elements, and will allow for an understanding at a higher level of abstraction of the system, albeit with less accuracy.

Within the description standard a specific set of concepts is used to support these accurate models (e.g. resource container, component, connector, etc..), and another set to support the approximate models (e.g. nodes, connections, etc..). The latter are then defined as aggregations of the former (e.g. a node is defined as a collection of resource containers) and will effectively function as such a ‘placeholder’.

This approach allows for approximate modeling i.e. only aggregate properties are known of e.g. the nodes/connections and on many occasions this is sufficient. On other occasions we want to have accurate models, and consequently we will detail nodes/connections as aggregations of components, deployment units, and connectors.

5. Core Concepts of the Description Standard

5.1 Introduction

This chapter lays the foundations for the concepts used in modeling systems. It focuses on defining a number of base classes from which all other SDS/ADS concepts inherit.

5.2 Core modeling concepts: the Model Core package

Scope

The Model Core package defines the concepts and the relationships which are the foundation for all other SDS/ADS concepts.

Import Relationships

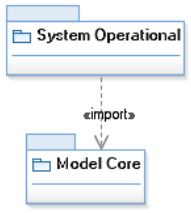


Figure 7: Import relationships of the Model Core package

Overview Diagram

The concepts and relationships of the Model Core package are described in the following diagrams:

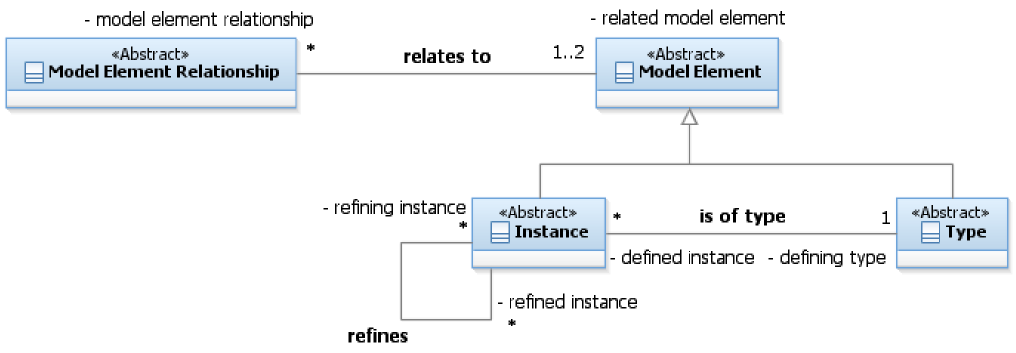


Figure 8: Concepts/Relationships of the Model Core package

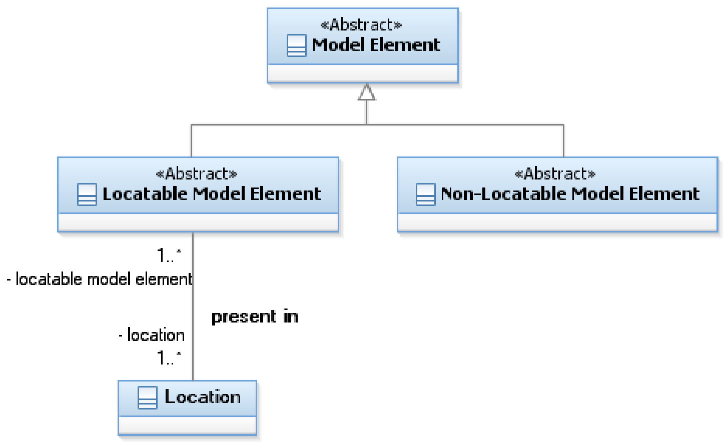


Figure 9: The notion of Locatable and Non-Locatable Model Elements

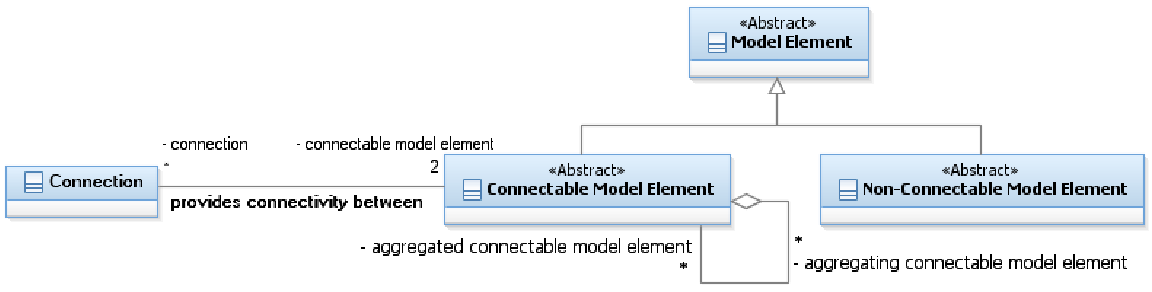

«Abstract»
Model Element
- connection
*
- connectable model element
2
Connection
provides connectivity between
«Abstract»
Connectable Model Element
«Abstract»
Non-Connectable Model Element
- aggregated connectable model element
*
*
- aggregating connectable model element

Figure 10: The notion of Connectable and Non-Connectable Model Elements

These diagrams show the relationships between model elements as these would appear in a valid system model, i.e. the cardinalities of the associations reflect valid associations between model elements in a complete system model and do not reflect the fact that during the actual engineering of the system, certain required associations between model elements may not yet have been decided upon or documented.

Concepts

Connectable & Non-Connectable Model Element

Definition

A connectable model element is a model element that has the capability to be connected to another connectable model element via a connection. Non-connectable model elements do not have that capability.

In the metamodel, the Connectable Model Element and Non-Connectable Model Element inherit from Model Element and serve as abstract base classes which will be sub typed in other SDS packages. Every Model Element inherits from either Connectable Model Element or Non-Connectable Model Element. Connectable Model Elements can aggregate other Connectable Model Elements and can connect to other Connectable Model Elements via a Connection.

Where useful the inheritance from either connectable or non-connectable model element has been shown in the views of the SDS metamodel. Chapter 9 contains the overview of all these inheritance relationships.

Associations

[Connectable Model Element as aggregating connectable model element] aggregates (to *) [Connectable Model Element as aggregated connectable model element]:* A Connectable Model Element can aggregate any number of Connectable Model Elements, while a Connectable Model Element can be aggregated by any number of Connectable Model Elements.

System Operational::Connection

Associations

provides connectivity between (to 2) Connectable Model Element:* A Connection provides connectivity between 2 Connectable Model Elements. A Connectable Model Element can have any number of Connections.

Instance

Definition

An instance is a model element, which has an associated definition or specification (type).

In the metamodel, an Instance inherits from Model Element. It is related to the Type providing its definition.

Throughout this document, the concepts have been represented with their ‘Type’ level names. The names used to refer to their corresponding instances can be easily derived.

Associations

[Instance as refined instance] refines (to *) [Instance as refining instance]:* An Instance can refine any number of other Instances whenever the refining Instance is more constrained than the refined one, i.e. more of its attributes have been bound to actual values. A certain Instance can be considered to have been refined with respect to any number of other Instances, when this Instance is less constrained than the Instances it is being compared to. Also when an Instance refines another, the features of the refining Instance are systematically related to the features of the refined Instance. For example, if the refined Instance defines a particular attribute, the refining Instance can define a refined version of the same attribute.

[Instance as defined instance] is of type (to 1) [Type as defining type]:* An Instance conforms to the definition (specification) of one Type. A Type can define any number of Instances. Conformance implies that an Instance has only and all of the attributes and associations that are defined for its Type. This association emphasizes the relationship between an Instance and the Type which was taken as its corresponding definition, and not the relationship with the possibly many subtypes from which that Type inherited.

Locatable & Non-Locatable Model Element

Definition

A locatable model element can be assigned to a location. Conversely, a non-locatable model element cannot be assigned to a location.

In the metamodel, Locatable Model Element and Non-Locatable Model Element inherit from Model Element and are abstract base classes which will be sub typed in other SDS packages. Every Model Element inherits from either Locatable Model Element or Non-Locatable Model Element.

Where useful the inheritance from either locatable or non-locatable model element has been shown in the views of the SDS metamodel, chapter 9 contains the overview of all these inheritance relationships.

Associations

present in [1.. to 1..*] Location:* A Locatable Model Element is present in at least one Location. A Location has at least one Locatable Model Element present.

Model Element

Definition

A model element is a part of the system being modeled.

In the metamodel, a Model Element is an abstract named entity. It is the base type for all concepts in SDS.

Attributes

name: An identifier for the Model Element

Model Element Relationship

Definition

A model element relationship is a relationship established between one or two model elements.

In the metamodel, a Model Element Relationship is an abstract named entity. It is the base for the relationships defined in the SDS.

Associations

relates to (to 1..2) [Model Element]:* A Model Element Relationship establishes a relationship between one or two Model Elements. A Model Element can be involved in any number of Model Element Relationships.

Attributes

name: An identifier for the Model Element Relationship

Type

Definition

A type defines the properties and behavioral characteristics of a set of instances.

In the metamodel, a Type inherits from Model Element and provides the definition for a number of Instances.

6. System Description Standard (SDS)

6.1 Introduction

The Architecture Description Standard for Systems (SDS) defines the concepts used in modeling systems.

Throughout the discussion, we will assume that the boundary of the system under scrutiny does not change and that there is only a single target system. Although we could further decompose any system into a set of subsystems, and then consider these in turn as systems in their own right (and adjust the scope accordingly, while remaining in the same context, by for example considering these other systems, i.e. the other subsystems of the system as initially defined, to be actors in relation to our focus system, i.e. the focus system being one of the subsystems of the system as initially defined), we will refrain from doing so in order to maintain a consistent backdrop for our concept related discussions. The impact of either maintaining the system boundary or changing it as part of recursive decomposition is shown in the figure below.

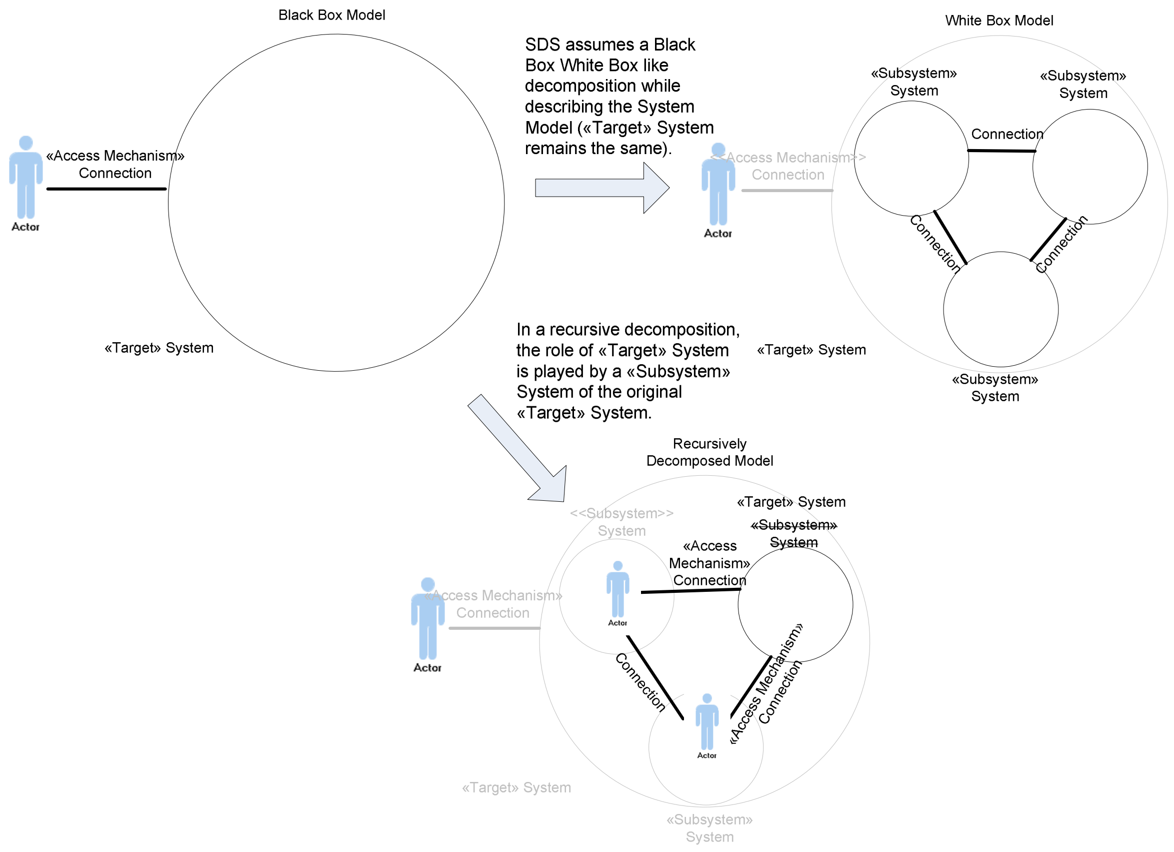

Black Box Model
«Access Mechanism» Connection
Actor
«Target» System
SDS assumes a Black Box White Box like decomposition while describing the System Model («Target» System remains the same).
White Box Model
«Subsystem» System
«Subsystem» System
Connection
<<Access Mechanism>> Connection
Actor
Connection
Connection
«Target» System
«Subsystem» System
In a recursive decomposition, the role of «Target» System is played by a «Subsystem» System of the original «Target» System.
Recursively Decomposed Model
<<Subsystem>> System
«Target» System
~~«Subsystem» System~~
«Access Mechanism» Connection
«Access Mechanism» Connection
Actor
Actor
Connection
«Access Mechanism» Connection
Actor
«Target» System
«Subsystem» System

Figure 11: Maintaining the system boundary in our description of the system model

After an explanation of the overall package structure of the SDS, we will introduce the definition of system and subsequently the main concepts will be defined, organized by viewpoint (see [5]).

6.2 Package structure

Figure 12 represents the overall package structure of the SDS. Note that the arrows flow from the supplier to the client.

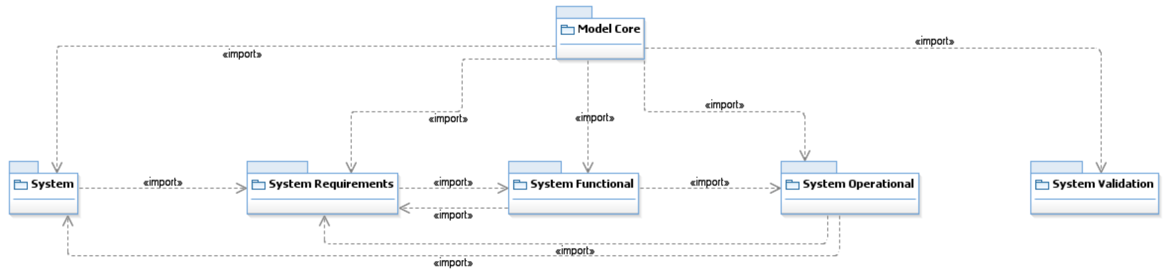


Figure 12: SDS Package structure

6.3 The notion of a System: the System package

Scope

The scope of the System package includes those concepts, which are used to model entire systems and their subsystems.

