



Contract number: 2004 - 026976

EAST-ADL2 UML2 Profile Specification

This specification of the EAST-ADL2 UML profile is based on the EAST-ADL2.0 language defined in the ATESSST project. It presents the UML representation of the language concepts of EAST-ADL2.

The EAST-ADL2 is harmonized with AUTOSAR.

The metamodel and UML profile of EAST-ADL2 is defined in two steps: A domain (automotive) metamodel is defined, capturing only the domain specific needs of the language, without adding the UML2 details. The basic concepts of UML are used for this purpose, such as classes, compositions and associations. Based on the domain metamodel, a UML2 profile for the domain metamodel is defined, specifying stereotypes with properties and constraints.

Comments on the content of this document are welcomed, and should be directed to atesst-coordinator@vtec.volvo.se

Please download the latest available specification and the XMI file ready for use in UML2 tools from the atesst.org website.

Dissemination level

PU

(Restricted to a group specified by the Consortium, including the Commission Services.)

Report status:

Draft

Version	Date	Reason
1.0 prel	2008-01-31	Profile for EAST-ADL2.0 developed in the EU project ATESSST

Specification
2008-01-31

Copyright © 2000-2004, AUDI AG
Copyright © 2000-2004, BMW AG
Copyright © 2006-2008, Carmeq GmbH
Copyright © 2000-2004, CRF
Copyright © 2000-2008, DaimlerChrysler AG
Copyright © 2000-2008, ETAS GmbH
Copyright © 2006-2008, Mecel AB
Copyright © 2006-2008, Mentor Graphics
Copyright © 2000-2004, OPEL GmbH
Copyright © 2000-2004, PSA
Copyright © 2000-2004, Renault
Copyright © 2000-2004, Robert Bosch GmbH
Copyright © 2000-2008, Siemens, Continental
Copyright © 2000-2004, Valeo
Copyright © 2000-2004, Vector
Copyright © 2006-2008, Volvo Car Corporation
Copyright © 2000-2008, Volvo Technology AB
Copyright © 2000-2004, ZF
Copyright © 2000-2008, CEA-LIST
Copyright © 2000-2004, INRIA
Copyright © 2000-2004, LORIA
Copyright © 2000-2004, Paderborn University-C-LAB
Copyright © 2000-2004, Technical University of Darmstadt
Copyright © 2000-2008, Technische Universität Berlin
Copyright © 2006-2008, The Royal Institute of Technology

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

This document describes a language specification developed by an informal partnership of vendors and users, with input from additional reviewers and contributors. This document does not represent a commitment to implement any portion of this specification in any company's products. See the full text of this document for additional disclaimers and acknowledgments. The information contained in this document is subject to change without notice.

LICENSES

THIS SPECIFICATION IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SPECIFICATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

AUTHORS

Contributors version 1.0

Lars-Olof Berntsson, Volvo

Hans Blom, Volvo

DeJiu Chen, The Royal Institute of Technology

Philippe Cuenot, Continental

Ulrich Freund, ETAS

Patrick Frey, ETAS

Sebastien Gerard, CEA

Rolf Johansson, Mentor Graphics

Henrik Lönn, Volvo

Mark-Oliver Reiser, Technische Universität Berlin

David Servat, CEA

Patrick Tessier, CEA

Ramin Tavakoli, Technische Universität Berlin

Martin Törngren, The Royal Institute of Technology

Matthias Weber, Carmeq

The Consortium

Volvo(S)

ETAS (D)

The Royal Institute of Technology (S)

Carmeq GmbH (D)

Daimler (D)

Mentor Graphics (Hu)

Technische Universität Berlin (D)

Volvo Cars (S)

Continental (F)

CEA (F)

Mecel (S)

Version details

Enterprise Architect Domain Model last modified: 2008-01-22 14:38:28, XMI generated: 2008-01-22 16:13:26.

Profile version 1.7.1 (DomainModel_2007-11-09) without Autosar and RIF : 2008-01-11

Table of Contents

Version details	4
Table of Contents	4
Part I Introduction.....	7
1 Language Formalism.....	8
2 Scope	8
3 Normative references	8
Part II Structural Constructs	9
4 EAST-ADL2 extensions for ADLCoreConstructs	9
4.1 «ADLContext» (from ADLCoreConstructs) {abstract}.....	10
4.2 «ADLEntity» (from ADLCoreConstructs) {abstract}	10
4.3 «ADLRelationship» (from ADLCoreConstructs) {abstract}	10
4.4 «ADLTraceableSpecification» (from ADLCoreConstructs) {abstract}.....	11
5 EAST-ADL2 extensions for ADLRelationshipModeling	11
5.1 «ADLDeriveReq» (from ADLRelationshipModeling)	11
5.2 «ADLRealization» (from ADLRelationshipModeling)	12
5.3 «ADLRefine» (from ADLRelationshipModeling).....	13
5.4 «ADLSatisfy» (from ADLRelationshipModeling)	13
5.5 «ADLVerify» (from ADLRelationshipModeling)	14
6 EAST-ADL2 extensions for ADLTypes	14
6.1 «ADLDesignDataType» (from ADLTypes)	14
6.2 ADLDesignDataTypeKind (from ADLTypes)	15
6.3 «ADLDouble» (from ADLTypes).....	16
6.4 «ADLFloat» (from ADLTypes)	16
7 EAST-ADL2 extensions for SystemModel.....	16
7.1 «ADLSystemModel» (from SystemModel)	16
7.2 «AnalysisArchitecture» (from SystemModel)	17
7.3 «DesignArchitecture» (from SystemModel)	17
7.4 «FunctionalDesignArchitecture» (from SystemModel)	18
7.5 «MWAbstraction» (from SystemModel)	18
8 EAST-ADL2 extensions for FunctionModeling	19
8.1 «ADLClientPort» (from FunctionModeling)	19
8.2 «ADLClientServerPort» (from FunctionModeling) {abstract}.....	19
8.3 «ADLConnectorPrototype» (from FunctionModeling).....	20
8.4 «ADLDiscFunctionType» (from FunctionModeling)	20
8.5 «ADLFlowPort» (from FunctionModeling) {abstract}.....	21
8.6 «ADLFunctionType» (from FunctionModeling)	21
8.7 «ADLInFlowPort» (from FunctionModeling)	24
8.8 «ADLInOutFlowPort» (from FunctionModeling).....	25
8.9 «ADLOutFlowPort» (from FunctionModeling).....	25
8.10 «ADLPortGroup» (from FunctionModeling)	26
8.11 «ADLServerPort» (from FunctionModeling)	26
8.12 «ADLStructurePrototype» (from FunctionModeling)	26
8.13 «ADLVariableElement» (from FunctionModeling) {abstract}	27
8.14 «DesignInterface» (from FunctionModeling).....	27
8.15 «FunctionalAnalysisArchitecture» (from FunctionModeling)	28
8.16 «FunctionalDevice» (from FunctionModeling).....	28

8.17	«LocalDeviceManager» (from FunctionModeling)	29
8.18	«Trigger» (from FunctionModeling)	29
9	EAST-ADL2 extensions for HardwareDesignArchitecture	30
9.1	«Actuator» (from HardwareDesignArchitecture)	30
9.2	«ADLHwConnector» (from HardwareDesignArchitecture) {abstract}	30
9.3	«ADLHwElement» (from HardwareDesignArchitecture) {abstract}	31
9.4	«ADLHwElementPrototype» (from HardwareDesignArchitecture)	31
9.5	«ADLHwPort» (from HardwareDesignArchitecture) {abstract}	32
9.6	«CommunicationHwPort» (from HardwareDesignArchitecture)	32
9.7	«ECU» (from HardwareDesignArchitecture)	33
9.8	«HardwareDesignArchitecture» (from HardwareDesignArchitecture)	33
9.9	«IoComponent» (from HardwareDesignArchitecture) {abstract}	33
9.10	«IoHwConnector» (from HardwareDesignArchitecture)	34
9.11	«IoHwPort» (from HardwareDesignArchitecture)	34
9.12	«PowerHwConnector» (from HardwareDesignArchitecture)	34
9.13	«PowerHwPort» (from HardwareDesignArchitecture)	35
9.14	«PowerSupply» (from HardwareDesignArchitecture)	35
9.15	«Sensor» (from HardwareDesignArchitecture)	35
9.16	«SystemBusConnector» (from HardwareDesignArchitecture)	36
10	EAST-ADL2 extensions for EnvironmentModeling	36
10.1	«ADLContFunctionType» (from EnvironmentModeling)	36
10.2	«EnvironmentModel» (from EnvironmentModeling)	37
	Part III Behavioral Constructs	37
11	EAST-ADL2 extensions for Behavior	37
11.1	«ADLAction» (from Behavior)	37
11.2	«ADLActivityParameterNode» (from Behavior)	38
11.3	«ADLBehavior» (from Behavior) {abstract}	38
11.4	«ADLBehaviorMapping» (from Behavior)	38
11.5	«ADLUseCase» (from Behavior)	38
11.6	«ExternalBehavior» (from Behavior)	39
11.7	ExternalBehaviorRepresentationKind (from Behavior)	39
11.8	«NativeBehavior» (from Behavior)	40
12	EAST-ADL2 extensions for ErrorBehavior	41
12.1	«ErrorBehavior» (from ErrorBehavior)	41
12.2	«ErrorModel» (from ErrorBehavior)	41
12.3	«ErrorModelToTarget» (from ErrorBehavior)	42
12.4	«ErrorPropagation» (from ErrorBehavior)	43
12.5	«ErrorToHazard» (from ErrorBehavior)	44
12.6	«Hazard» (from ErrorBehavior)	45
12.7	PropagationDirection (from ErrorBehavior)	45
12.8	«PropagationPort» (from ErrorBehavior)	45
	Part IV Cross-Cutting Constructs	46
13	EAST-ADL2 extensions for Requirements	46
13.1	«ADLRequirement» (from Requirements)	46
13.2	«ADLRequirementContainer» (from Requirements)	47
13.3	«AllocateableElement» (from Requirements) {abstract}	47
13.4	«AllocationConstraint» (from Requirements)	47
13.5	«DelayRequirement» (from Requirements)	48
13.6	«DesignConstraint» (from Requirements)	49
13.7	DesignConstraintKind (from Requirements)	50
13.8	«FunctionalRequirement» (from Requirements)	50
13.9	«PrecedenceConstraint» (from Requirements)	51
13.10	«QualityRequirement» (from Requirements)	51
13.11	QualityRequirementKind (from Requirements)	52
13.12	«RequirementDescription» (from Requirements)	52
13.13	«SafetyRequirement» (from Requirements)	53

13.14	SILLevelKind (from Requirements).....	53
13.15	«TimingRestriction» (from Requirements) {abstract}	53
14	EAST-ADL2 extensions for ArtifactLevelVariationManagement	54
14.1	«CombinationDecision» (from ArtifactLevelVariationManagement).....	54
14.2	«DecisionTree» (from ArtifactLevelVariationManagement)	55
14.3	«DependencyConstraint» (from ArtifactLevelVariationManagement)	55
14.4	«EffectNode» (from ArtifactLevelVariationManagement)	56
14.5	«Resolution» (from ArtifactLevelVariationManagement)	56
14.6	«ReuseMetaInformation» (from ArtifactLevelVariationManagement).....	56
14.7	«VariantLink» (from ArtifactLevelVariationManagement)	57
14.8	«VariationGroup» (from ArtifactLevelVariationManagement) {abstract}	57
14.9	VariationGroupKind (from ArtifactLevelVariationManagement)	59
14.10	«VariationPoint» (from ArtifactLevelVariationManagement)	59
14.11	«VariationPointConfiguration» (from ArtifactLevelVariationManagement)	60
15	EAST-ADL2 extensions for Support.....	60
15.1	«ArchivedEntity» (from Support) {abstract}.....	60
15.2	«Configuration» (from Support).....	61
15.3	«Version» (from Support).....	61
15.4	«VersionArchive» (from Support).....	61
16	EAST-ADL2 extensions for VerificationValidation	62
16.1	«AbstractVVCASE» (from VerificationValidation).....	62
16.2	«AbstractVVProcedure» (from VerificationValidation).....	62
16.3	«ConcreteVVCASE» (from VerificationValidation)	63
16.4	«ConcreteVVProcedure» (from VerificationValidation).....	63
16.5	«VVActualOutcome» (from VerificationValidation)	64
16.6	«VVIntendedOutcome» (from VerificationValidation).....	64
16.7	«VVStimuli» (from VerificationValidation)	65
17	Index	65

Part I Introduction

The purpose of the EAST-ADL2 language is to capture the software and electronics architecture with enough detail to allow modeling for documentation, design, analysis and synthesis. These activities require system descriptions on several abstraction levels, from top level user features down to tasks and communication frames in CPUs and communication links. Moreover, the activities also involve the expression of non-structural aspects of the system under development, e.g. requirements, behavior and validation and verification.

By hosting all aspects in this domain model, the relations between them can be managed more efficiently. The different abstraction levels give a modeling context and a view of systems or functions features with different level of detail, and with a clear separation of concerns.

The EAST-ADL2 does not prescribe a process or methodology. For example, information needed for relevant analysis and synthesis can be captured, but it is not defined how the analysis or synthesis are done. This approach was chosen in order to allow company specific processes while harmonizing the design artifacts to allow information exchange between tools and organizations.

This document defines the EAST-ADL2 profile. For further details, see the EAST-ADL2 specification, which defines the EAST-ADL2 metamodel.

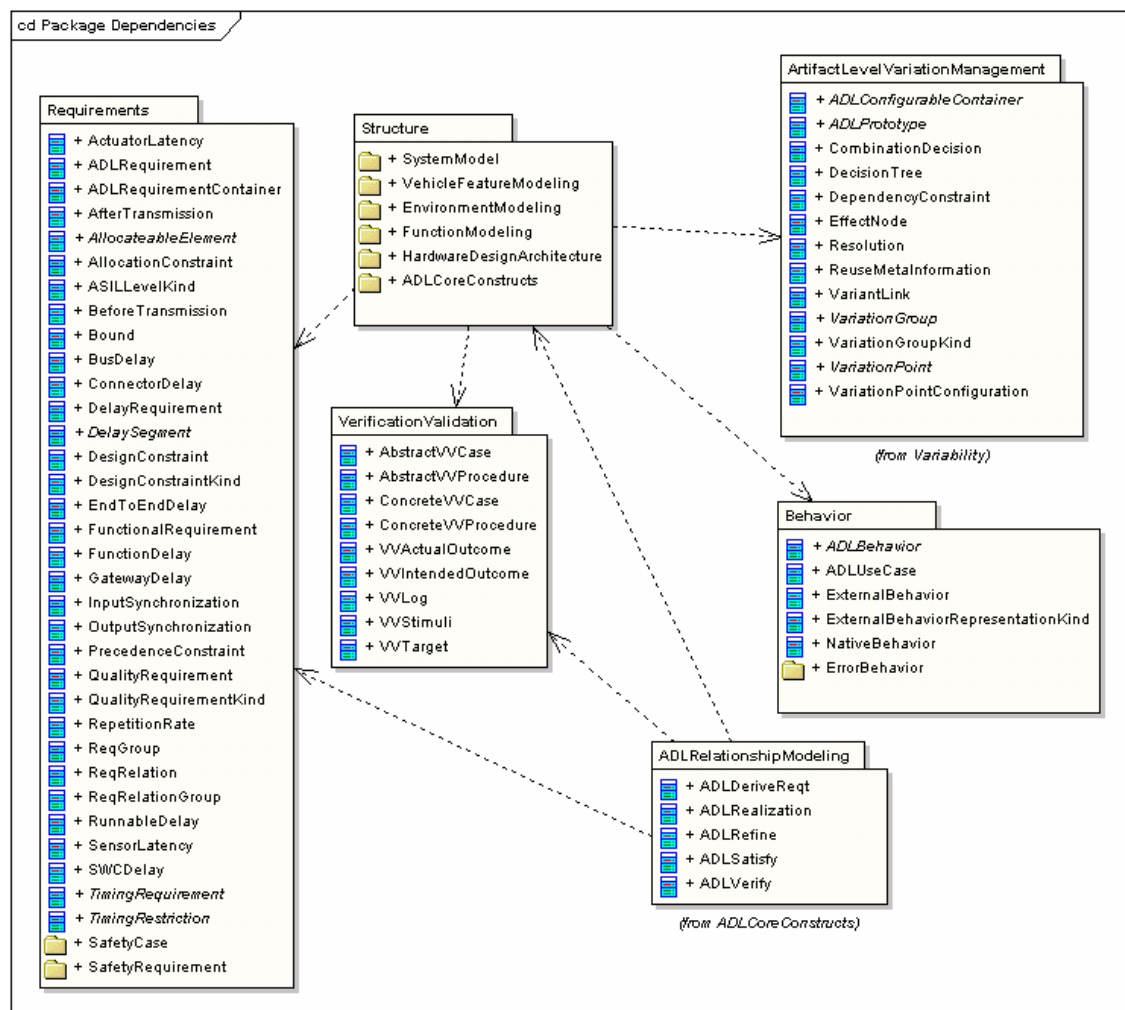


Figure 1: Package Dependencies.

1 Language Formalism

The EAST-ADL2 metamodel and profile are specified using a combination of UML modeling techniques and precise natural language to balance rigor and understandability.

2 Scope

The purpose of this specification is to specify EAST-ADL2, a modeling language for electronics systems engineering within the automotive domain. Its intent is to specify the language so that electronics systems engineering modelers may learn to apply and use EAST-ADL2, modeling tool vendors may implement and support EAST-ADL2, and both can provide feedback to improve future versions. EAST-ADL2 reuses a subset of UML 2 and of SysML (the UML profile for system engineering) and provides additional extensions to satisfy the automotive domain requirements.

The EAST-ADL2 profile is defined based on the EAST-ADL2 metamodel:

EAST-ADL2 metamodel:

A domain model (also called metamodel) related to automotive requirements for system engineering is defined, capturing only the domain specific needs of the language, without adding the UML2 details. The basic concepts of UML are used for this purpose, such as classes, compositions and associations.

EAST-ADL2 Profile:

Based on the domain metamodel, a UML2 profile for the domain metamodel is then designed. It consists in implementing in terms of UML extensions the domain model previously defined. UML extensions, stereotypes with properties and constraints, are then gathered within a UML profile: the UML profile for EAST-ADL2.

3 Normative references

The following web sites provide normative documents, which contain provisions which, through reference in this text, constitute provisions of this specification.

- UML 2.0 and MARTE, www.omg.org
- SysML, www.omg-sysml.org
- AUTOSAR www.autosar.org

Part II Structural Constructs

This part of the specification defines the structural constructs used in EAST-ADL2. The structural view of a model focuses on the static structure of the instances of the system being modeled and their static relationships. This includes the internal structure of such instances just like their external interfaces through which they can be connected to communicate with each other, by exchanging data or sending messages.

Additionally, data types are defined in this part of the specification for the exchanged data, and for parameters, are. These definitions conditions are specified that hold for the instances during their lifetime at the execution of the system . For the design of systems of arbitrary size and complexity, the possibility of hierarchical structuring of the instances is provided in the language.