Import Relationships

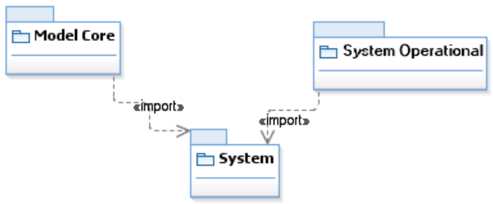


Figure 13: Import relationships of the System package

Overview Diagram

The following diagrams show the relationships between model elements as these could occur in a valid system model, i.e. the cardinalities of the associations reflect what valid associations exist between model elements in a complete system model and do not reflect the fact that during the actual engineering of the system, certain required associations between model elements may not yet have been decided or documented.

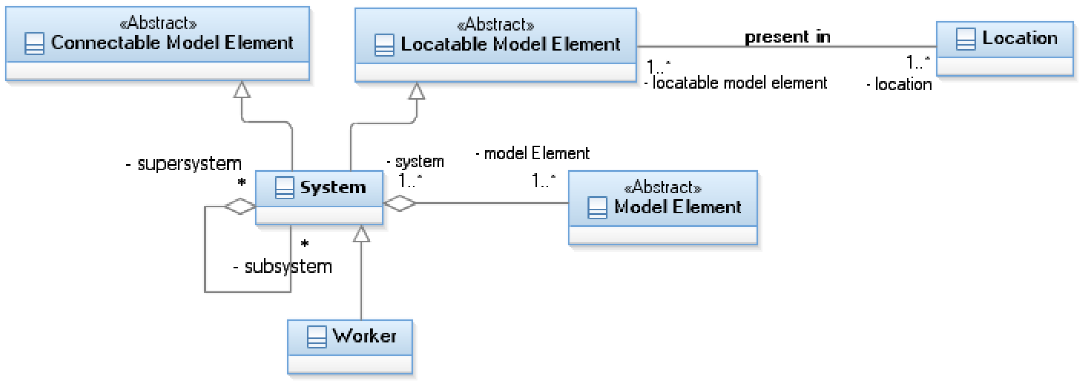


Figure 14: Concepts/Relationships of the System package

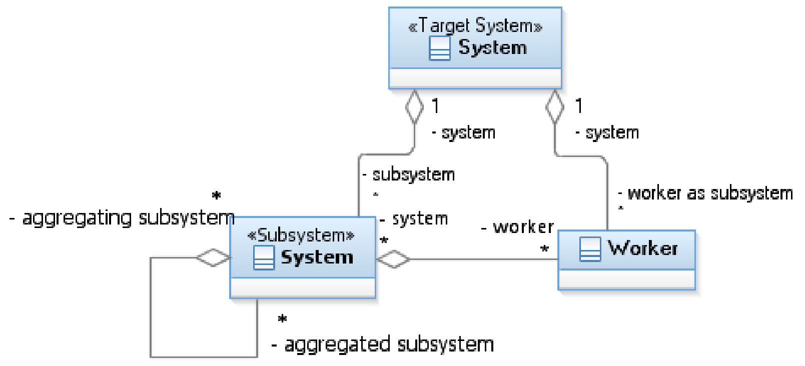


Figure 15: Concepts/Relationships of the System package for «Target System» System

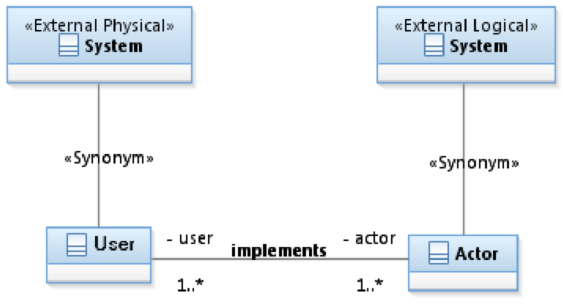


Figure 16: Actor/User as synonyms of «External System» System

Concepts

System Operational::Location

Definition

Documented elsewhere.

System

Definition

A system is a collection of model elements and model element relationships organized and working together to accomplish a specific function or set of functions. A system exists to fulfill one or more missions in its environment.

This definition (adapted from [1]) relies on the following additional definitions:

- the environment, or context, determines the setting and circumstances of developmental, operational, political, and other influences upon that system.
- a mission is a use or operation for which a system is intended by one or more stakeholders to meet some set of objectives.
- a stakeholder is an individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system.

The concepts defined in this document, together with those defined in UML (see [2]) will typically be used in system descriptions. In this sense, a system is related to all concepts defined in the current document.

In the metamodel, a System inherits from both the Locatable Model Element and the Connectable Model Element abstract types. A System can be decomposed into subsystems and aggregated into super-systems. A System is also the base type for Worker which is considered to be a specific kind of (human) System. It also aggregates Model Elements, hence it acquires all the properties of these Model Elements, and consequently can behave as a Node (e.g. have an «Access Mechanism» Connection) or as a Resource Container (e.g. support Interfaces).

Associations

present in [1.. to 1..*] Location:* A System has a presence in one or more Locations if any of the Model Elements of which the System is made up are present in these Locations. In a Location, at least one Locatable Model Element is present.

[System as supersystem] aggregates [to *] [System as subsystem]: A System can be decomposed into any number of subsystems. A System can be a part of any number of super-systems.*

aggregates [1.. to 1..*] Model Element: A System aggregates at least one Model Element. A Model Element is aggregated into at least one System.*

Related Concepts

Stereotypes:

«*External System*» *System*

«*Subsystem*» *System*

«*Target System*» *System*

«External System» System

Definition

An external system refers to any system interacting with the target system.

In the metamodel, the external system is modeled as a System with a stereotype of «External System».

Related Concepts

Synonyms:

Actor is a synonym of a logical[3] «External System» System

User is a synonym of a physical «External System» System

Associations

User implements [1.. to 1..*] Actor:* A User implements at least one Actor. An Actor is implemented by at least one User.

«Subsystem» System

Definition

A subsystem, which is a system in its own right, is any subset of the model elements and model element relationships of a system, e.g. when reasoning within the context of an Enterprise System, this system can be decomposed into an IT subsystem and a business subsystem, each of which establishes a

[3] 'logical' and 'physical' as defined in the 'Product/Technology' engineering perspective (section 4.2.4)

different context (as discussed in section 4.2.2). The decomposition into subsystems may or may not cross these contexts.

In the metamodel, a «Subsystem» System is a stereotype of System and can be further decomposed into additional «Subsystem» Systems.

Associations

(«Subsystem» System as aggregating subsystem) aggregates [..*] («Subsystem» System as aggregated subsystem):* A «Subsystem» System can aggregate any number of «Subsystem» Systems and a «Subsystem» System can be aggregated into any number of «Subsystem» Systems.

«Subsystem» System aggregates [..*] Worker:* A «Subsystem» System can aggregate any number of Workers and a Worker can be aggregated into any number of «Subsystem» Systems.

«Target System» System

Definition

The target system is the system being modeled and engineered such that it complies with the requirements imposed on it by its associated actors.

In the metamodel, the target system is modeled as a System with a stereotype of «Target System». A «Target System» can be decomposed into any number of «Subsystem» Systems and can aggregate Workers.

Associations

*aggregates [1 to *] «Subsystem» System:* A «Target System» System can be decomposed into any number of «Subsystem» Systems. A «Subsystem» System is part of a single «Target System» System.

*aggregates [1 to *] Worker:* A «Target System» System can be decomposed into any number of Workers. A Worker is part of a single «Target System» System.

Worker

Definition

A worker is a person who is part of the target system.

In the metamodel, a Worker is modeled as a subtype of System and belongs to the overall «Target System», emphasizing that he/she is part of the target system and of some of its subsystems.

6.4 The System Requirements viewpoint supported by the System Requirements package

Scope

The scope of the System Requirements package contains those concepts which are involved in the modeling of a system’s requirements. Our context is a single system, implying that we ignore the fact that requirements and related concepts could be applicable to any number of systems.

Import Relationships

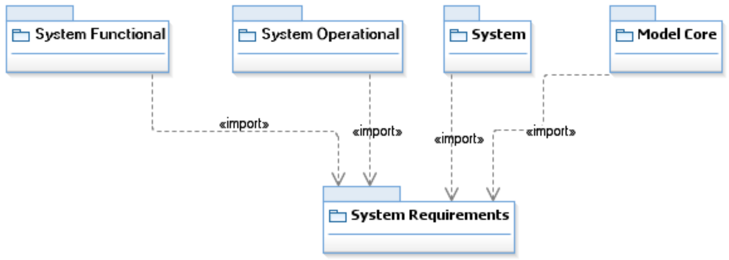


Figure 17: Import relationships of the System Requirements package

Overview Diagram

The following diagrams show the relationships between model elements as these could occur in a valid system model, i.e. the cardinalities of the associations reflect what valid associations exist between model elements in a complete system model and do not reflect the fact that during the actual engineering of the system, certain required associations between model elements may not yet have been decided or documented.

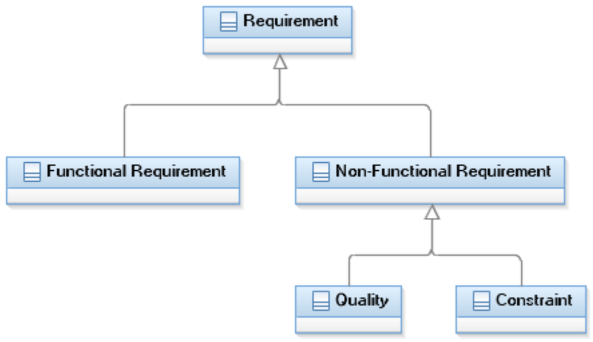


Figure 18: Concepts of the System Requirements viewpoint

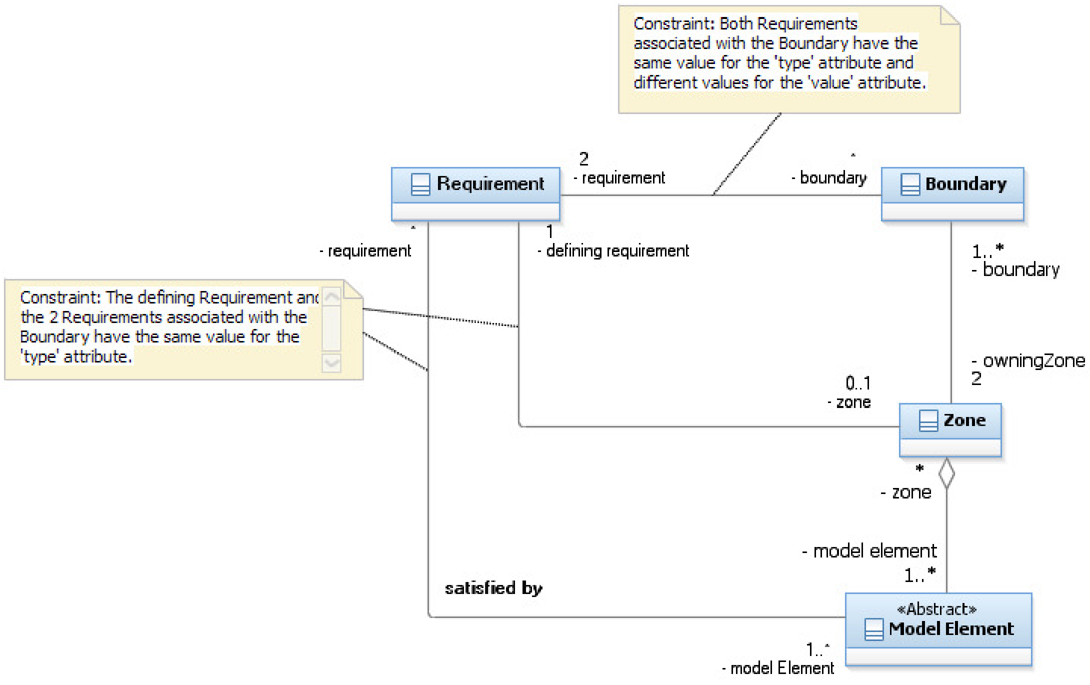

Constraint: Both Requirements associated with the Boundary have the same value for the 'type' attribute and different values for the 'value' attribute.
Requirement
2
- requirement
*
- boundary
Boundary
*
- requirement
1
- defining requirement
1..*
- boundary
Constraint: The defining Requirement and the 2 Requirements associated with the Boundary have the same value for the 'type' attribute.
- owningZone
2
0..1
- zone
Zone
*
- zone
- model element
1..*
satisfied by
«Abstract»
Model Element
1..*
- model Element

Figure 19: The Model Core Package extended for the System Requirements viewpoint

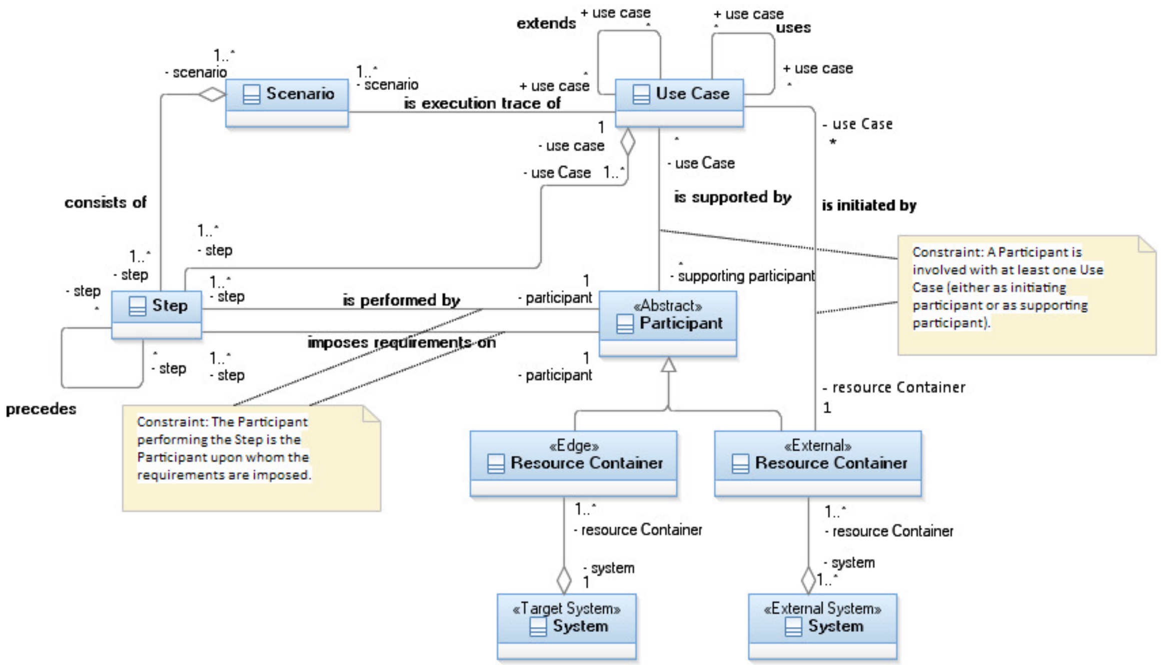


Figure 20: Concepts/Relationships of the System Requirements viewpoint – supporting the functional viewpoint

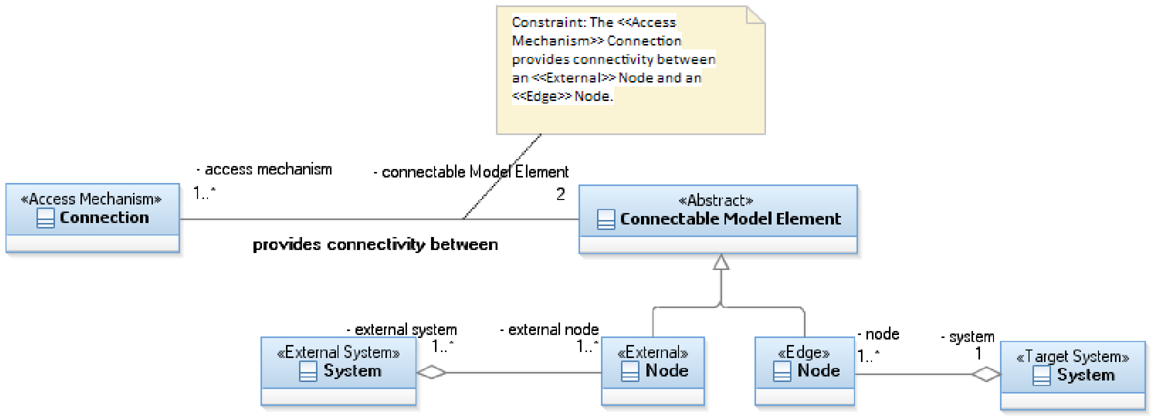

Constraint: The <<Access Mechanism>> Connection provides connectivity between an <<External>> Node and an <<Edge>> Node.
- access mechanism
1..*
- connectable Model Element
2
«Access Mechanism» Connection
«Abstract» Connectable Model Element
provides connectivity between
- external system
1..*
- external node
1..*
«External System» System
«External» Node
«Edge» Node
- node
1..*
- system
1
«Target System» System

Figure 21: Concepts/Relationships of the System Requirements viewpoint – supporting the operational viewpoint

Concepts

Actor (as synonym of «External Logical» System)

Definition

An actor describes an external logical system which interacts with the target system.

In the metamodel, an Actor is a synonym of an «External Logical» System. An Actor will exhibit the properties of the concept it represents, and consequently behaves as an «External» Node (e.g. have an «Access Mechanism» Connection) when we emphasize the operational viewpoint, or as an «External» Resource Container (e.g. support Interfaces) when we emphasize the functional viewpoint.

Boundary

Definition

A boundary is associated with a change in value for a particular requirement between two model elements. As such, a boundary has ‘zero width’ – no model element can be ‘on’ a boundary, only ever on one side or the other.

In the metamodel, a Boundary is associated with two different values for a particular Requirement and defines the transition between the two involved Zones, each associated with one of these values.

Associations

associated with (1.. to 2) [Zone]:* A Boundary indicates the change of values for a given Requirement between 2 Zones. A Zone has at least one Boundary given that a Zone can have a relationship with many different Zones and having no Boundary would imply that a single Zone contained all of the System’s elements, which is not deemed relevant.

System Operational::«Access Mechanism» Connection

Definition

An access mechanism defines the necessary connectivity between an external node, part of an external system, and an edge node, part of a (target) system, such that the interactions between this external system and the (target) system can take place. Access mechanisms refer to those connections that cross the target system boundary.

In the metamodel, Access Mechanism is a stereotype of Connection, and is associated with «External» Nodes and «Edge» Nodes.

Associations

provides connectivity between (..1 to 2) [Connectable Model Element]:* An Access Mechanism provides connectivity between two Connectable Model Elements, namely an «External» Node and an «Edge» Node. An «External» Node and an «Edge» Node have at least one Access Mechanism.

Constraint: The «Access Mechanism» Connection provides connectivity between an «External» Node and an «Edge» Node.

Constraint

Definition

A constraint refers to conditions with which the engineering of the target system and/or the target system itself have to comply.

In the metamodel, a Constraint is a subtype of Non-Functional Requirement.

All kinds of constraints on a system are represented using this construct, including business constraints (e.g. geography of locations), IT standards (e.g. WS-Security compliance), current infrastructure constraints (e.g. must run on specified existing middleware), and specific development skills.

Functional Requirement

Definition

A functional requirement defines the desired behavior of the target system.

In the metamodel, a Functional Requirement is a sub type of Requirement.

Model Core::Model Element

Definition

Documented elsewhere.

System Operational::«Edge» Node

Definition

Documented elsewhere.

Associations

aggregated into (1.. to 1) [«Target System» System]:* An «Edge» Node belongs to the «Target System». A «Target System» System aggregates at least one «Edge» Node.

System Operational::«External» Node

Definition

Documented elsewhere.

Non-Functional Requirement

Definition

A non-functional requirement (NFR) is a quality requirement or constraint that (some part of) a system must satisfy, typically related to an associated set of circumstances (such as a NFR of ‘5 second response time in 95% of the cases’ when ‘remotely connected’).

In the metamodel, a Non-Functional Requirement inherits from Requirement.

Related Concepts

Related via an engineering perspective:

Non-functional characteristics are positioned as a ‘Non-Functional Requirement’ at a specific position (‘obtained’) in the ‘agreement’ engineering perspective.

System Functional::Participant

Definition

Documented elsewhere.

Quality

Definition

A quality refers to an intrinsic characteristic of a (part of the) target system.

In the metamodel, a Quality inherits from Non-Functional Requirement.

Qualities (such as performance, availability and security) are an important class of non-functional requirements. Maintainability is also considered a quality, even though it is more related to attributes of the development process, than to attributes of the target system (as do service levels).

Requirement

Definition

A requirement is the expression of a need. Requirements are used to define the desired behavior and quality of the target system.

In the metamodel, a Requirement needs to be satisfied by one or more Model Elements. A Requirement can be used to define a Zone and different values for these Requirements are then associated with the Boundaries between the Zones it defines.

Associations

satisfied by (to 1..*) [Model Element]:* A Model Element is subject to any number of Requirements. Requirements are satisfied by at least one Model Element.

*associated with (2 to *) [Boundary]:* A Requirement is associated with any number of Boundaries, and a Boundary is associated with two Requirements.

<u>Constraint:</u> Both Requirements associated with the Boundary have the same value for the 'type' attribute and different values for the 'value' attribute.

[Requirement as defining requirement] (1 to 0..1) [Zone]: A Requirement is the basis for the definition of at most one Zone. A Zone is defined on the basis of one Requirement.

<u>Constraint:</u> The defining Requirement and the 2 Requirements associated with the Boundary have the same value for the 'type' attribute.

Attributes

type: type of requirement, e.g. performance, availability, security, capacity, throughput, maintainability, etc. related.

units: units in which requirement is measured (e.g. seconds, Mb, hours available, Boolean).

measurement: procedure, mechanism or convention for measuring/interpreting actual value. (e.g. response time is measured from end-to-end, from the user pressing a key/mouse to the response screen being fully displayed).

Where the requirement is used to represent a constraint (a predicate returning a binary value), this is a description of the constraint, which the associated objects must satisfy, (e.g. 'Year 2000 compliant', or 'runs on Linux').

value

System Functional::«Edge» Resource Container

Definition

The edge resource container, part of the target system, is the resource container with which the external resource container, part of the external system, interacts.

In the metamodel, an «Edge» Resource Container is a stereotype of Resource Container.

Associations

aggregated into (1.. to 1) [«Target» System]:* An «Edge» Resource Container belongs to the «Target» System. A «Target» System aggregates at least one «Edge» Resource Container.

System Functional::«External» Resource Container

Definition

The external resource container, part of the external system, is the resource container with which the edge resource container, part of the target system, interacts.

In the metamodel, an «External» Resource Container is a stereotype of Resource Container.

Associations

aggregated into (1.. to 1..*) [«External System» System]:* An «External» Resource Container belongs to at least one «External System» System. An «External System» System aggregates at least one «External» Resource Container.

Scenario

Definition

A scenario is the trace of an execution of a use case under well specified circumstances. In contrast to a use case which contains various alternatives, a scenario only details one specific execution of its associated use case according to a specific circumstance.

In the metamodel, a Scenario is associated with the Use Case of which it is an execution trace and the Steps out of which the Scenario consists.

Associations

is execution trace of (1.. to 1) [Use Case]:* A Scenario is an execution trace of a Use Case. A Use Case has at least one associated Scenario.

aggregates (1.. to 1..*) [Step]:* A Scenario consists of at least one Step. A Step appears in at least one Scenario.

Attributes

system assumptions: The assumptions made concerning the global state of the «Target System» System that will affect the execution of the Scenario.

external assumptions: The assumptions made about the state of the external world (i.e. outside the «Target System» System). This is typically reflected either by Actor behavior, or by input parameters.

outcome: The outcome of the Scenario.

Step

Definition

A step is an elementary piece of behavior of a participant, being either the target system or an actor\user (external system), observable from either an actor\user by the target system, or from the target system by an actor\user.

In the metamodel, a Step is associated with both Scenarios and Use Cases and can be preceded and succeeded by any number of Steps. A Step is performed by a Participant and becomes the origin of the requirements of the participant performing the step.

Associations

imposes requirements on (1.. to 1) [Participant]:* A Step imposes requirements on a single Participant. A Participant has requirements imposed on it by at least one Step.

is performed by (1.. to 1) [Participant]:* A Step is performed by a single Participant as part of his participation in the execution of a Scenario. A Participant performs at least one Step.

Constraint: The Participant performing the Step is the Participant upon whom the requirements are imposed.

precedes (to *) [Step]:* A Step precedes any number of Steps, and a Step is preceded by any number of Steps.

System::«External System» System

Associations

aggregates (1.. to 1..*) [«External» Node]:* An «External System» System aggregates at least one «External» Node. An «External» Node is aggregated in at least one «External System» System.

System::«Target System» System

Definition

Documented elsewhere.

Use Case

Definition

A use case is an identifiable and externally observable behavior of the target system.

In the metamodel, a Use Case is associated with its initiating and supporting Participants. Use Cases are related to other Use Cases, to the Scenario of which they define the execution trace, and to the Steps they aggregate.

Associations

extends (to *) [Use Case]:* A Use Case description may be inserted into, and thus extend, any number of Use Case descriptions. 'Extends' refines the behavior of a Use Case by inserting alternative or additional behavior, based on conditions that require special handling. A Use Case can be extended by any number of other Use Cases.

uses (to *) [Use Case]:* A Use Case can use any number of other Use Cases. The 'uses' association is employed when two or more Use Cases have common behavior. The common part is factored out and described only once. A Use Case can be used by any number of other Use Cases.

aggregates (1.. to 1..*) [Step]:* A Use Case aggregates at least one Step. A Step is aggregated into at least one Use Case.

[Use Case] is initiated by (to 1) [«External» Resource Container as initiating participant]:* An «External» Resource Container can initiate any number of Use Cases and a Use Case is initiated by a single «External» Resource Container.

[Use Case] is supported by (to *) [Participant as supporting participant]:* A Participant can support any number of Use Cases and a Use Case is supported by any number of supporting Participants.

Constraint: A Participant is involved with at least one Use Case (either as initiating participant or as supporting participant).

Attributes

purpose

entry condition

exit condition

outcome

variation

alternative

User (as synonym of «External Physical» System)

Definition

A user describes an external physical system which interacts with the target system.

In the metamodel, a User is a synonym of an «External Physical» System. A User will exhibit the properties of the concept it represents, and consequently behaves as an «External» Node (e.g. have an «Access Mechanism» Connection) when we emphasize the operational viewpoint or as an «External» Resource Container (e.g. support Interfaces) when we emphasize the functional viewpoint.

Zone

Definition

A zone aggregates a number of model elements for which a common set of values for a particular kind of requirement is defined.

In the metamodel, a Zone is related to its Boundary, to the Model Elements it aggregates and the Requirement on the basis of which the Zone has been defined.

A model element can only appear in a single zone for any given kind of requirement, given that a particular kind of requirements effectively creates partitions, called zones, for the different relevant ranges of values, e.g. in a security context, model elements will belong to exactly one partition e.g. to either a trusted, a DMZ or non-trusted zone. However, zones can be defined for multiple kinds of requirements, and consequently model elements can appear in multiple zones, each associated with a specific set of values for the different kinds of requirements.

Associations

aggregates (to 1..*) [Model Element]:* A Zone refers to at least one Model Element. A Model Element can appear in any number of Zones.

6.5 The System Functional viewpoint supported by the System Functional package

Scope

The scope of the System Functional package involves those concepts, which support the functional viewpoint of the architecture of a system. This viewpoint focuses on the structural elements from which the system is built and their (structural and behavioral) relationships.

Import Relationships

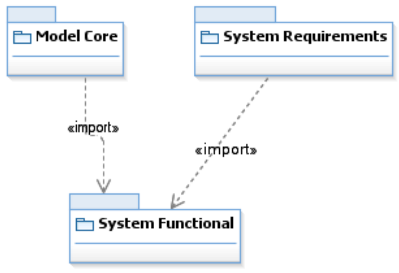


Figure 22: Import relationships of the System Functional package

Overview Diagram

The following diagrams show the relationships between model elements as these could occur in a valid system model, i.e. the cardinalities of the associations reflect what valid associations exist between model elements in a complete system model and do not reflect the fact that during the actual engineering of the system, certain required associations between model elements may not yet have been decided or documented.