EAST-ADL2 abstraction layers are used to allow reasoning of the features on several levels of abstraction. Note, however, that the abstraction levels are only conceptual; the modeling elements are organized according to the artifacts which may span more than one of these layers. Where applicable, entities on different abstraction levels are related with a realization association to allow traceability analysis. Traceability can also be deduced from the requirements structure.

The EAST-ADL2 abstraction layers with their corresponding artifacts are:

- Vehicle layer, with the Vehicle Feature Model describing function decomposition in a feature tree view with a focus on external visibility properties and organization in product line.
- Analysis level with FunctionalAnalysisArchitecture build with an abstract functional definition of the system capturing analysis support of what the system shall do, and ensuring relation with features from the Vehicle layer view. There is an n-to-m mapping between VehicleFeature and Feature entities and FunctionalAnalysisArchitecture entities (i.e., one or several functions may realize one or several features).
- Design level with FunctionalDesignArchitecture representing a decomposition of functionalities denoted in the FunctionalAnalysisArchitecture, including behavioral description but excluding software implementation constraints. The decomposition has the purpose of making it possible to meet constraints regarding non-functional properties such as for example allocation, efficiency, re-use, or supplier concerns. Again, there is an n-to-m mapping between entities of the FunctionalDesignArchitecture and the one of the FunctionalAnalysisArchitecture. Non-transparent infrastructure functionality of the AUTOSAR Basic SW Architecture, such as mode changes and error handling are also represented at the Design Level.
- Implementation level is based on AUTOSAR concepts.
- Operational level representing the binary entities and their related tools.

The Hardware Architecture and Environment Model span several abstraction levels. The Hardware Architecture models Electronic Control Units (ECUs), communication links, sensors and actuators and their connections. The Hardware Architecture is considered already at the analysis level because models of sensors, actuators, and early assumptions of hardware may be needed for the Functional Analysis Architecture

The EnvironmentModel contains Environment functions which are encapsulations of plant models, i.e. models of the behavior of the vehicle and its non-electronic systems.. Environment models are needed for validation and verification of early analysis models, all the way to the implemented embedded system.

4 EAST-ADL2 extensions for ADLCoreConstructs

This section contains the UML-profile specification, specifying stereotypes in the UML-profile, defined from the metaclasses in the ADLCoreConstructs package. It includes specification details for each stereotype. If the stereotype has properties, which may be referred to as tag definitions, or if the stereotype has constraints, this section also includes specification details for these properties and constraints.

4.1 «ADLContext» (from ADLCoreConstructs) {abstract}

Description

ADLContext is an abstract metaclass with no superclass. It is used as the common superclass for all structural metaclasses in the EAST-ADL2 domain model. It is defined to achieve a simple and practical way to associate traceable specifications and relationships to a specific context.

Extensions

None

Generalizations

None

Properties

- relationship : ADLRelationship [0..*] relationship(s) associated to this context.
- traceableSpecification : ADLTraceableSpecification [0..*] traceable specification(s) associated to this context.

Semantics

Subclasses of ADLContext provide semantics appropriate to the concepts they represent. The traceable specifications and/or relationships add no semantics, but may represent information useful for the reader of the model or for tools extracting information from the model.

Constraints

No additional constraints

4.2 «ADLEntity» (from ADLCoreConstructs) {abstract}

Description

ADLEntity is an abstract metaclass that represents an arbitrary named entity in the domain model. The name is used for identification of the element within the namespace in which it is defined.

Extensions

None

Generalizations

- ADLContext

Properties

- name : String [1] name of the ADLEntity.

Semantics

The name attribute is used for identification of the ADL entity within the namespace where its name is accessible. Note that the attribute has a multiplicity of [0..1] which provides for the possibility of the absence of a name (which is different from the empty name).

Also the ADLEntity can be used to extend the EAST-ADL approach to other languages and standards by adding a generalize relation from the respective (non EAST-ADL) element to the ADLEntity, by that be able to participate in e.g. the ADLSatisfy and ADLRealization dependencies (Extension Point).

Constraints

No additional constraints

4.3 «ADLRelationship» (from ADLCoreConstructs) {abstract}

Description

ADLRelationship is an abstract metaclass which defines a relationship, which references elements.

Extensions

None

Generalizations

None

Properties

- context : ADLContext [1] specifies the context which is associated to the relationship.

Semantics

ADLRelationship has no specific semantics. Further subclasses of ADLRelationship will add semantics appropriate to the concept they represent.

Constraints

No additional constraints

4.4 «ADLTraceableSpecification» (from ADLCoreConstructs) {abstract}

Description

ADLTraceableSpecification is an abstract metaclass which defines some kind of traceable specification. Specializations of traceable specifications are for example ADL requirements, ADL requirement containers or ADLUseCase.

Extensions

None

Generalizations

None

Properties

No additional properties

Semantics

ADLTraceableSpecification has no specific semantics. Further subclasses of ADLTraceableSpecification will add semantics appropriate to the concept they represent.

Constraints

No additional constraints

5 EAST-ADL2 extensions for ADLRelationshipModeling

This section contains the UML-profile specification, specifying stereotypes in the UML-profile, defined from the metaclasses in the ADLRelationshipModeling package. It includes specification details for each stereotype. If the stereotype has properties, which may be referred to as tag definitions, or if the stereotype has constraints, this section also includes specification details for these properties and constraints.

5.1 «ADLDeriveReq» (from ADLRelationshipModeling)

Description

ADLDeriveReq is a relationship metaclass, which signifies a dependency relationship in-between two sets of ADL requirements, showing the relationship when a set of derived client ADL requirement (client requirement) is derived from a set of ADL requirements (supplier requirement).

Extensions

- Dependency

Generalizations

- ADLRelationship
- SYSML::DeriveReq

Properties

- **derived** : ADLRequirement [1..*] The set of ADL requirements derived from the supplier ADL requirement.
- **derivedFrom** : ADLRequirement [1..*] The set of ADL requirements that the client ADL requirement are derived from.

Semantics

ADLDeriveReq metaclass signifies a derived/derived by relationship between ADLRequirements, where the modification of the supplierADLRequirement may impact the derived client ADLRequirement. The ADLDeriveReq metaclass implies the semantics that the derived client ADLRequirement is not complete, without the supplier ADLRequirement.

Constraints

No additional constraints

5.2 «ADLRealization» (from ADLRelationshipModeling)

Description

ADLRealization is a relationship metaclass, which signifies realization relationship in-between two sets of model elements, one representing a specification (the supplier) and the other represents an implementation of the latter (the client).

Extensions

- Realization

Generalizations

- ADLRelationship

Properties

- **base_Realization** : Realization [1]
- **realizedADLEntity** : ADLEntity [1..*] The set of ADL entities, which are realized by the set of client ADL entities or realized by the set of client AUTOSAR elements.
- **realizedByADLEntity** : ADLEntity [0..*] The set of client ADL entities, realizing the set of supplier ADL entities.
- **realizedByARElement** : NamedElement [0..*]

Semantics

ADLRealization metaclass signifies a realized/realized by relationship between entities, where the modification of the supplier entity impact the realizing client entity. The ADLRealization metaclass implies the semantics that the realizing client entity is not complete, without the supplier entity.

Constraints

[1] When a Feature is realized entity the realizing ADLEntity is constrained to ADLFunction, or one of its subtypes, and the realizing ARElement is constrained to.

[2] Realized ADLEntity and realizing ADLEntities/ARElements must be contained in architectures on different abstraction levels. This constraint must be refined.

[3] When a FunctionalDevices is the realized entity: the realizing ADLEntities are LocalDeviceManager and Sensor or Actuator.

No additional constraints specified in profile.

5.3 «ADLRefine» (from ADLRelationshipModeling)

Description

ADLRefine is a relationship metaclass, which signifies a dependency relationship in-between an ADL requirements and an ADL entity, showing the relationship when a client ADL Entity refines the supplier ADL requirement.

Extensions

- Dependency

Generalizations

- ADLRelationship

Properties

- base_Dependency : Dependency [1]
- refinedBy : ADLEntity [1..*] List of ADLEntity participating to the refinement of the refined ADL requirements.
- refinedRequirement : ADLRequirement [1..*] List of refined requirements.

Semantics

ADLRefine metaclass signifies a refined requirement/refined by relationship between an ADLRequirement and an ADL entity, where the modification of the supplierADLRequirement may impact the refining client ADLEntity. The ADLRefine metaclass implies the semantics that the refining client ADLEntity is not complete, without the supplier ADLRequirement.

Constraints

[1] The property refinedBy must not have the types ADLRequirement or ADL RequirementContainer.

No additional constraints specified in profile.

5.4 «ADLSatisfy» (from ADLRelationshipModeling)

Description

ADLSatisfy is a relationship metaclass, which signifies relationship in-between ADLRequirement and satisfying ADL entity, showing the relationship when a satisfied supplier ADL Requirement is satisfied by the client ADL entity or satisfied by the client AUTOSAR element.

Extensions

- Dependency

Generalizations

- ADLRelationship
- SYSML::Satisfy

Properties

- satisfiedByADLEntity : ADLEntity [0..*] List of satisfying client ADL entities, which satisfies the supplier ADL requirements.
- satisfiedByARElement : NamedElement [0..*]
- satisfiedRequirement : ADLRequirement [1..*] List of satisfied ADL requirements, which are satisfied by the client ADL entities or satisfied by the client AUTOSAR elements.