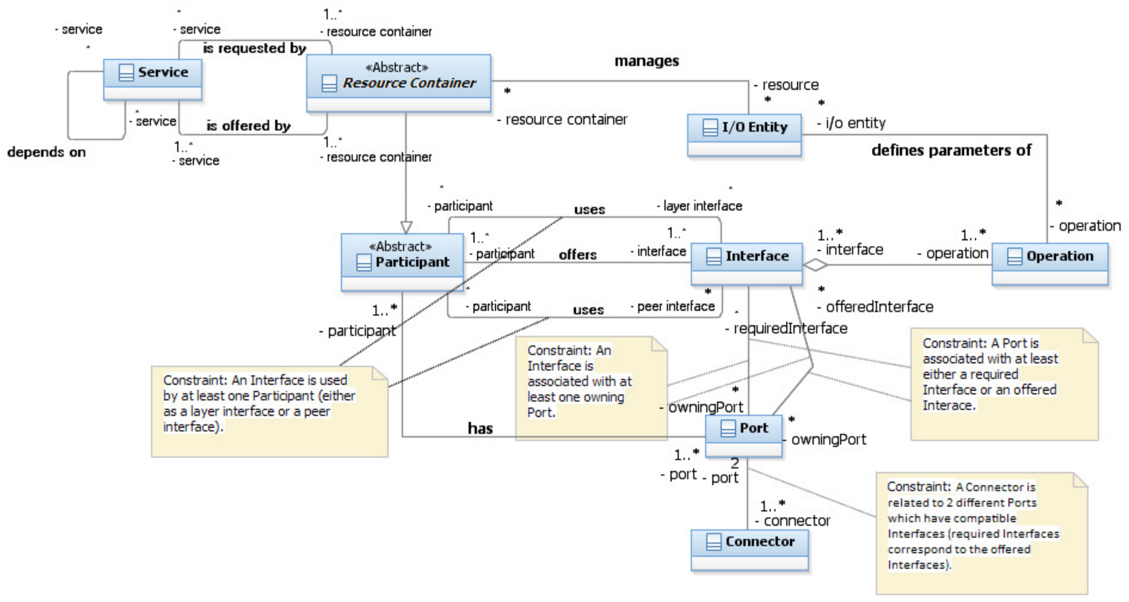

Service
«Abstract» Resource Container
I/O Entity
«Abstract» Participant
Interface
Operation
Port
Connector
is requested by
is offered by
depends on
manages
defines parameters of
uses
offers
has
Constraint: An Interface is used by at least one Participant (either as a layer interface or a peer interface).
Constraint: An Interface is associated with at least one owning Port.
Constraint: A Port is associated with at least either a required Interface or an offered Interace.
Constraint: A Connector is related to 2 different Ports which have compatible Interfaces (required Interfaces correspond to the offered Interfaces).

Figure 23: Concepts/Relationships of the System Functional viewpoint – supporting structural relationships

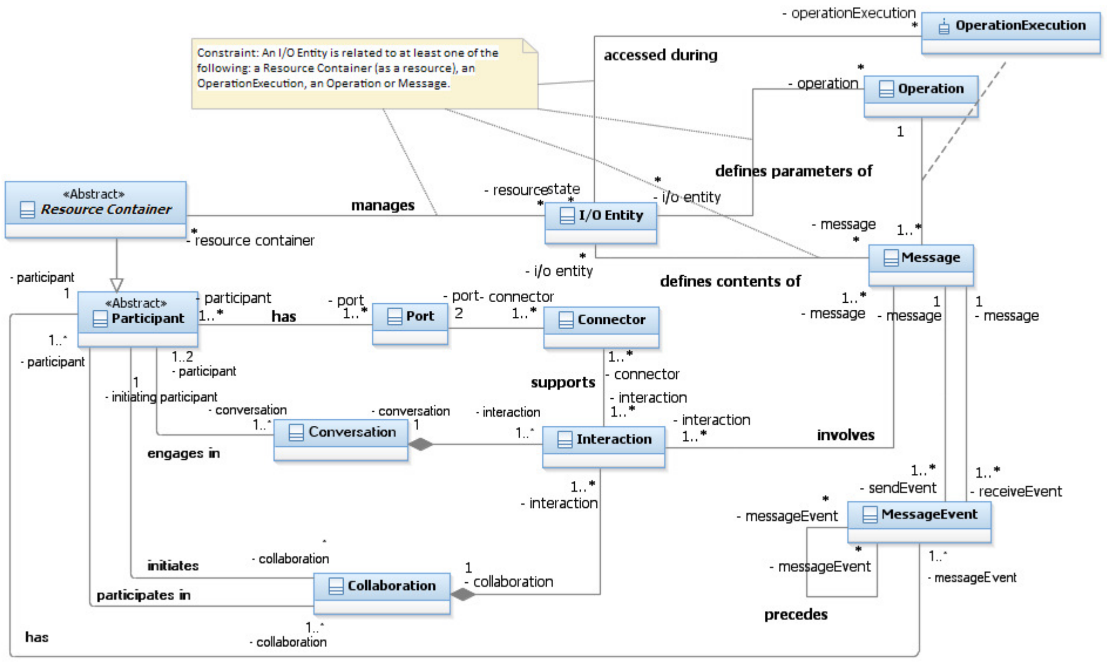


Figure 24: Concepts/Relationships of the System Functional viewpoint – supporting behavioral relationships

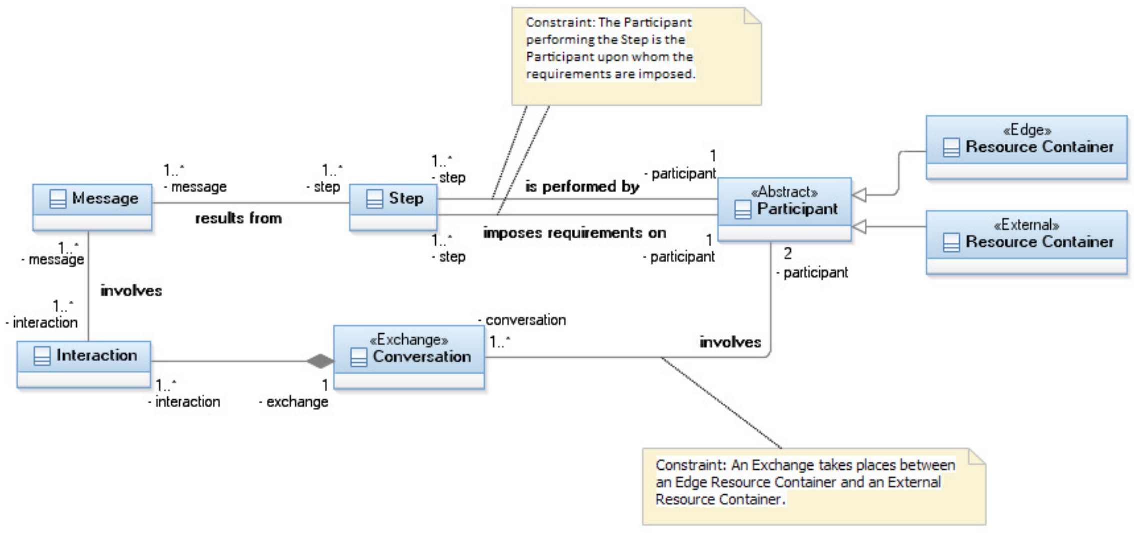


Figure 25: Concepts/Relationships of the System Functional viewpoint – supporting behavioral relationships across the system boundary

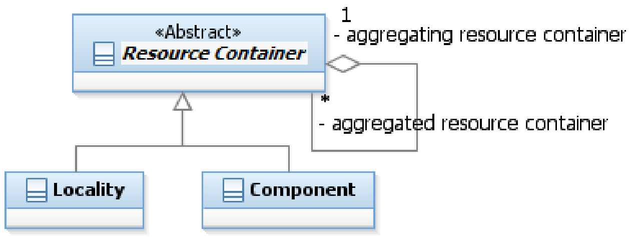

«Abstract» Resource Container
1 - aggregating resource container
* - aggregated resource container
Locality
Component

Figure 26: The notion of a Resource Container

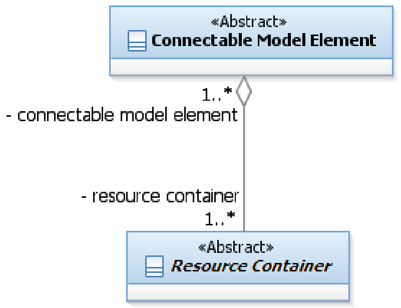

«Abstract» Connectable Model Element
1..* - connectable model element
- resource container 1..*
«Abstract» Resource Container

Figure 27: The notion of Connectable Model Elements extended for the System Functional viewpoint

Concepts

Collaboration

Definition

A collaboration captures the exchange of messages between participants in the context of a particular scenario.

In the metamodel, a Collaboration consists of the Interactions that occur between the Participants that participate in it. Via the MessageEvent's 'precedes'-relationship, a partial order is built among all the

MessageEvents associated with a particular Participant that are generated either during the message send or receive. Via this relationship, the sequence of Messages exchanged during an Interaction can be derived. A Collaboration can then be constructed from these Interactions by appropriately interleaving the Messages on the basis of the partial ordering established at the level of each Participant.

Associations

composed of (1 to 1..) [Interaction]:* Any Collaboration consists of at least one Interaction. An Interaction always occurs in the context of a single Collaboration.

Component

Definition

A component is a modular unit of functionality, which makes this functionality available through an interface.

In the metamodel, a Component inherits from a Resource Container. A Component is a Resource Container to which functionality has been allocated on the basis of considerations of consistency, coherency and loose coupling.

Model Core::Connectable Model Element

Definition

Connectable model elements can aggregate resource containers. Via this aggregation, connectable model elements become related to the interfaces, ports, etc.. associated with the aggregated resource containers.

In the metamodel, a Connectable Model Element aggregates Resource Containers.

Association

aggregates (1.. to 1..*) [Resource Container]:* A Connectable Model Element aggregates at least one Resource Container, while a Resource Container is aggregated at least by one Connectable Model Element (this includes at least the System itself, which is a Connectable Model Element).

Connector

Definition

A connector enables the exchange of messages during interactions between resource containers. The actual messages that can flow across the connector are defined via the required and offered interfaces associated with the ports.

In the metamodel, a Connector is related to the Interactions it supports, and the Ports it connects to.

Associations

supports (1.. to 1..*) [Interaction]:* A Connector supports at least one Interaction. An Interaction is supported by at least one Connector.

Conversation

Definition

A conversation refers to all the interactions taking place between 2 participants across all their collaborations (i.e. across all contexts within which they collaborate).

In the metamodel, a Conversation is related to the Participants between whom it takes place and the Interactions out of which it is composed.

Associations

composes (1 to 1..) [Interaction]:* A Conversation consists out of at least one Interaction. An Interaction is part of a Conversation.

Related Concepts

Stereotype:

«Exchange» Conversation

«Exchange» Conversation

Definition

An exchange refers to the conversation which takes place across the systems boundary between an external and an edge resource container.

In the metamodel, an Exchange is a stereotype of Conversation, and is related to the «Edge» Resource Container and the «External» Resource Container between which it takes place.

Associations

involves (1.. to 2) [«Abstract» Participant]:* An «Exchange» Conversation is related to 2 Participants, an «Edge» Resource Container and an «External» Resource Container. An «Edge» Resource Container and an «External» Resource Container are related to at least one «Exchange» Conversation.

Constraint: An «Exchange» Conversation takes places between an «Edge» Resource Container and an «External» Resource Container.

I/O Entity

Definition

An input/output (I/O) entity is anything exchanged either internally, i.e. between parts of a system, or externally, i.e. between parts of the system and its environment. I/O entities can refer to information or physical items such as air, fuel and tickets.

In the metamodel, an I/O Entity relates to the Resource Containers which manage it as a resource, the Operations which rely on it for the definition of their associated parameters, the Messages of which it defines the contents, and the OperationExecutions which access it.

Associations

[I/O Entity as state] accessed during (to *) [OperationExecution]:* an I/O Entity is accessed during the execution of any number of Operations. An OperationExecution can access any number of I/O Entities. Note that this access can imply creation, consumption, modification, or use without modification.

defines parameters of (to *) [Operation]:* Any number of I/O Entities are involved in defining the parameters of an Operation. An I/O Entity defines the parameters of any number of Operations.

defines contents of (to *) [Message]:* Any number of I/O Entities can be involved in defining the contents of a Message. The contents of any given Message can be defined by any number of I/O Entities.

Constraint: An I/O Entity is related to at least one of the following: a Resource Container (as a resource), an OperationExecution, an Operation or a Message.

Interaction

Definition

An interaction refers to the messages exchanged between one or two participants in the context of a collaboration.

In the metamodel, an Interaction is related to the one or two Participants involved in it, the Collaboration, which defines its context and the Conversation of which it is part. Furthermore, it is related to the Messages exchanged during its occurrence and the Connector supporting the exchange of these Messages.

A collaboration involves a number of participants. During the collaboration, participants will invoke operations on each other via messages. The collection of these messages and the sequence of their corresponding message send/receive events will be specific to the given collaboration.

Associations

involves (1.. to 1..*) [Message]:* An Interaction involves the exchange of at least one Message. Any given Message is involved in at least one Interaction.

Interface

Definition

An interface specifies a set of operations offered by participants.

In the metamodel, Interfaces are used and offered by Participants. They consist of one or more Operations and can be used to define Ports.

Interfaces can be used to capture the role a participant plays in a collaboration.

Associations

aggregates (1.. to 1..*) [Operation]:* An Interface specifies at least one Operation. Any given Operation is specified by at least one Interface.

[Interface as requiredInterface] associated with (to *) [Port as owningPort]:* An interface can be associated with any number of owning Ports. A Port is associated with any number of required Interfaces.

[Interface as offeredInterface] associated with (to *) [Port as owningPort]:* An interface can be associated with any number of owning Ports. A Port is associated with any number of offered Interfaces.

<u>Constraint:</u> A Port is associated with at least one Interface, either a required or offered interface.

<u>Constraint:</u> An Interface is associated with at least one owning Port.

Attributes

role definition

responsibility

Locality

Definition

A locality is a modular unit of functionality, reflecting that this functionality will be jointly distributed without indicating an exact geographic location. It makes this functionality available through interfaces.

In the metamodel, a Locality is a sub type of Resource Container with the distinguishing feature that functionality has been allocated to it on the basis of its joint distribution.

Message

Definition

A message defines a single communication between the participants of an interaction. Such a communication can be further characterized by indicating whether the communication is synchronous, immediate, reliable, etc.

In the metamodel, Message is related to the Interaction during which it is sent and to the triggered Operation. It is also related to the MessageEvents defining the moment at which it is either sent or received.

Associations

associated with (1 to 1..) [MessageEvent as receiveEvent]*: At least one ‘receive’ MessageEvent is associated with a Message. A MessageEvent is associated with the receipt of exactly one Message.

associated with (1 to 1..) [MessageEvent as sendEvent]*: At least one ‘send’ MessageEvent is associated with a Message. A MessageEvent is associated with the sending of exactly one Message.

MessageEvent

Definition

A message event indicates either the sending or receiving of a message by a participant.

In the metamodel, a MessageEvent is related to the Participant sending or receiving the Message, and to the actual Message. The ‘precedes’ relationship between MessageEvents creates a partial order between the MessageEvents associated with a particular Participant. This partial order defines the appropriate interleaving of the sending and receiving of Messages by this Participant.

Associations

precedes (to *) [MessageEvent]:* A MessageEvent can precede any number of MessageEvents. Any given MessageEvent can be preceded by any number of MessageEvents.

OperationExecution

Definition

An operation execution represents the work undertaken by a participant on the receipt of a message.

In the metamodel, the OperationExecution is dependent both on the Message and on the Operation triggered by the Message. The OperationExecution is also related to the I/O Entity, representing the state of the Resource Container, it affects.

Attributes

resource consumption: the amount of resources consumed during the OperationExecution

Operation

Definition

An operation defines the message which can trigger a specific behavior of a participant.

In the metamodel, an Operation is grouped into one or more Interfaces and is related to the Messages that trigger it. The parameters of an Operation are defined via the associated I/O Entities.

Associations

associated with (1 to 1..) [Message]:* An Operation is triggered by at least one Message. A Message triggers a single Operation.

Attributes

signature: the actual input, output and return I/O Entity types (including any possible errors or exceptions)

precondition: logic expression specifying the valid state of the System (and operation input parameters) prior to the Operation

postcondition: logic expression specifying the state of the System (and operation output parameters) after the Operation.

Participant

Definition

A participant is model element able to interact with other participants via a structured exchange of messages.

In the metamodel, a Participant is related to the Interfaces it uses and offers and to Interactions, composed in Collaborations and Conversations. It is also associated with the MessageEvents indicating that a Participant sends or receives Messages in a particular sequence during an Interaction. Interfaces offered and required within the context of a particular Interaction can be organized via Ports which define the points of interaction between a Participant and its environment.

Interfaces can make the role a participant plays in a collaboration more explicit by capturing all operations that are invoked on the participant during this collaboration. Given that a participant will generally participate in more than a single collaboration, it will play more than one role and can present more than one interface to the outside world.

A distinction is also introduced between the interfaces required by a participant based on whether these interfaces are positioned at either the same level of abstraction (i.e. belong to the same layer) – referring to the notion of ‘peer interface’, or at the next lower level of abstraction (i.e. belong to the layer underneath) – referring to the notion of ‘layer interface’.

Associations

uses (to *) [Interface as peer interface]:* A Participant can use any number of Interfaces positioned at the same level of abstraction. Any given Interface can be used by any number of Participants.

uses (to *) [Interface as layer interface]:* A Participant can use any number of Interfaces positioned at the next lower level of abstraction. Any given Interface can be used by any number of Participants.

offers (1.. to 1..*) [Interface]:* A Participant offers at least one Interface by providing an implementation for the Operations specified by that Interface. Any given Interface is offered by at least one Participant.

*initiates (1 to *) [Collaboration]:* A Participant can initiate any number of Collaborations. A Collaboration is initiated by one Participant.

participates in (1.. to 1..*) [Collaboration]:* A Participant participates in at least one Collaboration. A Collaboration involves at least one Participant.

has (1 to 1..) [MessageEvent]:* A Participant has at least one MessageEvent. A MessageEvent is associated with exactly one Participant.

has (1.. to 1..*) [Port]:* A Participant has at least a single Port. A Port is associated with at least one Participant.

<u>Constraint:</u> An Interface is used by at least one Participant (either as a layer interface or a peer interface).

engages in (1..2 to 1..) [Conversation]:* A Participant engages in at least one Conversation. A Conversation is engaged in by one or two Participants.

Port

Definition

A port defines the messages, via its associated interfaces and their operations, which a participant can exchange with another participant during an interaction.

In the metamodel, a Port is associated with Participants and their Interfaces. It is also related to the Connectors that connect it to other Ports.

Associations

associated with (2 to 1..) [Connector]:* A Port is connected to at least one Connector. A Connector connects 2 Ports with one another.

<u>Constraint:</u> A Connector is related to 2 different Ports which have compatible Interfaces (required Interfaces correspond to the offered Interfaces).

Resource Container

Definition

A Resource Container is a collection of resources which enable it to deliver a certain function.

In the metamodel, a Resource Container is an abstract sub type of Participant. Resource Containers can manage I/O Entities as a resource. In the case of a single Resource Container managing the I/O Entity, it can be considered that the Resource Container encapsulates that I/O Entity. Resource Containers can aggregate other Resource Containers.

When aggregating resource containers (referring to the ‘aggregates’ relationship), any of the following relationships between the interface(s) of the aggregating and those of the aggregated resource containers can be valid. The interface of the aggregating resource container is:

- The union (sum) of the interfaces of the aggregated resource containers
- A subset of the interfaces of the aggregated resource containers
- A superset of the interfaces of the aggregated resource containers
- Completely different to the interfaces of the aggregated resource containers

In this situation, the behavior of the aggregating resource container can be:

- The same as the behavior of the individual aggregated resource containers, this should be interpreted as follows: the aggregating resource container acts as a pure façade and only delegates to the appropriate aggregated resource container.
- Richer than the behavior of the aggregated resource containers, this should be interpreted as follows: the aggregating resource container has its own functionality besides that associated with the aggregated resource containers.

Associations

manages (to *) [I/O Entity as a resource]:* A Resource Container can manage any number of I/O Entities as a resource. Any given I/O Entity can be managed as a resource by any number of Resource Containers.

[Resource Container as aggregating resource container] aggregates (1.. to *) [Resource Container as* aggregated *resource container]:* A Resource Container can aggregate other Resource Containers. This relationship implies that the aggregating Resource Container can use the Interfaces offered by the aggregated Resource Containers. The Resource Containers are aggregated into a single aggregating Resource Container, i.e. the Interfaces offered by the Resource Containers will only be available to the aggregating Resource Container.

Attributes

responsibility

Related Concepts

Stereotypes:

«Edge» Resource Container

«External» Resource Container

«Edge» Resource Container

Definition

The edge resource container is the resource container with which an external system directly interacts.

In the metamodel, an «Edge» Resource Container is a stereotype of Resource Container.

«External» Resource Container

Definition

The external resource container is the resource container with which the target system directly interacts.

In the metamodel, an «External» Resource Container is a stereotype of Resource Container.

Service

Definition

A Service refers to the delivery of something of value to a requestor by a supplier according to a previously arranged agreement.

In the metamodel, a Service is related to the Resource Container requesting/offering the Service. A Service can be dependent on other Services.

Associations

depends on (to *) [Service]:* A Service can depend on any number of Services.

is requested by (to 1..*) [Resource Container]:* A Service is requested by at least one Resource Container. A Resource Container requests any number of Services.

is offered by (1.. to 1..*) [Resource Container]:* A Service is offered by at least one Resource Container. A Resource Container offers at least one Service.

System Requirements::Step

Definition

Documented elsewhere.

6.6 The System Operational viewpoint supported by the System Operational package

Scope

The scope of the System Operational package involves those concepts, which support the operational viewpoint of the architecture of a system. This viewpoint focuses on how the target system is built from its structural elements and integrates into its environment.

Import Relationships

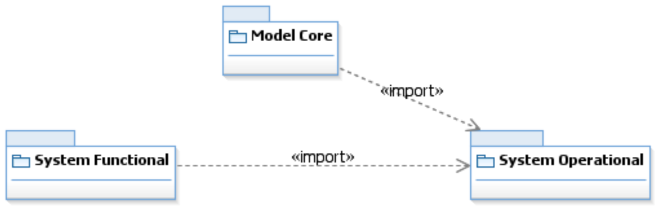


Figure 28: Import relationships of the System Operational package

Overview Diagram

The following diagrams show the relationships between model elements as these could occur in a valid system model, i.e. the cardinalities of the associations reflect what valid associations exist between model elements in a complete system model and do not reflect the fact that during the actual engineering of the system, certain required associations between model elements may not yet have been decided or documented.

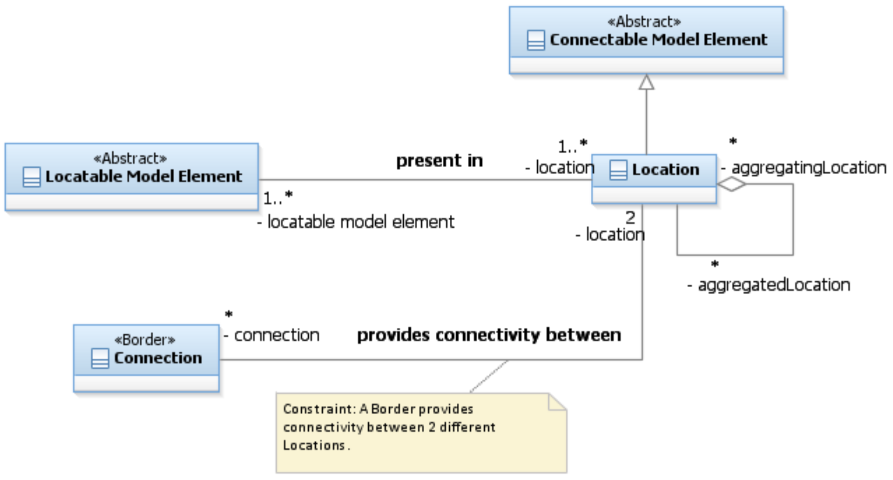


Figure 29: The notion of spatial organization

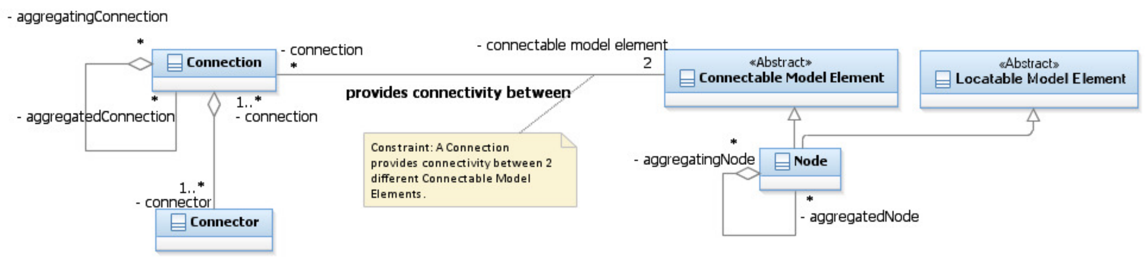


Figure 30: The notion of Connectable Model Elements for the System Operational viewpoint

Concepts

Connection

Definition

A connection delivers the required connectivity between connectable model elements.

In the metamodel, a Connection provides the connectivity between 2 Connectable Model Elements, and can aggregate other Connections and Connectors.

The interactions between resource containers are supported via connectors. Aggregating these resource containers into different connectable model elements, creates an aggregate requirement for the support of these interactions, which is captured via the aggregation of connectors into connections.

Associations

provides connectivity between (to 2) [Connectable Model Element]:* A Connection provides the required connectivity between two Connectable Model Elements. A Connectable Model Element can have any number of Connections.

[Connection as aggregatingConnection] aggregates (to *) [Connection as aggregatedConnection]:* A Connection can be decomposed into any number of Connections, while a Connection can be aggregated into any number of Connections.

aggregates (1.. to 1..*) [Connector]:* A Connection aggregates at least one Connector, while a Connector is aggregated by at least one Connection.

Attributes

connection type: mechanical linkage, pipe, LAN (and type), WAN (and type), dial-up line, etc.

transmission delay

bandwidth: volume, energy, or force that can be transmitted

cost: initial and annual costs

redundancy

Related Concepts

Stereotypes:

«Border» Connection

«Access Mechanism» Connection

«Border» Connection

Definition

A border is the connection between two locations.

In the metamodel, a Border is a stereotype of Connection and is associated with exactly two Locations.

Associations

provides connectivity between (to 2) [Location]:* A «Border» Connection is the connection between 2 Locations. A Location can have any number of «Border» Connections with other Locations.

Constraint: A «Border» Connection provides connectivity between 2 different Locations.

Connector

Definition

Documented elsewhere.

Location

Definition

A location represents a geographical area or position.

In the metamodel, Locations define the places where Locatable Model Elements can be found. These Locatable Model Elements include «Target System» Systems and Nodes. Locations can be further decomposed into other Locations. Locations have Borders with other Locations.

Associations

[Location as aggregatingLocation] aggregates (to *) [Location as aggregatedLocation]*: A Location can be decomposed into any number of Locations. A Location can be aggregated into any number of Locations.

Note: A Location is considered to be aggregated in multiple Locations when the same representative Location Instance is present in multiple Locations.

Node

Definition

A node is a collection of resource containers in a location.

In the metamodel, a Node inherits both from the Connectable Model Element and Locatable Model Element abstract types.

Associations

[Node as aggregating Node] aggregates (to *) [Node as aggregatedNode]:* A Node aggregates any number of other Nodes. Nodes can be aggregated into any number of Nodes.

present in (1.. to 1..*) [Location]:* A Node is present in at least one Location. A Location contains at least one Node.

Note: A Node is considered to be in multiple Locations when the same representative Node Instance is present in multiple Locations.

Related Concepts

Stereotypes:

«Edge» Node

«External» Node

«Edge» Node

Definition

An external node, part of an external system, is the node to which an edge node, part of a target system, is connected. Such a node indicates from where the external system will connect to the target system.

In the metamodel, an «External» Node is a stereotype of Node.

«External» Node

Definition

An edge node, aggregated into a target system, is the node to which an external node, part of an external system, is connected. Such a node indicates from where in the target system, the connection to the external node is made.

In the metamodel, an «Edge» Node is a stereotype of Node.

6.7 The System Validation viewpoint supported by the System Validation package

Scope

The scope of the System Validation package contains those concepts related to assessing whether a system will deliver its intended functionality with the expected quality of service.