Semantics

ADLSatisfy metaclass signifies a satisfied requirement/satisfied by relationship between a set of ADL requirements and a set of satisfying entities, where the modification of the supplier ADL requirements may

impact the satisfying client entities. The ADLSatisfy metaclass implies the semantics that the satisfying client entities are not complete, without the supplier ADL requirement.

Constraints

[1] Satisfy by ADLEntity may not be an ADLRequirement or ADLRequirementContainer.

No additional constraints specified in profile.

5.5 «ADLVerify» (from ADLRelationshipModeling)

Description

ADLVerify is a relationship metaclass, which signifies a dependency relationship in-between an ADL requirement and a VV Case, showing the relationship when a client VV Case verifies the supplier ADL requirement.

Extensions

- Dependency

Generalizations

- ADLRelationship
- SYSML::Verify

Properties

- targetProcedure : ConcreteVVProcedure [0..*]
- verifiedBy : ConcreteVVCASE [1..*]
- verifiedRequirement : ADLRequirement [1..*] The set of ADL requirements which the client VV cases verify.

Semantics

ADLVerify metaclass signifies a refined requirement/verified by relationship between an ADLRequirement and a VV case, where the modification of the supplierADLRequirement may impact the verifying client VV case. The ADLVerify metaclass implies the semantics that the verifying client VV case is not complete, without the supplier ADLRequirement.

Constraints

No additional constraints

6 EAST-ADL2 extensions for ADLTypes

This section contains the UML-profile specification, specifying stereotypes in the UML-profile, defined from the metaclasses in the ADLTypes package. It includes specification details for each stereotype. If the stereotype has properties, which may be referred to as tag definitions, or if the stereotype has constraints, this section also includes specification details for these properties and constraints.

6.1 «ADLDesignDataType» (from ADLTypes)

Description

ADLDesignDataType is a DataType metaclass, which signifies a type whose instances are identified only by their value. The ADLDesignDataType metaclass describes the value set for some variable, parameter etc. without a description of how these possible values are represented on implementation level. The implementation representation is defined on implementation level, and the implemented datatype shall be associated with an ADLRealization relationship. The realizing datatype must match the ADLDesignDataType regarding range, resolution, unit, and dimension.

Extensions

- DataType

Generalizations

- SYSML::ValueType

Properties

- accuracy : String [0..1] The accuracy of the value typed by the ADLDesignDatatype.
- adlbaseDataType :
ADLDesignDataTypeKind =
String [1]
- defaultValue : String [0..1] Defines a default value to be used to initialize the attribute.
Same type as Datatype.
- description : String [1] Description of the data type.
- physicalMax : String [0..1] The maximum value the attribute might represent in the
physical world, e.g. +230° Celsius. Same type as Datatype.
- physicalMin : String [0..1] The minimum value the attribute might represent in the
physical world, e.g. -50° Celsius. Same type as Datatype.
- resolution : String [0..1] The resolution of the variable expressed as the size of the
minimum difference between variables. Same type as
Datatype.
- significantDigits : Integer [0..1] The number of significant digits.

Semantics

ADLDesignDataType is a special kind of classifier, similar to a class. It differs from the class in that instances of a data type are identified only by their value.

Constraints

[1] The realizing AR PrimitiveTypeWithSemantics has to be consistent w.r.t. range, resolution, etc.

No additional constraints specified in profile.

6.2 *ADLDesignDataTypeKind (from ADLTypes)*

Description

ADLDesignDataTypeKind is an enumeration for the available basic data types for a certain datatype.

Generalizations

None

Enumeration Literals

- Boolean
- Enumeration
- Integer
- Real
- String

Semantics

The enumerations of ADLDesignDataTypeKind correspond to the datatypes Boolean, Enumeration, Integer, Real and String according to their mathematical definitions.

Constraints

No additional constraints

6.3 «ADLDouble» (from ADLTypes)

Description

An instance of double is an element from the set of real numbers. The value must comply with IEEE 754.

Extensions

- DataType

Generalizations

None

Properties

- base_DataType : DataType [1]

Semantics

ADLDouble has the semantics of the Double datatype as defined by IEEE

Constraints

No additional constraints

6.4 «ADLFloat» (from ADLTypes)

Description

An instance of double is an element from the set of real numbers. The value must comply with IEEE 754.

Extensions

- PrimitiveType

Generalizations

None

Properties

- base_PrimitiveType : PrimitiveType [1]

Semantics

ADLFloat has the semantics of the Float datatype as defined by IEEE

Constraints

No additional constraints

7 EAST-ADL2 extensions for SystemModel

This section contains the UML-profile specification, specifying stereotypes in the UML-profile, defined from the metaclasses in the SystemModel package. It includes specification details for each stereotype. If the stereotype has properties, which may be referred to as tag definitions, or if the stereotype has constraints, this section also includes specification details for these properties and constraints.

7.1 «ADLSystemModel» (from SystemModel)

Description

The ADLSystemModel metaclass is a namespace whose members can include other features. The ADLSystemModel is a practical way to associate other models/architectures to a specific context.

Extensions

- Element
-
- Class

Generalizations

- ADLEntity

- ADLFunctionType

Properties

- analysisArchitecture : AnalysisArchitecture [0..1] The AnalysisArchitecture contained in the SystemModel and connected to the EnvironmentModel through ports/connectors
- designArchitecture : DesignArchitecture [0..1] The designArchitecture contained in the SystemModel and connected to the EnvironmentModel through ports/connectors
- environmentModel : EnvironmentModel [0..1] The environment model contained in the SystemModel and connected to the Architectures of the ADLSystemModel through ports/connectors
- implementationArchitecture : Element [0..1] The implementation architecture is the AUTOSAR System from the SystemTemplate
- requirementInterchange : Element [0..1]
- vehicleFeatureModel : VehicleFeatureModel [0..1] The Vehicle Feature Model contained in the SystemModel.

Semantics

The ADLSystemModel represents the electronics and software of the vehicle.

Constraints

No additional constraints

7.2 «AnalysisArchitecture» (from SystemModel)

Description

The AnalysisArchitecture is the representation of the EE architecture on the analysis abstraction level. It contains a FunctionalAnalysisArchitecture which represent the abstract functional behavior.

Extensions

- Class

Generalizations

- ADLFunctionType

Properties

- functionAnalysisArchitecture : FunctionalAnalysisArchitecture [1]

Semantics

The AnalysisArchitecture is the representation of the EE architecture on the analysis abstraction level. It corresponds to the complete EE architecture, with interfaces such as sensors, actuators and HMI as boundary. Non-electric/electronic entities are not included.

Constraints

No additional constraints

7.3 «DesignArchitecture» (from SystemModel)

Description

The DesignArchitecture is a container element for the FunctionalDesignArchitecture, MWAbstraction and HardwareDesignArchitecture. It represents the EE architecture of the vehicle, as specified by a FunctionalAnalysisArchitecture or a VehicleFeatureModel (if no FunctionalAnalysisArchitecture has been defined).

Extensions

- Class

Generalizations

- ADLFunctionType

Properties

- functionalDesignArchitecture : FunctionalDesignArchitecture [1]
- hardwareDesignArchitecture : HardwareDesignArchitecture [1]
- MWAbstraction : MWAbstraction [1]

Semantics

The DesignArchitecture is the representation of the EE architecture on the design abstraction level. It corresponds to the complete EE architecture, with interfaces such as sensors, actuators and HMI as boundary. Non-electric/electronic entities are not included.

Constraints

No additional constraints

7.4 «FunctionalDesignArchitecture» (from SystemModel)

Description

The FunctionalDesignArchitecture metaclass is a top level ADLFunctionType. It is supposed to implement all the functionalities of a vehicle, as specified by a FunctionAnalysisArchitecture or a VehicleFeatureModel (if no FunctionAnalysisArchitecture has been defined during the process).

Extensions

- Class

Generalizations

- ADLFunctionType

Properties

No additional properties

Semantics

The FunctionalDesignArchitecture is a container for the application software of the E/E architecture.

The FunctionalDesignArchitecture contains LocalDeviceManager and parts typed by ADLFunctionTypes. The view of the HardwareArchitecture facilitates the realization of LocalDeviceManager as sensor/actuator HW elements.

Contains

- LocalDeviceManager

Constraints

[1] A FunctionalDesignArchitecture may not be used as type of a ADLFunctionPrototype.

No additional constraints specified in profile.

7.5 «MWAbstraction» (from SystemModel)

Description

The MWAbstraction class represents the functional abstraction of the middleware. Transparent services of the middleware, such as communication or task scheduling are not represented. Services for error handling, diagnostics or mode handling are represented such that their impact on application's behaviors can be estimated.

Extensions

- Class

Generalizations

- ADLFunctionType

Properties

No additional properties

Semantics

The contents of the MWAbstraction represents the middleware functionality as it appears to the application functionality of the FDA.

The interaction of an ADLFunction in the FDA with ADLFunctions of the MWAbstraction indicates that the ECU where this application function eventually resides will provide access to the functionality represented by the MWAbstraction ADLFunctionType.

ADLFunctions that are contained in the MWAbstraction entity are realized by middleware or basic software components as opposed to application software components.

Constraints

No additional constraints

8 EAST-ADL2 extensions for FunctionModeling

This section contains the UML-profile specification, specifying stereotypes in the UML-profile, defined from the metaclasses in the FunctionModeling package. It includes specification details for each stereotype. If the stereotype has properties, which may be referred to as tag definitions, or if the stereotype has constraints, this section also includes specification details for these properties and constraints.

8.1 «ADLClientPort» (from FunctionModeling)

Description

The ADLClientPort metaclass denotes a port for clients in a client-server interaction

Extensions

- Port

Generalizations

- ADLClientServerPort

Properties

- base_Port : Port [1]

Semantics

The client port represent an interaction point with a server. The operations that are called are defined in the interface associated to the port.