Import Relationships

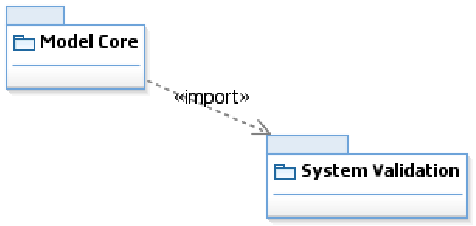


Figure 31: Import relationships of the System Validation package

Overview Diagram

The concepts and relationships of the System Validation package are described in the following diagram:

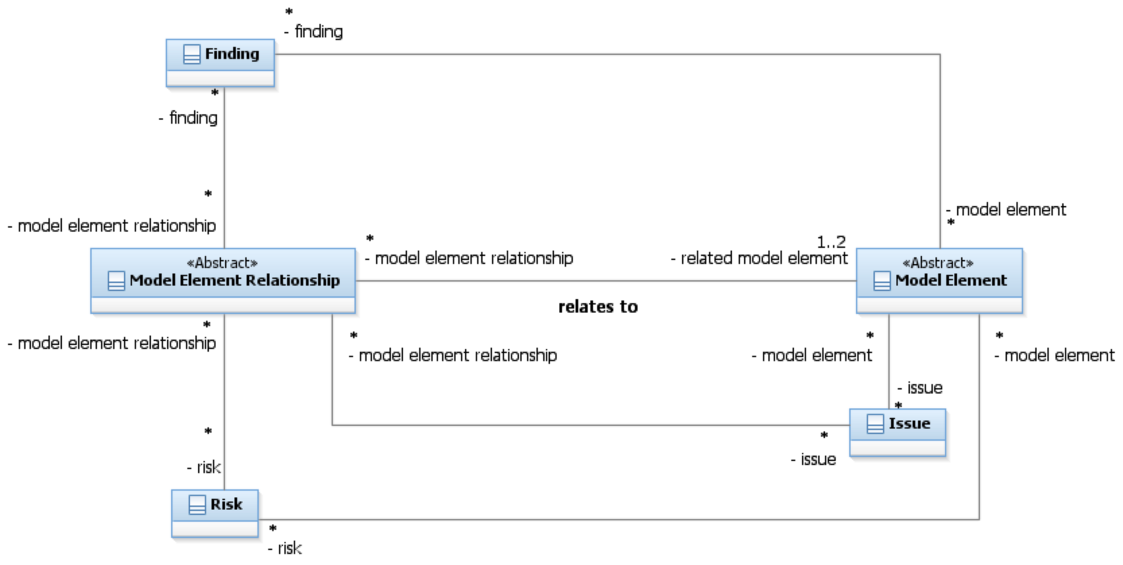

Finding
* - finding
* - finding
* - model element relationship
«Abstract» Model Element Relationship
* - model element relationship
1..2 - related model element
relates to
«Abstract» Model Element
- model element *
* - model element relationship
* - model element relationship
* - model element
* - model element
- issue *
Issue
* - issue
* - risk
Risk
* - risk

Figure 32: Concepts/Relationships of the System Validation viewpoint

Concepts

Finding

Definition

A finding is the result of an investigation ([7]).

In the metamodel, a Finding is related to the Model Elements and Model Element Relationships it concerns.

Associations

associated with (to *) [Model Element]:* A Finding can be associated with any number of Model Elements. A Model Element can be associated with any number of Findings.

associated with (to *) [Model Element Relationship]:* A Finding can be associated with any number of Model Element Relationships. A Model Element Relationship can be associated with any number of Findings.

Issue

Definition

An issue is a generic term for a matter of concern on a project ([7]).

In the metamodel, an Issue is related to the Model Elements and Model Element Relationships it concerns.

Associations

Documented elsewhere.

Model Core::Model Element

Associations

associated with (to *) [Issue]:* A Model Element can be associated with any number of Issues. An Issue can be associated with any number of Model Elements.

Model Core::Model Element Relationship

Associations

associated with (to *) [Issue]:* A Model Element Relationship can be associated with any number of Issues. An Issue can be associated with any number of Model Element Relationships.

associated with (to *) [Risk]:* A Model Element Relationship can be associated with any number of Risks. A Risk can be associated with any number of Model Element Relationships.

Risk

Definition

A risk is a potential event or future situation that may adversely affect the project ([7]).

In the metamodel, an Issue is related to the Model Elements and Model Element Relationships it concerns.

Associations

associated with (to *) [Model Element]:* A Risk can be associated with any number of Model Elements. A Model Element can be associated with any number of Risks.

7. IT System Description Standard (ADS)

7.1 Introduction

The Architecture Description Standard for IT Systems (ADS) defines the concepts used in modeling IT systems.

The concepts defined in SDS for modeling systems provide the basis for modeling IT systems (ADS) and hence remain fully applicable. However, some concepts will be refined where such redefinition allows for a richer set of relationships and meanings. As a consequence, such redefinitions will then only be applicable to the IT system's context. However, the majority of concepts are equally well applicable to systems as to IT system's contexts. So whenever, a concept is used without the 'IT' qualifier, the actual interpretation is inferred from the context in which it is being used, explicit qualification will be used to disambiguate where necessary or when specific relationships are only valid within the IT context.

Throughout the discussion, we will assume that the boundary of the IT system under scrutiny does not change, as we did in section 6.1.

7.2 Package structure

Figure 33 represents the overall package structure of the ADS. Note that the arrows flow from the supplier to the client.

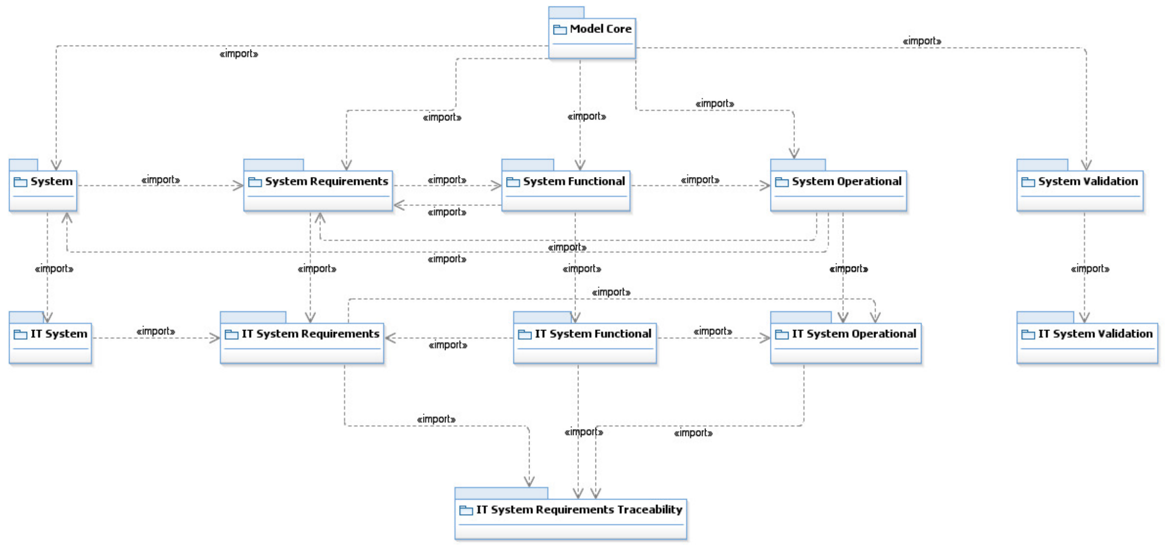


Figure 33: ADS package structure

7.3 The notion of an IT System: the IT System package

Scope

The scope of the IT System package includes those concepts, which are used to model entire IT systems and their IT subsystems.

Import Relationships

The IT System package imports all the concepts of the System package and constrains the concepts to an IT system's context. Where necessary, 'IT system' (and 'IT subsystem') will be used to make the context explicit.

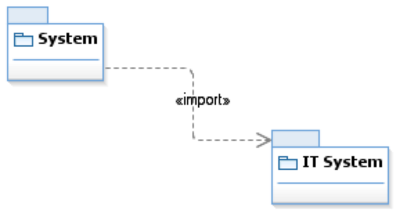


Figure 34: Import relationships of the IT System package

Concepts

No specializations of system level concepts are introduced in this section.

7.4 IT System Requirements viewpoint supported by the IT System Requirements package

Scope

The scope of the IT System Requirements package contains those concepts which are involved in the modeling of IT system requirements. As previously, our context is a single system, implying that we ignore the fact that requirements and related concepts could be applicable to any number of IT systems.

Import Relationships

All concepts defined within the System Requirements package are applicable to the IT System Requirements package.

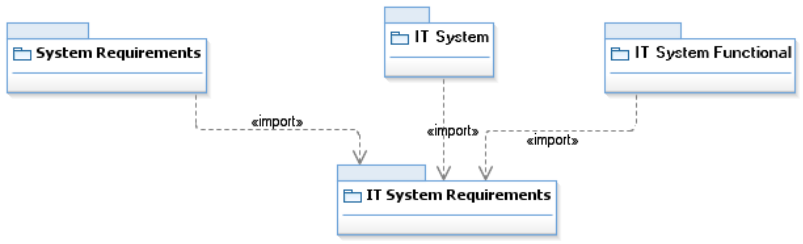


Figure 35: Import relationships of the IT System Requirements Package

To emphasize the specific IT system's context, we have added in Figure 36 the explicitly qualified concepts.

Overview Diagram

The following diagram shows the relationships between model elements as these could occur in a valid system model, i.e. the cardinalities of the associations reflect what valid associations exist between model elements in a complete system model and do not reflect the fact that during the actual engineering of the system, certain required associations between model elements may not yet have been decided or documented.

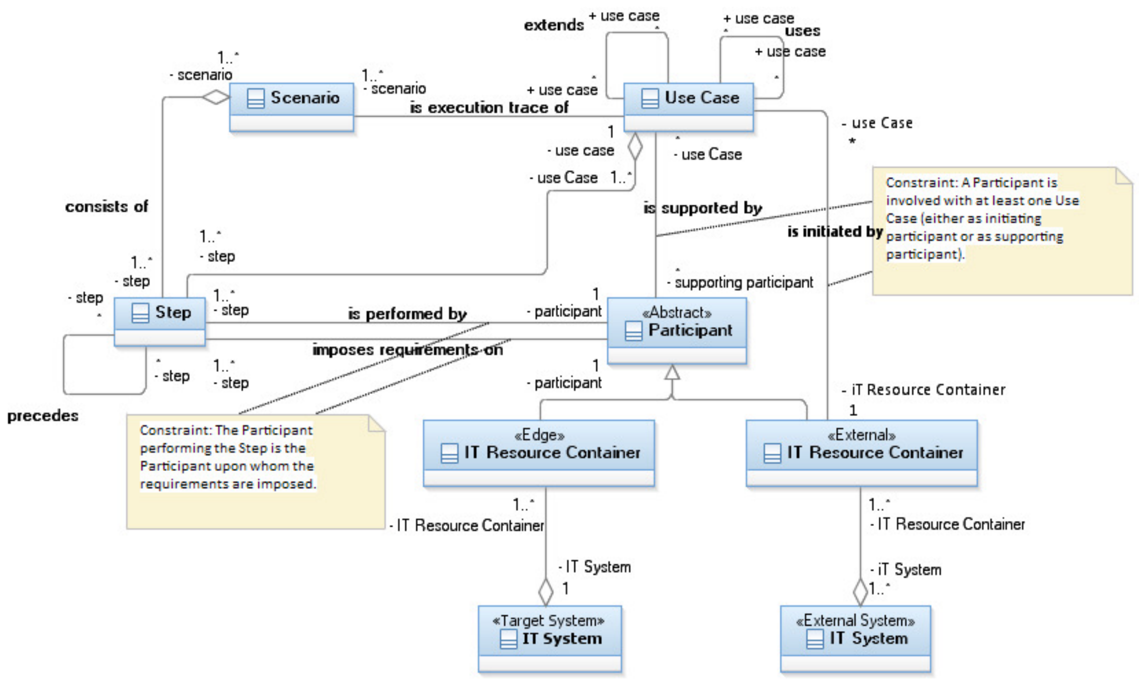


Figure 36: Concepts/Relationships of the IT System Requirements package

Concepts

No specializations of system level concepts are introduced in this section.

7.5 IT System Functional viewpoint supported by the IT System Functional package

Scope

The scope of the IT System Functional package involves those concepts, which support the functional viewpoint of the architecture of an IT system. This viewpoint focuses on the structural elements from which the IT system is built and their (structural and behavioral) relationships.

Import Relationships

The IT System Functional package imports all the concepts of the System Functional package and constrains the following system's concepts to an IT system's context:

- I/O Entity to Data Type
- Resource Container to IT Resource Container
- Locality to IT Locality
- Component to IT Component

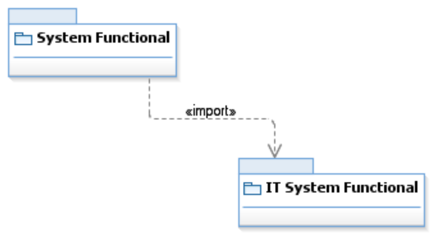


Figure 37: Import relationships of the IT System Functional package

Overview Diagrams

The following diagrams show the relationships between model elements as these could occur in a valid system model, i.e. the cardinalities of the associations reflect what valid associations exist between model elements in a complete system model and do not reflect the fact that during the actual engineering of the system, certain required associations between model elements may not yet have been decided or documented.