The ADLClientPort denotes that the containing ADLFunction requires access to a server according to the associated interface.

Constraints

No additional constraints

8.2 «ADLClientServerPort» (from FunctionModeling) {abstract}

Description

ADLClientServerPort metaclass is an abstract port for client-server interaction, which is subtyped to allow for different notation in an ATESSST tool.

Extensions

None

Generalizations

- ADLEntity

Properties

- type : DesignInterface [1] The interface of this port.

Semantics

TBD

Constraints

No additional constraints

8.3 «ADLConnectorPrototype» (from FunctionModeling)

Description

See description for AUTOSAR SWComponentTemplate::Composition::ConnectorPrototype.

Extensions

- Connector

Generalizations

- ADLEntity
- ADLVariableElement

Properties

- base_Connector : Connector [1]
- clientServerPort : ADLClientServerPort [0..2]
- flowPort : ADLFlowPort [0..2] The ports that are connected by this connector.

Semantics

The ADLConnectorPrototype indicates that the connected ports exchange signals or client-server requests-responses.

The ADLConnectorPrototype connects a pair of flowports or a client and server ports. If two FlowPorts are connected, data elements of the type of the port flows from the output to the inport. If a client and server port are connected, the client calls the server according to the operations in the interfaces.

Constraints

No additional constraints

8.4 «ADLDiscFunctionType» (from FunctionModeling)

Extensions

- Class

Generalizations

- ADLFunctionType

Properties

- behavior : ADLBehavior [0..1]
- triggerparameters : Trigger [0..1]

Semantics

TBD

Constraints

No additional constraints

8.5 «ADLFlowPort» (from FunctionModeling) {abstract}

Description

The ADLFlowPort is an abstract metaclass, subtyped to allow for different notation in an ATESSST tool.

Extensions

- Port

Generalizations

- ADLEntity
- SYSML::FlowPort

Properties

- period : Element [1] Period in milliseconds.
- type : ADLDesignDataType [1] The single data type for this port.

Semantics

ADLFlow ports may be connected by:

- assembly connectors from ADLOutFlowPort to ADLInFlowPort
- delegation connectors from ADLOutFlowPort on part to ADLOutFlowPort on composition
- delegation connectors from ADLInFlowPort on composition to ADLInFlowPort on part

ADLOutFlowPort may be connected to ADLInFlowPort if their port signatures match. Matching is defined as:

The source ADLFlowPort ADLOutFlowPort is typed by a ADLDesignDataType which is compatible with the ADLDesignDataType of the destination ADLFlowPort

ADLDesignDataTypes are compatible if

- * They have the same Dimension
- * The source ADLDesignDataType has same or better Precision
- * The source ADLDesignDataType has same or better Accuracy
- * The source ADLDesignDataType has same or smaller maxValue
- * The source ADLDesignDataType has same or higher minValue

Constraints

[1] ADLFlowPort may only be typed by ADLDesignDataType.

No additional constraints specified in profile.

8.6 «ADLFunctionType» (from FunctionModeling)

Description

The ADLFunctionType metaclass is based on SysML metaclass Block, and not AUTOSAR ComponentType, to get clear distinction between design (SysML) and implementation (AUTOSAR).

The ADLFunctionType metaclass is used for problem description in FAA and solution description in FDA, and it will appear as parts of these.

FAA

When used on the FAA level, the ADLFunctionType is representing the analysis function used to describe the functionalities provided by a car on the analysis level. At the analysis level, functions are defined and structured according to the functional requirements, i.e. the functionalities provided to the user. A more optimized structure (from an implementation point of view) is defined at the design level in the Functional Design Architecture.

The analysis functions may interact with other analysis functions and FunctionalDevices through their ports.

Furthermore an analysis function may be decomposed into (sub-)analysis functions. This allows breaking up hierarchically the functionalities provided by the parent analysis function into sub-functionalities.

A behavior description may be associated with each analysis function. In the case where the analysis function is decomposed, the behavior is a specification for the composed behavior of the sub-analysis functions. If the analysis function is not decomposed, then the behavior is describing the behavior of the sub-analysis functions which is to be used when building the global behavior of the FunctionalAnalysisArchitecture by composition of the leaf behaviors.

FDA

When used on the FDA level, the ADLFunctionType is representing the elementary software function that is used to describe the leaves of the functional hierarchy. The composition of these leaves makes up the implementation behavior of the entire functional hierarchy.

Variability (on FAA and FDA level):

Ports can be used in an ADLFunctionType that is a variationPoint or variableElement. The cardinality of a port can be given as [1] or [0..1] when the role is variationPoint. The cardinality [1] means that the variants have to obey this contract, the cardinality [0..1] means that it is optional for the variants to follow this contract. This has the following implications in the context where the ADLFunctionPrototype of the ADLFunctionType is used:

- A connected port (now called other port) to this port, of another ADLFunctionPrototypes need to have a corresponding cardinality in the port-owning ADLFunctionType. The following table gives the cardinality of that other port, for the different types of the port in this variationPoint:

If the port of the variationPoint has cardinality [1] and has the type:

- InFlowPort, an input is required, hence cardinality [1] is required on the other port (an OutFlowPort).
- OutFlowPort, it is optional to connect to this port, if the other ADLFunctionType is variable the other port may have [0..1].
- ServerPort, it is optional to connect to this port, if the other ADLFunctionType is variable the other port may have [0..1].
- ClientPort, a corresponding ServerPort is required, gives [1] on the other port.

If the port of the variationPoint has cardinality [0..1] nothing can be said on the unresolved model, see resolved model below. For the types:

- InFlowPort, input might be required by the variants.
- OutFlowPort, output might be delivered by the variants.
- ServerPort, the port might provide operations.
- ClientPort, the port might request operations.

In the resolved model, meaning that all variationPoints have been replaced by the designed variants:

The port of the variationPoint with cardinality [1] must have a corresponding port of cardinality [1] in the variant.

The port of the variationPoint with cardinality [0..1] has been resolved to [0] or [1] by the variant.

- If the port of the variant has cardinality [1], the same rules apply as for the variationPoint with cardinality [1].
- If the port of the variant has cardinality [0] the other port may be resolved as [0], i.e. neither connectors nor other ports are required.
- Note that every ADLFunctionType variationpoint must be designed such that each variant must have at

least one non-optional port, i.e. cardinality [1].

A port with cardinality [0..1] may only be connected to an "other" port of cardinality [0..1], which makes the owning "other" ADLFunctionType variable as well. The possible variant to select is deduced from the resolved model. This can be compared with the case when the VariationPointConfiguration attribute Optional is true for a variationPoint ADLFunctionType, this implies that other ports connected to this ADLFunctionType must have the cardinality [0..1].

The ADLFunctionType can only contain a feature model if the ADLFunctionType itself includes any kind of internal variability, i.e. the ADLFunctionType must be at least a composite software function.

Extensions

- Class

Generalizations

- ADLEntity
- AllocateableElement
- ADLVariableElement
- SYSML::Block

Properties

- clientServerPort : ADLClientServerPort [0..*] Owned client and server ports.
- connector : ADLConnectorPrototype [0..*] The connectors that connect ports of parts as assembly connectors or ports of this type and ports of parts as delegation connectors.
- flowPort : ADLFlowPort [0..*] Owned in- and out-flow ports.
- hwElement : ADLHwElement [0..*]
- isElementary : Boolean = false [1] True, when this type does not have any parts.
- localFeatureModel : FunctionFeatureModel [0..1]
- part : ADLStructurePrototype [0..*] The parts contained in this ADLFunctionType.
- pDecModel : ProductDecisionModel [0..1]

Semantics

The ADLFunction represents a node in a tree structure corresponding to the functional decomposition of a top level analysis function.

If a behavior is attached, the execution semantic for a discrete elementary ADLFunctionType complies with the run-to-completion semantic. This has the following implications:

1. Input that arrives at the input ports after execution begin will be ignored until the next execution cycle.
2. If more than one input value arrives per Port and Signal before execution begin, the last value will override all previous ones in the public part of the input port (single element buffers for input).
3. The local part of a port does not change its value during execution of the behavior.
4. During an execution cycle only one output value can be sent per Port and Signal. If consecutive output values are produced for one Signal on the same Port during a single execution cycle, the last value will override all previous ones in the local part of the output port (single element buffers for output).
5. Output will not be available at public part of an output port before execution end.

6. Elementary ADLFunctionTypes may not produce any side effects (i.e. all data passes the ports).

If 2 or more instances of an elementary software function are placed on the same ECU, the code will be placed on the ECU only once (so these instances will use the same code but separate memory areas for data).

Constraints

[1] FAA & FDA: Ports that are part of an analysis function must be EAST-ADL2 Ports.

[2] FAA: Classes and Components that are parts of an analysis function must be analysis functions.

[3] FDA: Elementary ADLFunctionTypes may not have classes or components as parts.

[4] Variability: For all contained variation points there must exist a corresponding variation point configuration in ownedVariationGroup.

[5] Variability: An ADLFunctionType that is a variation point must have at least one port.

[6] If isElementary then the ADLFunctionType does not have a ProductDecisionModel.

No additional constraints specified in profile.

8.7 «ADLInFlowPort» (from FunctionModeling)

Description

The ADLInFlowPort represents a port that requires data input. The data is specified by the associated datatype, and may be discrete or continuous in value. The value may be continuous or discrete in time, depending on the properties of the sending ADLFunction.

Extensions

- Port

Generalizations

- ADLFlowPort

Properties

No additional properties

Semantics

The ADLInFlowPort indicates that the containing entity, typically a n ADLFunction, requires input data. The datatype of this data is defined by the associated datatype. The data is sampled at the invocation of the containing entity for discrete ADLFunctions. For continuous ADLFunctions the port represents a continuous input connection point.

For non-elementary ADLFunctions, the temporal properties are defined by the innermost (elementary) ADLFunction.

The ADLInFlowPort declares a reception point of data. It represents a single element buffer which is overwritten with the latest data. The type of the data is defined by the associated ADLDesignDatatype. If isTriggered is true, this means that the owning ADLFunction is triggered on data reception on this port. See ADLFlowPort for further semantics.

Constraints

[1] isAtomic = TRUE.

[2] Typed by ADLDesignDataType.

[3] The direction attribute inherited from SysML::FlowPort must have the value IN.

No additional constraints specified in profile.

8.8 «ADLInOutFlowPort» (from FunctionModeling)

Description

The ADLInOutFlowPort metaclass is a Port with both provided and required interfaces. The direction attribute has the value INOUT.

Extensions

- Port

Generalizations

- ADLFlowPort

Properties

No additional properties

Semantics

The ADLInOutFlowPort indicates that the containing entity, typically an ADLFunction, is able to send and receive data. The datatype of this data is defined by the associated datatype. The data is sampled at the invocation of the containing entity for discrete ADLFunctions. For continuous ADLFunctions the port represents a continuous input connection point.

For non-elementary ADLFunctions, the temporal properties are defined by the innermost (elementary) ADLFunction.

The ADLInOutFlowPort declares a bi-directional port. With semantics according to the ADLInFlowPort and ADLOutFlowPort. See ADLFlowPort and ADLInFlowPort and ADLOutFlowPort for further semantics.

Constraints

No additional constraints

8.9 «ADLOutFlowPort» (from FunctionModeling)

Description

The ADLOutFlowPort metaclass is a port that provides data according to the associated datatype.

Extensions

- Port

Generalizations

- ADLFlowPort

Properties

No additional properties

Semantics

The ADLOutFlowPort indicates that the containing entity, typically an ADLFunction, provides output data. The datatype of this data is defined by the associated datatype. The data is sent at the completion of the containing entity for discrete ADLFunctions. For continuous ADLFunctions the port represents a (time)continuous output connection point.

For non-elementary ADLFunctions, the temporal properties are defined by the innermost (elementary) ADLFunction, i.e. the source of the data.

The ADLOutFlowPort declares a transmission point of data. The type of the data is defined by the associated ADLDesignDataType. See ADLFlowPort for further semantics.

Constraints

[1] isAtomic = TRUE.

[2] Typed by ADLDesignDataType.

[3] The direction attribute inherited from SysML::FlowPort must have the value OUT.

No additional constraints specified in profile.

8.10 «ADLPortGroup» (from FunctionModeling)

Description

The ADLPortGroup metaclass is used to collapse several ports to one. All ports that are part of a port group are graphically represented as a single port. Connectors connected to ports of a port group pair are graphically collapsed to a single line.

The ADLPortGroup has no semantic meaning except that it makes graphical representation of the connected ports easier to read, and provides a means to logically organize several ports to one group.

Connectors are still connected to the contained ports, but tool support may simplify connections by allowing semi-automatic or automatic connection to all ports of a port group.

Extensions

- Class

Generalizations

- ADLEntity

Properties

- base_Class : Class [1]
- groupedPort : ADLFlowPort [1..*] The grouped ports.

Semantics

The ADLPortGroup provides a means to organize ports and connectors. It does not add semantics. In the model, the ports contained in the port group are connected as individual ports.

Constraints

No additional constraints

8.11 «ADLServerPort» (from FunctionModeling)

Description

The ADLServerPort metaclass is a port for servers in a client-server interaction.

The interface type of the port specifies the operations that are provided.

Extensions

- Port

Generalizations

- ADLClientServerPort

Properties

- base_Port : Port [1]

Semantics

The ADLServerPort is an interaction point for client-server communication. The operations of the ADLClientServerInterface that types the port, are provided by the containing ADLFunction.

Constraints

No additional constraints

8.12 «ADLStructurePrototype» (from FunctionModeling)

Extensions

- Property

Generalizations

- ADLEntity
- SYSML::BlockProperty

Properties

- type : ADLFunctionType [1]

Semantics

TBD

Constraints

No additional constraints

8.13 «ADLVariableElement» (from FunctionModeling) {abstract}

Description

The variable element is an abstract class for artifact elements that can be variable. Essentially the classes inheriting from variable element (e.g. a function type) have all the associations and other relationships of the variable element, especially dependency constraints and variant link. In most cases it is required that the respective class also inherits from the abstract class variation point in order to have the variation point configuration relations.

Also the variable element can be used to extend the EAST-ADL variability approach to other languages and standards by adding a generalize relation from the respective (non EAST-ADL) element to the variable element (Extension Point).

Extensions

None

Generalizations

None

Properties

- configuration : String [1] The configuration string describes the variability of the element and the constraints of the variability for the configuration.
- reuseMetaInformation :
ReuseMetaInformation [0..1]
- variationGroup : VariationGroup
[0..*]

Semantics

This is an abstract class providing needed variability relationships for concrete classes refining this class.

Constraints

No additional constraints

8.14 «DesignInterface» (from FunctionModeling)

Extensions

- Interface

Generalizations

None

Properties

- base_Interface : Interface [1]

Semantics

TBD

Constraints

No additional constraints

8.15 «FunctionalAnalysisArchitecture» (from FunctionModeling)

Description

The FunctionalAnalysisArchitecture metaclass is a top level ADLFunctionType. It is an abstract functional representation of the embedded system and realizes the features of a vehicle, as specified by a VehicleFeatureModel.

Extensions

- Class

Generalizations

- ADLFunctionType

Properties

No additional properties

Semantics

The FunctionalAnalysisArchitecture represents the complete functionality of the embedded system. The border goes by sensors and electrical actuators.

Constraints

[1] A FunctionalAnalysisArchitecture may not be used as type of an ADLFunctionPrototype.

No additional constraints specified in profile.

8.16 «FunctionalDevice» (from FunctionModeling)

Description

The Functional Device metaclass is an abstract sensor or actuator that encapsulates sensor/actuator dynamics and the interfacing software. The FunctionalDevice is the interface between the electronic architecture and the environment. As such, it is a transfer function between the ADLFunctionType and the physical entity that it measures or actuates.

A Realization dependency can be used to define LocalDeviceManagers and Sensors/Actuators that are represented by the Functional Device.

Semantics

The behavior associated with the FunctionalDevice is the transfer function between the EnvironmentModel representing the environment and an ADLFunction of the FunctionalAnalysisArchitecture. The transfer function represents the sensor or actuator and its interfacing HW and SW (connectors, electronics and I/O, driver software and application software)

Extensions

- Class

Generalizations

- ADLFunctionType

Properties

No additional properties

Semantics

TBD

Constraints

[1] Ports associated with the FunctionalDevice must be ADLFlowPorts.

[2] The FunctionalDevice may not be hierarchical.

[3] Only appear as part of the FunctionalAnalysisArchitecture.

No additional constraints specified in profile.

8.17 «LocalDeviceManager» (from FunctionModeling)

Description

The LocalDeviceManager (LDM) metaclass is a special kind of ADLFunctionType that is the software interface to sensors and actuators. It interacts with the HardwareAbstractionFunction and contains the functional aspects rather than the computer hardware aspects of interfacing. This interaction takes place over SystemPorts.

Extensions

- Class

Generalizations

- ADLDiscFunctionType

Properties

No additional properties

Semantics

The LocalDeviceManager encapsulates the device specific or functional parts of a sensor or actuator interface.

The period attribute is interpreted as the trigger period in case of periodic operation, and minimum period in case of event triggered operation.

Constraints

[1] The LocalDeviceManager has to be allocated to the same ECU as the Sensor or Actuator that it interfaces

[2] A LocalDeviceManager may only interface either Sensors or Actuators.

[3] Only appear as part of the FunctionalDesignArchitecture.

No additional constraints specified in profile.

8.18 «Trigger» (from FunctionModeling)

Description

The trigger class contains the trigger parameters necessary to define the execution of the containing ADLFunctionType.

Extensions

None

Generalizations

None

Properties

- offset : TimingRestriction [1] The time from the trigger of the ADLFunctionType until its invocation
- triggerCondition : String [1] An OCL expression that allows release of the ADLFunctionType only if it evaluates to TRUE.
- triggerPeriod : This is the period of the ADLFunctionType TrigPolicy is periodic,

- TimingRestriction [1] and inter-arrival time if TrigPolicy is Event
- trigPolicy : String [1] Defines whether time or trigger events on ports makes the Function execute

Semantics

The Trigger attributes defines the invocation pattern of the contained ADLFunction.

The containing ADLFunction is triggered according to the attributes of the Trigger class. Periodic ADLFunctions has trigPolicy=TIME, while event-triggered ADLFunctions has trigPolicy=EVENT.

Periodic ADLFunctions are executed with an interval of triggerPeriod with an offset of offset. The ADLFunction is executed only if its triggerCondition evaluates to TRUE.

Event-triggered ADLFunctions are executed after offset, once the ports marked as isTriggered receive data and the triggerCondition evaluates to TRUE. The interval since the last invocation must be at least triggerPeriod.

Constraints

Triggers may only be contained in ADLFunctionTypes with isTriggered set to true.

No additional constraints specified in profile.

9 EAST-ADL2 extensions for HardwareDesignArchitecture

This section contains the UML-profile specification, specifying stereotypes in the UML-profile, defined from the metaclasses in the HardwareDesignArchitecture package. It includes specification details for each stereotype. If the stereotype has properties, which may be referred to as tag definitions, or if the stereotype has constraints, this section also includes specification details for these properties and constraints.