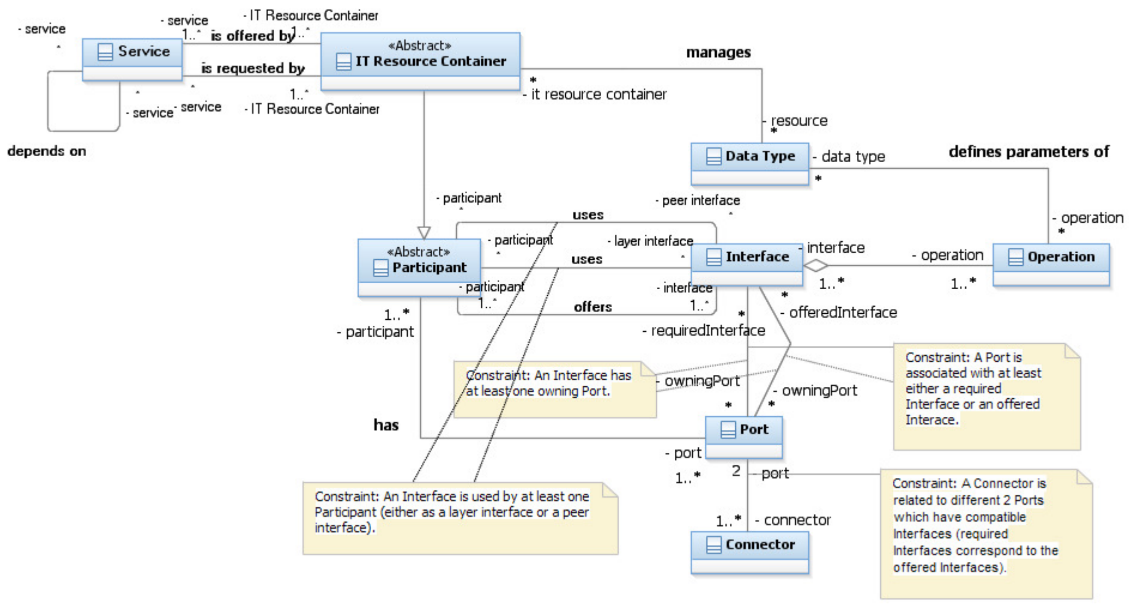

Service
«Abstract» IT Resource Container
is offered by
is requested by
depends on
- service
- IT Resource Container
- it resource container
manages
- resource
Data Type
- data type
defines parameters of
- operation
Operation
«Abstract» Participant
- participant
uses
- peer interface
- layer interface
Interface
- interface
offers
- offeredInterface
- requiredInterface
- owningPort
has
Port
- port
2
- connector
Connector
1..*
Constraint: An Interface has at least one owning Port.
Constraint: A Port is associated with at least either a required Interface or an offered Interace.
Constraint: An Interface is used by at least one Participant (either as a layer interface or a peer interface).
Constraint: A Connector is related to different 2 Ports which have compatible Interfaces (required Interfaces correspond to the offered Interfaces).

Figure 38: Concepts/Relationships of the IT System Functional viewpoint – supporting the structural relationships

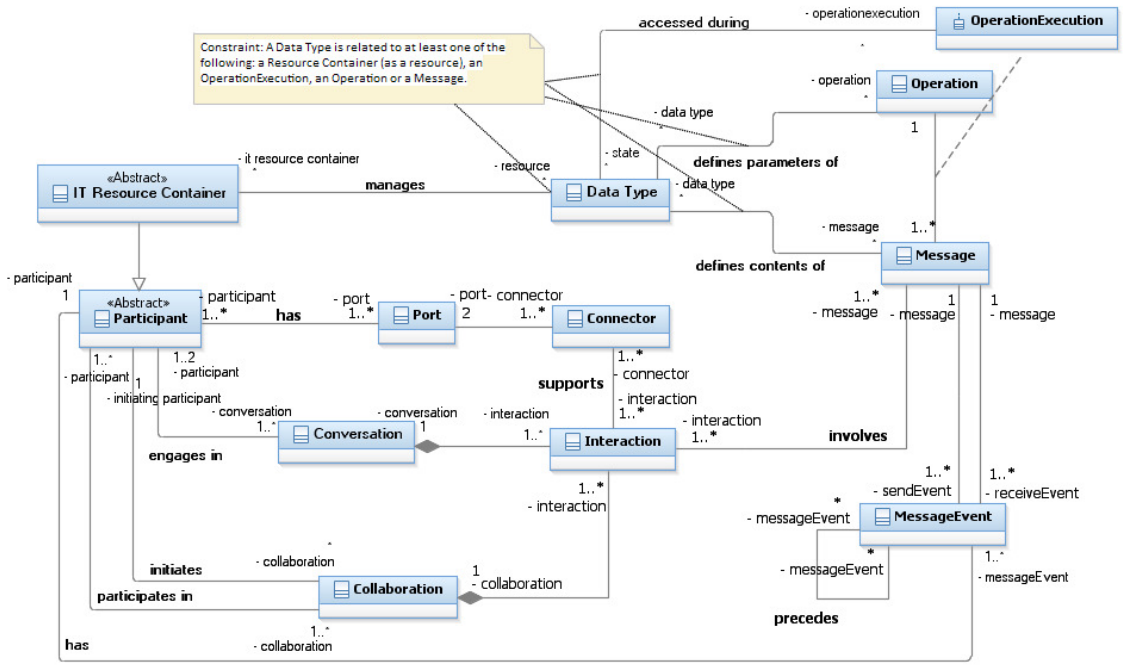


Figure 39: Concepts/Relationships of the IT System Functional viewpoint – supporting the behavioral relationships

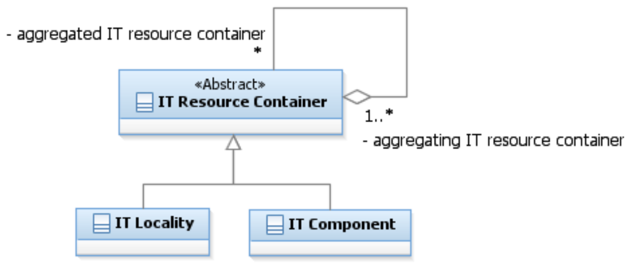

- aggregated IT resource container
«Abstract» IT Resource Container
- aggregating IT resource container
IT Locality
IT Component

Figure 40: The notion of an IT Resource Container

Concepts

Data Type

Definition

A data type is a definition of a named set of attributes.

In the metamodel, Data Types are associated with IT Resource Containers that manage them as a resource. These resources can be accessed during the execution of an Operation. Data Types are also related to the Operations for which they define the parameters and the Messages of which they define the contents.

Associations

[Data Type as state] is accessed during (to *) [OperationExecution]:* a Data Type is accessed during the execution of any number of Operations. An OperationExecution can access any number of Data Types. Note that this access may be in create, read, update or delete mode.

defines parameters of (to *) [Operation]:* A Data Type defines the parameters of any number of Operations. An Operation can have any number Data Types defining its parameters.

defines contents of (to *) [Message]:* A Data Type can define the contents of any number of Messages. The contents of a Message can be defined by any number of Data Types.

Constraint: A Data Type is related to at least one of any of the following: an IT Resource Container (as a resource), an OperationExecution, an Operation or a Message.

Attributes

size

format

IT Component

Definition

An IT component is a modular unit of IT functionality, which makes this functionality available through an interface.

In the metamodel, an IT Component inherits from an IT Resource Container and constrains the notion of a Component to an IT specific context. An IT Component is an IT Resource Container to which functionality has been allocated on the basis of considerations of consistency, coherency and loose coupling.

IT Locality

Definition

An IT locality is a modular unit of IT functionality, reflecting that its functionality will be jointly distributed without indicating an exact geographic location. It makes this functionality available through interfaces.

In the metamodel, an IT Locality inherits from an IT Resource Container whereby functionality has been allocated on the basis of its joint distribution. An IT Locality constrains the notion of a Locality to an IT specific context.

IT Resource Container

Definition

An IT Resource Container is a collection of IT resources which enable it to deliver a certain function. Examples of resources are data, functionality, …

In the metamodel, an IT Resource Container is a sub type of Participant. IT Resource Containers can manage Data Types as a resource. In the case of a single IT Resource Container managing the Data Type, it can be considered that the IT Resource Container encapsulates that Data Type. An IT Resource Container can aggregate other IT Resource Containers.

Associations

manages (to *) [Data Type as a resource]:* An IT Resource Container can manage any number of Data types as a resource. Any given Data Type can be managed as a resource by any number of IT Resource Containers.

Note: the other associations are as defined for the Resource Container.

7.6 IT System Operational viewpoint supported by the IT System Operational package

Scope

The scope of the IT System Operational package involves those concepts, which support the operational viewpoint of the architecture of an IT System. This aspect focuses on how the target IT system is built from its structural elements and integrates into its environment.

Import Relationships

The IT System Operational package imports all the concepts of the System Operational package and constrains one system level concept to an IT system specific context:

- Node to IT Node

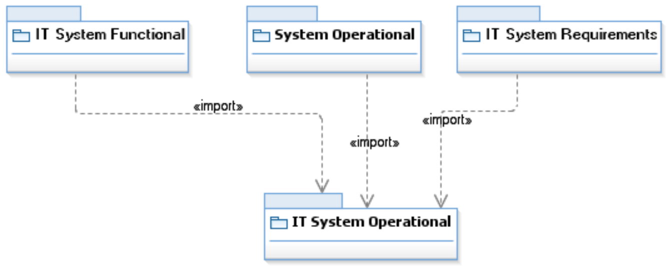


Figure 41: Import relationships of the IT System Operational package

Overview Diagrams

The following diagram shows the relationships between model elements as these could occur in a valid system model, i.e. the cardinalities of the associations reflect what valid associations exist between model elements in a complete system model and do not reflect the fact that during the actual engineering of the system, certain required associations between model elements may not yet have been decided or documented.

Figure 42 emphasizes the important role of the qualities in the definition of the presentation deployment unit type, however, as stated in section 6.4, qualities can be related to any model element and model element relationship, and hence they are also associated to the other deployment unit types.

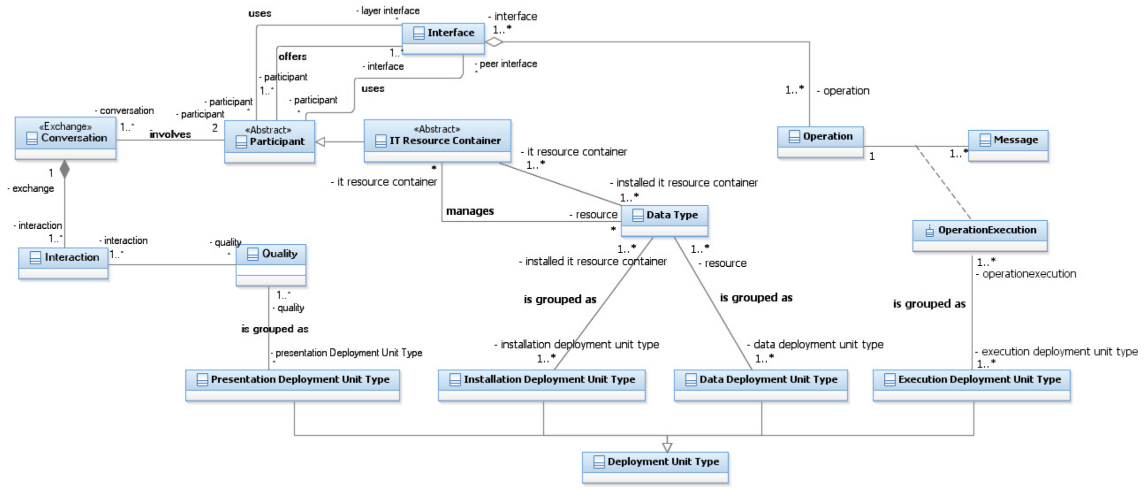
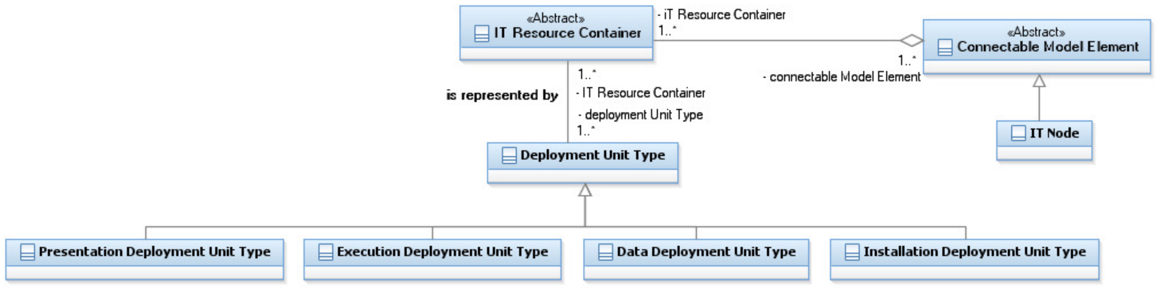


Figure 42: Concepts/Relationships of the IT System Operational package



«Abstract»
IT Resource Container
- iT Resource Container
1..*
«Abstract»
Connectable Model Element
- connectable Model Element
is represented by
- IT Resource Container
- deployment Unit Type
Deployment Unit Type
IT Node
Presentation Deployment Unit Type
Execution Deployment Unit Type
Data Deployment Unit Type
Installation Deployment Unit Type

Figure 43: The notion of placement

Concepts

IT System Functional::Data Type

Associations

[Data Type as resource] is grouped as (1.. to 1..*) [Data Deployment Unit Type]:* a Data Type as resource is grouped in at least one Data Deployment Unit Type. Any given Data Deployment Unit Type groups at least one Data Type as resource.

[Data Type as installed it resource container] is grouped as (1.. to 1..*) [Installation Deployment Unit Type]:* a Data Type, representing an installed IT Resource Container, is grouped in at least one Installation Deployment Unit Type. An Installation Deployment Unit Type groups at least one installed IT Resource Container.

Deployment Unit Type

Definition

A deployment unit type is an abstraction of an IT resource container created to simplify the placement process. Different abstractions can be distinguished (consider for example the sub typing of the deployment unit type), each of which is associated with a characteristic set of requirements, intended to support the joint implementation decisions leading to the selection of S/W and H/W components of the IT system.

In the metamodel, Deployment Unit Type defines the base type of a hierarchy of specialized subtypes: Presentation Deployment Unit Type, Data Deployment Unit Type, Installation Deployment Unit Type and Execution Deployment Unit Type. Deployment Unit Types are associated with the IT Nodes to which they are allocated. Each of the Deployment Unit Types is associated with specific types of Requirements. This relationship is established via the association between any Model Element and Requirement.

Data Deployment Unit Type

Definition

A data deployment unit type is an abstraction of the data types managed by an IT resource container as a resource or in other words, the data aspect of an IT resource container. For placement purposes, the data deployment unit type will be associated with requirements characterizing these data types such as e.g. volatility, size, availability, …

In the metamodel, a Data Deployment Unit Type groups Data Types which are managed as a resource by an IT Resource Container.

The consequence of deploying a data deployment unit type onto an IT node is that the corresponding data types will be stored on that node.

Associations

Documented elsewhere.

Attributes

copy: is this a primary, operational or informational copy

segmentation: is the data segmented, and on what basis

currency: how current is the data

data size: initial, average and maximum values

retention period: for how long does this data need to be kept

System Functional::«Exchange» Conversation

Definition

Documented elsewhere.

Execution Deployment Unit Type

Definition

An execution deployment unit type is an abstraction of an IT resource container emphasizing the execution of this IT resource container, or in other words, the execution aspect of an IT resource container. For placement purposes, an execution deployment unit type will be associated with the requirements characterizing this execution such as e.g. frequency, availability, …

In the metamodel, an Execution Deployment Unit Type groups a number of OperationExecutions.

The consequence of deploying an execution deployment unit type onto an IT node is that the IT resource container, of which the operation executions have been grouped in the execution deployment unit type, will execute on that particular IT node.