9.1 «Actuator» (from HardwareDesignArchitecture)

Description

The Actuator is the ADL entity that represents electrical actuators, such as motors, lamps, brake units, etc. Non-electrical actuators such as the engine, hydraulics, etc. are considered part of the plant model (environment). Plant models are not part of the Hardware Design Architecture. The behavior of the actuator is represented by its contained ADLFunction (See generalization).

Extensions

- Class

Generalizations

- IoComponent

Properties

No additional properties

Semantics

The actuator entity represents the actuator hardware and its connectors. Its behavior is represented by the contained ADLFunction.

Constraints

No additional constraints

9.2 «ADLHwConnector» (from HardwareDesignArchitecture) {abstract}

Description

See specializations.

Extensions

- Connector

Generalizations

- ADLEntity

Properties

- base_Connector : Connector [1]

Semantics

See specializations.

Constraints

No additional constraints

9.3 «ADLHwElement» (from HardwareDesignArchitecture) {abstract}

Description

The ADLHwElement is the ADL entity that is used to describe functional hardware. It is abstract and is thus not instantiated.

Extensions

- Class

Generalizations

- ADLEntity

Properties

- base_Class : Class [1]
- connector : ADLHwConnector [0..*]
- part : ADLHwElementPrototype [0..*]
- port : ADLHwPort [0..*]
- transferFunction : ADLFunctionType [0..1]

Semantics

The HwElement is a structural entity that is part of an electrical architecture. Through its ports it can be connected to electrical sources and sinks and ground. Its logical behavior is represented by the contained ADLFunction, the transfer function. This is typically connected through its ports to the environment model and to the middleware abstraction to participate in the end-to-end behavioral definition of a function.

Constraints

No additional constraints

9.4 «ADLHwElementPrototype» (from HardwareDesignArchitecture)

Description

Appear as parts of an ADLHwElement and is itself typed by an ADLHwElement. This allows for a reference to the occurrence of an ADLHwElement when it acts as a part. The purpose is to support the definition of hierarchical structures, and to reuse the same type of Hardware at several places. For example, a wheel speed sensor may occur at all four wheels, but it has a single definition.

Extensions

- Property

Generalizations

None

Properties

- base_Property : Property [1]
- type : ADLHwElement [1]

Semantics

The ADLHwElementPrototype represents an occurrence of a hardware element, according to the type of the ADLHwElementPrototype.

Constraints

No additional constraints

9.5 «ADLHwPort» (from HardwareDesignArchitecture) {abstract}

Description

The ADLHwPort represents electrical connection points in the hardware architecture. Depending on type of port, the actual wire or a logical connection can be considered.

Extensions

- Port

Generalizations

None

Properties

- base_Port : Port [1]

Semantics

See specializations.

Constraints

No additional constraints

9.6 «CommunicationHwPort» (from HardwareDesignArchitecture)

Description

IoHwPort represents an electrical connection point that can be used to define how the wire harness is logically defined.

Extensions

- Port

Generalizations

- ADLHwPort

Properties

No additional properties

Semantics

The CommunicationHwPort represents the hardware connection point of a communication bus. Depending on modeling style, the port may represent the logical connection point of the bus or the physical pins.

Constraints

No additional constraints

9.7 «ECU» (from HardwareDesignArchitecture)

Description

The Electronic Control Unit (ECU) models the computer nodes of the embedded system. ECUs consist of processor(s) and may be connected to sensors, actuators and other ECUs via a SystemBusConnector.

Extensions

- Class

Generalizations

- ADLHwElement

Properties

No additional properties

Semantics

The ECU element represents Electronic Control Unit.

Constraints

No additional constraints

9.8 «HardwareDesignArchitecture» (from HardwareDesignArchitecture)

Description

The Hardware Design Architecture is the hardware design from a system perspective. The HardwareDesignArchitecture has three purposes:

It shows the physical entities and how they are connected.

It is an allocation target for the ADLFunctions of the FDA and MWA

It represents the transfer functions of the contained hardware entities.

Extensions

- Class

Generalizations

- ADLEntity
- ADLFunctionType

Properties

No additional properties

Semantics

The HardwareDesignArchitecture represent the hardware architecture of the embedded system. Its contained HWElements represents the physical aspects of the hardware entities and how they are connected. Its contained ADLFunctions represents the logical behavior of the contained HWElements.

Constraints

No additional constraints

9.9 «IoComponent» (from HardwareDesignArchitecture) {abstract}

Description

See specializations.

Extensions

- Class

Generalizations

- ADLHwElement

Properties

No additional properties

Semantics

See specializations.

Constraints

No additional constraints

9.10 «IoHwConnector» (from HardwareDesignArchitecture)

Description

-

Extensions

- Connector

Generalizations

- ADLHwConnector

Properties

- port : IoHwPort [2]

Semantics

The IoHwConnector represents a physical electrical line that connects IoHwPorts.

Constraints

No additional constraints

9.11 «IoHwPort» (from HardwareDesignArchitecture)

Description

IoHwPort represents an electrical connection point that can be used to define how the wire harness is logically defined.

Extensions

- Port

Generalizations

- ADLHwPort

Properties

No additional properties

Semantics

The IoHwPort represents an electrical pin or connection point.

Constraints

No additional constraints

9.12 «PowerHwConnector» (from HardwareDesignArchitecture)

Description

-

Extensions

- Connector

Generalizations

- ADLHwConnector

Properties

- port : PowerHwPort [2]

Semantics

The PowerHwConnector denotes a electrical connector that supplies power.

Constraints

No additional constraints

9.13 «PowerHwPort» (from HardwareDesignArchitecture)

Description

-

Extensions

- Port

Generalizations

- ADLHwPort

Properties

No additional properties

Semantics

CommunicationHwPort denotes a connection point for bus lines. Depending on modeling style, one or two ports may be defined for a dual-wire bus.

Constraints

No additional constraints

9.14 «PowerSupply» (from HardwareDesignArchitecture)

Description

-

Extensions

- Class

Generalizations

- ADLHwElement

Properties

- isActive : Boolean = true [1]

Semantics

PowerSupply denotes a power source that may be active (e.g. a battery) or passive (main relay)

Constraints

No additional constraints

9.15 «Sensor» (from HardwareDesignArchitecture)

Description

Sensor is a hardware entity that represents digital and analog sensor elements. The Sensor entity is connected electrically to the electrical entities of the HDA. The logical behavior is defined in the transfer function, and this one is also connected to the environment model.

Extensions

- Class

Generalizations

- IoComponent

Properties

No additional properties

Semantics

Sensor denote an electrical sensor. The actuator entity represents the sensor hardware and its connectors. Its behavior is represented by the contained ADLFunction.

Constraints

No additional constraints

9.16 «SystemBusConnector» (from HardwareDesignArchitecture)

Description

-

Extensions

- Connector

Generalizations

- ADLHwConnector

Properties

- port : CommunicationHwPort [2]

Semantics

The SystemBusConnector represents a physical bus. Depending on modeling style, a single line may be used to represent the bus, or the two physical wires are exposed.

Constraints

No additional constraints

10 EAST-ADL2 extensions for EnvironmentModeling

This section contains the UML-profile specification, specifying stereotypes in the UML-profile, defined from the metaclasses in the EnvironmentModeling package. It includes specification details for each stereotype. If the stereotype has properties, which may be referred to as tag definitions, or if the stereotype has constraints, this section also includes specification details for these properties and constraints.

10.1 «ADLContFunctionType» (from EnvironmentModeling)

Extensions

- Class

Generalizations

- ADLFunctionType

Properties

- behavior : ADLBehavior [0..1]
- EnvironmentModel : EnvironmentModel [1]
- hazard : Hazard [0..*]
- peripheral : Element [1]

- sensorActuator : Element [1]

Semantics

TBD

Constraints

No additional constraints

10.2 «EnvironmentModel» (from EnvironmentModeling)

Description

The EnvironmentModel metaclass defines the ADL entity used to describe the environment of the vehicle electric and electronic architecture. A dedicated entity is used, however, to clearly separate the plant from the control system, i.e. the vehicle from its electronics. The interaction between EnvironmentModel and the ADLFunctions are through the FunctionalDevices: Ports of the FunctionalDevices are connected to the ADLFunctions on one side, and to the EnvironmentModel on the other side.

Extensions

- Class

Generalizations

- ADLFunctionType

Properties

- environmentFunction : ADLContFunctionType [0..*]

Semantics

The EnvironmentModel represents the environment to the E/E architecture that is defined by the FAA, FDA, etc.

Constraints

No additional constraints

Part III Behavioral Constructs

11 EAST-ADL2 extensions for Behavior

This section contains the UML-profile specification, specifying stereotypes in the UML-profile, defined from the metaclasses in the Behavior package. It includes specification details for each stereotype. If the stereotype has properties, which may be referred to as tag definitions, or if the stereotype has constraints, this section also includes specification details for these properties and constraints.

11.1 «ADLAction» (from Behavior)

Extensions

None

Generalizations

None

Properties

- port : ADLFlowPort [0..*]

Semantics

TBD

Constraints

No additional constraints

11.2 «ADLActivityParameterNode» (from Behavior)

Extensions

None

Generalizations

None

Properties

- port : ADLFlowPort [0..*]

Semantics

TBD

Constraints

No additional constraints

11.3 «ADLBehavior» (from Behavior) {abstract}

Description

ADLBehavior is an abstract entity used to represent both Native and External behavior.