Associations

Documented elsewhere.

Installation Deployment Unit Type

Definition

An installation deployment unit type is an abstraction of an IT resource container emphasizing the installed IT resource container, or in other words, the installation aspect of the IT resource container. For placement purposes, an installation deployment unit type will be associated with the requirements characterizing this installation such as e.g. size, frequency of update, …

In the metamodel, an Installation Deployment Unit Type is related to the Data Type used to represent the installed IT Resource Container.

The consequence of deploying an installation deployment unit type onto an IT node is that the IT resource containers associated with this deployment unit type will be installed on that particular IT node.

System Functional::Interaction

Definition

Documented elsewhere.

IT Node

Definition

An IT node is a collection of hardware and software components in a location.

In the metamodel, an IT Node aggregates IT Resource Containers, and given the ‘is represented by’ relationship between these Containers and the Deployment Unit Types, it also aggregates these Deployment Unit Types.

IT System Functional::IT Resource Container

Associations

associated with (1.. to 1..*) [Data Type as installed IT resource container]:* An IT Resource Container has at least one Data Type associated with its installation. Any given installation is associated with at least one IT Resource Container.

is represented by (1.. to 1..*) [Deployment Unit Type]:* An IT Resource Container is represented by at least one Deployment Unit Type. A Deployment Unit Type represents at least one IT Resource Container. Through this relationship, Deployment Unit Type assumes the behavior of the IT Resource Container and can be aggregated into an IT Node.

note: Other associations as defined previously.

System Functional::OperationExecution

Associations

is grouped as (1.. to 1..*) [Execution Deployment Unit Type]:* an OperationExecution is grouped in at least one Execution Deployment Unit Type. An Execution Deployment Unit Type groups at least one OperationExecution.

System Functional::Participant

Definition

Documented elsewhere.

System Requirements::Quality

Definition

Documented elsewhere.

Presentation Deployment Unit Type

Definition

A presentation deployment unit type groups the requirements, which apply to the interactions across the system boundary, i.e. the interactions taking place between the external and edge IT resource containers. It is said that the external resource container interacts with the presentation of the edge resource container.

In the metamodel, a Presentation Deployment Unit Type is related to the Qualities characterizing the Interactions, part of the «Exchange» Conversation, between the «External» and «Edge» IT Resource Containers.

The consequence of deploying a presentation deployment unit type onto an IT node is that at a later stage, additional IT resource containers will be identified to provide the external resource container with the required access to the presentation of the edge resource container such that once the joint implementation decisions will have been made, the characteristics of these IT resource containers meet the requirements attached to the access to this presentation.

7.7 The notion of IT System Traceability supported by IT System Requirements Traceability package

Scope

The scope of the IT System Requirements Traceability package involves those concepts and relationships which are required to establish the traceability from the model elements used in the requirements viewpoints of the architecture to those used in the functional and operational viewpoint.

Import Relationships

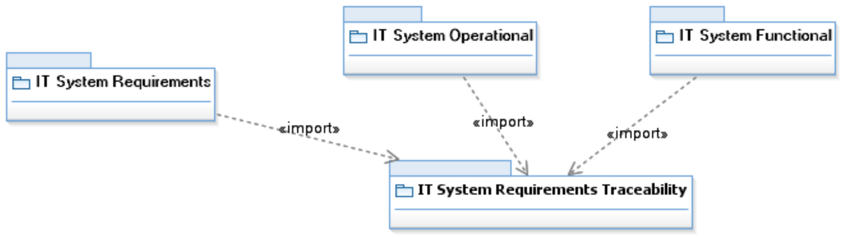


Figure 44: Import relationships of the IT System Requirements Traceability package

Overview Diagram

The following diagram shows the relationships between model elements as these could occur in a valid system model, i.e. the cardinalities of the associations reflect what valid associations exist between model elements in a complete system model and do not reflect the fact that during the actual engineering of the system, certain required associations between model elements may not yet have been decided or documented.

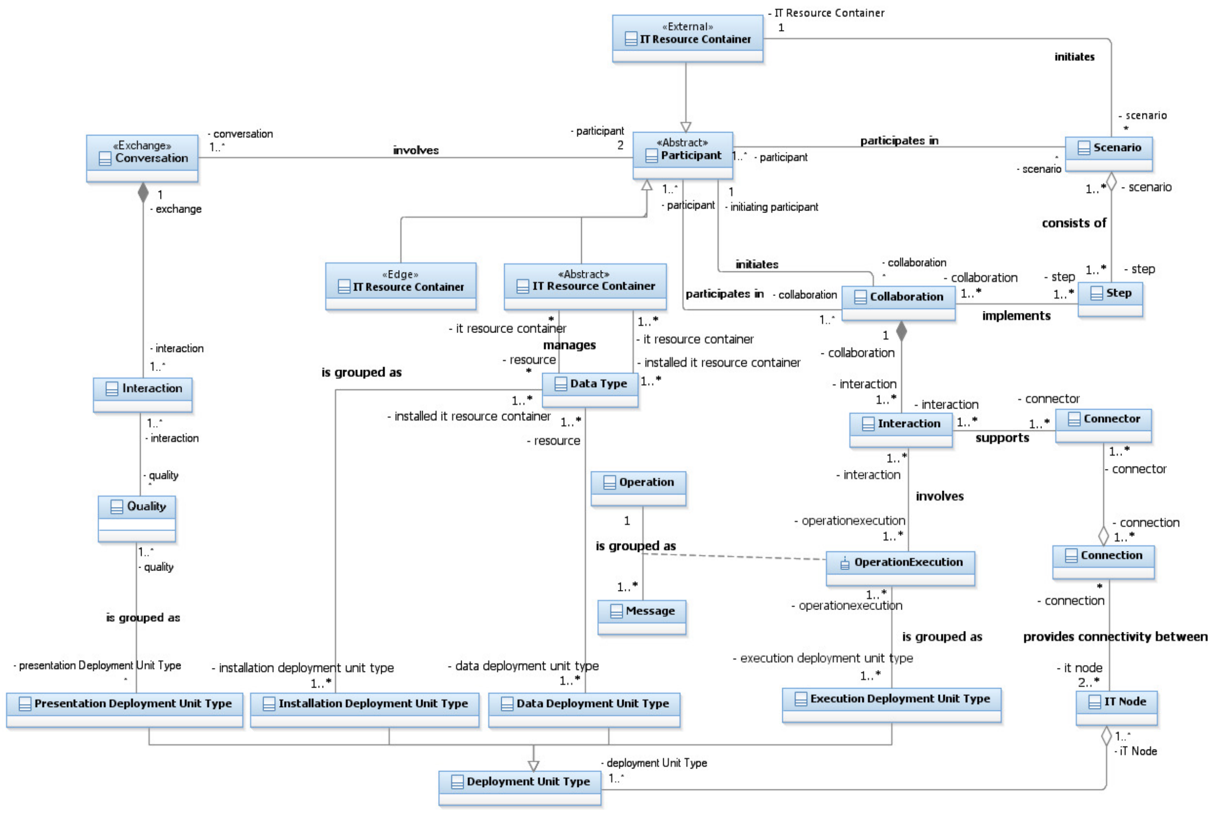


Figure 45: Concepts/Relationships establishing traceability between the concepts used in the requirements viewpoint to those used in the functional/operational viewpoint

The diagram above does not introduce any new concepts or relationships, although a number of derived relationships have been added in order to simplify the overall presentation.

The relationships between the concepts support the flow down of requirements from the scenarios and the associated steps to the expected behavior of the IT resource containers – ensuring the overall expected behavior of the system will be delivered through the collaboration of these IT resource containers. Subsequently the geographical structure of the system is taken into account via the allocation of the deployment units, reflecting the NFR's associated with the installation, execution and data of the IT resource containers to IT nodes. The co-location of these deployment unit types (reflecting the aggregate node level requirements) delivers the overall node specification – ensuring that the behavior will be delivered with the expected quality by the system. The requirements specifically associated with the exchanges between the system and its actors will be allocated to the (edge) IT nodes via the placement of presentation deployment units.

7.8 IT System Validation viewpoint supported by the IT System Validation package

Scope

The scope of the IT System Validation viewpoint involves those concepts which are involved in assessing whether the IT system will deliver its intended functionality with the expected quality of service.

Import Relationships

The IT System Validation package imports all the concepts from the System Validation package.

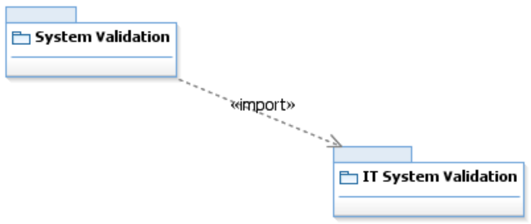


Figure 46: Import relationships of the IT System Validation package

Concepts

No specializations of system level concepts are introduced in this section.

8. References

[1] : IEEE Architecture Working Group, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, IEEE Std 1471-2000, IEEE, 2000.

[2] : OMG. *Unified Modeling Language: Superstructure Version 2.0*, OMG Formal Specification formal/05-07-04.

[3] : System Context Modeling – Ian Charters, white paper published as part of the UMF Internal Content.

[4] : 'ADS Functional Aspects Notation' document available from the System Architecture Description – SDS Community (https://w3-connections.ibm.com/communities/community/sds) – as part of the deliverables of 'Architecture Description Standard R2 (ADS R2)'.

[5] : An introduction to the IBM Views and Viewpoints Framework for IT systems – Denise Cook, Peter Cripps, Philippe Spaas – January 8th 2008 (available from developerWorks on http://www.ibm.com/developerworks/rational/library/08/0108_cooks-cripps-spaas/index.html)

[6] : For an overview of RUP-SE we refer to the following series of articles which have been published in the Rational Edge:

- Rational Unified Process for Systems Engineering by Murray Cantor in August – September – October 2003
- *An overview of the Systems Modeling Language for product and systems development* by L. Balmelli in August – September - October 2006

[7] : WWPMM Glossary (available from Worldwide Project Management Method on http://w3-03.ibm.com/transform/project/pmmethod/Concepts/key_concepts.html)

[8] : 'ADS Operational Model Notation' - document available from the System Architecture Description – SDS Community (https://w3-connections.ibm.com/communities/community/sds) – as part of the deliverables of 'Architecture Description Standard R2 (ADS R2)'.

[9] : 'SDS Overview' presentation available from the System Architecture Description – SDS Community (https://w3-connections.ibm.com/communities/community/sds) – as part of the deliverables of 'Architecture Description Standard R3.1 (SDS R3.1)'.

[10] : 'SDS Glossary' document available from the System Architecture Description – SDS Community (https://w3-connections.ibm.com/communities/community/sds) – as part of the deliverables of 'Architecture Description Standard R3.1 (SDS R3.1)'.

9. Annexes

9.1 Cross-References

9.1.1 Introduction

In order to provide an integrated overview of the way certain modeling constructs have been used throughout this document, we have created an overview of the main inheritance relationships present in the SDS model.

The overview focuses on the inheritance from the following abstract model elements:

- Connectable Model Element
- Non-Connectable Model Element
- Locatable Model Element
- Non-Locatable Model Element

9.1.2 Overview of inheritance relationships

Inherits from	Connectable Model Element	Non-Connectable Model Element	Locatable Model Element	Non-Locatable Model Element
Actor	Cfr. System			
Boundary		♈		♈
Collaboration		♈		♈
Component	Cfr. Resource Container			
Connection		♈		♈
Connector		♈		♈
Constraint	Cfr. Requirement			
Conversation		♈		♈
Data Deployment Unit Type	Cfr. Deployment Unit Type			
Data Type		♈		♈
Deployment Unit Type		♈		♈
Execution Deployment Unit Type	Cfr. Deployment Unit Type			
Finding		♈		♈
Functional Requirement	Cfr. Requirement			
I/O Entity		♈		♈
Installation Deployment Unit Type	Cfr. Deployment Unit Type			
Interaction		♈		♈
Interface		♈		♈
Issue		♈		♈

Inherits from	Connectable Model Element	Non-Connectable Model Element	Locatable Model Element	Non-Locatable Model Element
Locality	Cfr. Resource Container			
Location	♈			♈
Message		♈		♈
MessageEvent		♈		♈
Node	♈		♈	
Non-Functional Requirement	Cfr. Requirement			
Operation		♈		♈
OperationExecution		♈		♈
Participant		♈		♈
Port		♈		♈
Presentation Deployment Unit Type	Cfr. Deployment Unit Type			
Quality	Cfr. Requirement			
Requirement		♈		♈
Resource Container		♈		♈
Risk		♈		♈
Scenario		♈		♈
Service		♈		♈
Step		♈		♈
System	♈		♈	
Use Case		♈		♈
User	Cfr. System			
Worker	Cfr. System			
Zone		♈		♈

Table 1: Overview of inheritance relationships

Notes:

- concepts, refined for the IT system context, have similar inheritance relationships as the corresponding generalized concepts
- stereotyped concepts have the same inheritance relationships as the original concepts
- subtypes have the inheritance relationships of their base types

9.2 Overview of Past Changes

9.2.1 SDS R3: Summary of Changes

R3 introduces a number of significant changes, mainly related to a broadening of the context within which the concepts defined by the description standard can be used.

SDS R3 introduces a specific framework (based on IEEE1471-2000) which defines the relationships between the views, viewpoints and models which can be used for describing systems ([3]). Whereas the actual models are defined implicitly through the artifacts of the UMF, the concepts from which these models are built are defined within the present document. Through this framework, the various contexts within which the SDS concepts can be used and the relationships between these contexts have been made explicit.

An important change relates to the context within which the SDS concepts are intended to be used, since a large number of the SDS concepts are equally applicable within any number of different system contexts, as opposed to being only applicable to an IT systems context. Consequently, the Semantic Specification is organized with the general concepts first, and then only those which require redefinition in more specific contexts are further discussed (i.e. explicitly taking into account a more restrictive context of e.g. a IT system vs. a system). Concepts are only refined when this allows a richer set of relationships and meanings than would be the case in a more generalized context.

This broadening of the context is also the motivation for the name change from ‘Architecture Description Standard (ADS)’ to ‘System Description Standard (SDS)’.

SDS R3 also includes a definition of the main concepts and relationships underpinning the requirements and validation basic viewpoints. This aligns SDS’s scope with the basic viewpoints defined in the UADF.

Starting with this release (R3), all UML2 mapping considerations have been removed from the Semantic Specification, thereby enabling the creation of various UML2 or SysML based profiles. As a consequence, the Model Core package present in ADS R2 has been substantially simplified and now only includes those concepts which are of direct relevance to the SDS R3 concepts.

10. Index of Concepts