Extensions

None

Generalizations

None

Properties

No additional properties

Semantics

TBD

Constraints

No additional constraints

11.4 «ADLBehaviorMapping» (from Behavior)

Extensions

- Class

Generalizations

- ADLRelationship

Properties

- base_Class : Class [1]
- runnableEntity : Element [1]

Semantics

TBD

Constraints

No additional constraints

11.5 «ADLUseCase» (from Behavior)

Description

-

Extensions

None

Generalizations

- ADLTraceableSpecification

Properties

No additional properties

Semantics

TBD

Constraints

No additional constraints

11.6 «ExternalBehavior» (from Behavior)

Description

The ExternalBehavior is a specialization of ADLBehavior. It represents behavior defined in an external tool, such as Simulink or Statemate. It is merely a placeholder with the purpose of containing information about and links to the external behavioral model.

Extensions

- Class
- Behavior

Generalizations

- ADLBehavior

Properties

- base_Behavior : Behavior [1]
- base_Class : Class [1]
- path : String [1] The path to the file or model entity containing the ExternalBehavior.
- representation :
ExternalBehaviorRepresentationKind = SDL [1]

Semantics

The contents of the ExternalBehavior represent the behavior of the owning entity (association Owner). The internal behavioral semantics of ExternalBehavior follows the semantics of the behavioral representation used (attribute Representation). Externally, In relation to the EAST-ADL2model, however, the ExternalBehavior has synchronous execution semantics:

1. Read inputs from input ports
2. Execute ExternalBehavior with fixed inputs (run to completion)
3. Provide outputs to output ports

The data transfer between the EAST-ADL2 ports and the ExternalBehavior is representation specific and considered part of the execution of the ExternalBehavior.

Constraints

No additional constraints

11.7 ExternalBehaviorRepresentationKind (from Behavior)

Description

Enumeration of representation for ExternalBehavior.

Generalizations

None

Enumeration Literals

- ASCET
- SDL
- SIMULINK
- STATEMATE

Semantics

TBD

Constraints

No additional constraints

11.8 «NativeBehavior» (from Behavior)

Description

NativeBehavior is a specialization of ADLBehavior and the UML2 StateMachine. It is a placeholder for a Behavioral definition that is recognized by EAST-ADL2 compliant tools. The internal behavioral semantics and further integration of UML2 state machines in EAST-ADL2 are not defined.

Extensions

- Class
- Behavior

Generalizations

- ADLBehavior

Properties

- base_Behavior : Behavior [1]
- base_Class : Class [1]

Semantics

The contents of the NativeBehavior metaclass represent the behavior of the owning entity (association Owner). The internal behavioral semantics of NativeBehavior follows the semantics of the behavioral representation used (attribute Representation). In the EAST-ADL2 model, however, the NativeBehavior has synchronous execution semantics:

1. Read inputs from input ports
2. Execute ExternalBehavior with fixed inputs (run to completion)
3. Provide outputs to output ports

The data transfer between the EAST-ADL2 ports and the NativeBehavior is representation specific and considered part of the execution of the NativeBehavior

The behavioral classes determine the control algorithm by combining the operations of the attribute classes in an appropriate manner. Behavioral classes might furthermore aggregate other behavioral classes to describe hierarchies in data-flow graphs.

Typical attribute classes are state-variables and parameters of a control-algorithm. Sequencing of operations within a control-algorithm is specified by means of interaction diagrams.

Refining the attribute classes and associating them to refinements of behavioral classes allow the construction of implementation variants of a control algorithm with respect to the chosen quantization, scalings, non-linearities imposed by limited values.

Constraints

No additional constraints

12 EAST-ADL2 extensions for ErrorBehavior

This section contains the UML-profile specification, specifying stereotypes in the UML-profile, defined from the metaclasses in the ErrorBehavior package. It includes specification details for each stereotype. If the stereotype has properties, which may be referred to as tag definitions, or if the stereotype has constraints, this section also includes specification details for these properties and constraints.

12.1 «ErrorBehavior» (from ErrorBehavior)

Description

<ErrorBehavior> represents possible failure semantics of an ADLEntity or an ARElement. Each ErrorBehavior has a specification of error logics, capturing the temporal or causal relationships of events and failure modes (currently treated as String) in errorPropagationPort.

Extensions

- Class

Generalizations

None

Properties

- base_Class : Class [1]
- failureLogic : String = "NA" [1]
- genericDescription : String = "NA" [1]
- handled : Boolean [1]
- inADLErrorModel : ErrorModel [1..*]
- propagationPort : PropagationPort [1..*]
- targetARType : NamedElement [0..1]

Semantics

TBD

Constraints

[1] A particular ErrorBehavior metaclass targets either an instance of ADLEntity or ARElement.

[2] If a particular ErrorBehavior metaclass targets ADLEntity, the target should be of the following types: ADLFunctionType and HardwareArchitecture. (ImplementationArchitecture is not included due to analysis considerations.) Otherwise, the target should be of _HWElement

OCL:

context ErrorBehavior

inv: (self.targetADLType->isEmpty xor self.targetARType->isEmpty)

inv: if (self.targetADLType->notEmpty) then

(self.targetADLType.oclIsKindOf(ADLFunctionType) or
self.targetADLType.oclIsKindOf(HardwareArchitecture)) else

(self.targetARType.oclIsKindOf(_HWElement))

endif

No additional constraints specified in profile.

12.2 «ErrorModel» (from ErrorBehavior)

Description

<ErrorModel> defines the set of containedErrorBehaviors (1..*) that an ADLEntity or ARElement in a

particular context (i.e., system configuration) can have. One ErrorModel is normally applied for one ADLEntity or ARElement. When more than one ErrorModels are defined for an ADLEntity or ARElement, the EAST-ADL2 variability mechanism can be used for determining the choices for different analysis cases.

Extensions

- Property

Generalizations

None

Properties

- adlErrorContext : ADLEntity [0..1]
- arErrorContext : NamedElement [0..1]
- base_Property : Property [1]
- containedErrorBehavior : ErrorBehavior [1..*]

Semantics

TBD

Constraints

[1] The ErrorModel metaclass has either an ADLEntity or an ARElement as the context.

[2] If an ErrorModel metaclass has an adlErrorContext, the context should be the same ADLEntity as the context of its targetADLPart (through ErrorModelToTarget). Otherwise, the context is the same as the context of its targetARElement (through ErrorModelToTarget), which can be either an ADLEntity or an ARElement (e.g., HWElementContainer)

OCL:

context ErrorModel

inv: (self.adlErrorContext->isEmpty xor self.arErrorContext->isEmpty)

inv: if (self.adlErrorContext->notEmpty)

then (self.ErrorModelToTarget.targetADLPart = self.adlErrorContext.part) else
(self.ErrorModelToTarget.targetARElement = self.adlErrorContext.part or
self.ErrorModelToTarget.targetARElement=self.arErrorContext.part)

endif

No additional constraints specified in profile.

12.3 «ErrorModelToTarget» (from ErrorBehavior)

Description

Indicates the target of the ErrorModel.

Extensions

- Dependency

Generalizations

None

Properties

- attachedADLErrorModel : ErrorModel [1]
- base_Dependency : Dependency [1]
- targetADLPart : ADLEntity [0..1]
- targetARElement : NamedElement [0..1]

Semantics

TBD

Constraints

[1] The ErrorModelToTarget metaclass points either to an ADLEntity or to an ARElement.

[2] If an ErrorModelToTarget metaclass targets an ADLEntity, then the target is an ADLFunctionPrototype of which the type is the target of the contained errorBehavior by the attachedADLErrorModel. Alternatively, the target is a ARElement of which the type is the target of the contained errorBehavior by the attachedADLErrorModel.

in OCL:

```
context ErrorModelToTarget
```

```
inv: (self.targetADLPart->isEmpty xor self.targetARElement->isEmpty)
```

```
inv: if (self.targetADLPart->notEmpty) then
```

```
(self.targetADLPart.ocIsKindOf(ADLFunctionPrototype) and
```

```
self.attachedADLErrorModel.containedErrorBehavior.targetADLType =self.targetADLPart.type) else
```

```
(self.targetARElement.ocIsKindOf(HWElement) and
```

```
self.attachedADLErrorModel.containedErrorBehavior.targetARType =self.targetARElement.type)
```

```
endif
```

No additional constraints specified in profile.

12.4 «ErrorPropagation» (from ErrorBehavior)

Description

The ErrorPropagation metaclass is used to capture the propagations of errors of ADLEntity and ARElement (hardware only in current solution). A propagation can be based on: ADLEntity, ARElement (only for HW-HW propagation), or ADLRealization (for cross level propagations from SW to Func, HW to Func, or HW to SW).

The ErrorPropagation metaclass has a propagationLogic attribute for capturing the logic and temporal relationships of error propagations

Extensions

- Dependency
- Connector

Generalizations

None

Properties

- base_Connector : Connector [1]
- base_Dependency : Dependency [1]
- from : PropagationPort [1..*]
- otherRelatedPropagations : ErrorPropagation [0..*]
- propagationLogic : String [1]
- throughADLEntity : ADLEntity [0..1]
- throughARElement : NamedElement [0..1]
- to : PropagationPort [1..*]

Semantics

TBD

Constraints

The following constraints are currently applied:

- [1] A particular ErrorPropagation metaclass should connect either from OUT propagationPort to IN propagationPort, or from IN propagationPort to IN propagationPort (in the case of delegation),
- [2] A particular ErrorPropagation metaclass should not have relationship to itself.
- [3] A particular ErrorPropagation metaclass should have one propagation medium (e.g., either throughADLEntity, throughARElement, or throughADLRealization)
- [4] The throughARElement should only be chosen when an ErrorPropagation metaclass is between two hardware components of ARElement. The throughARElement should not be one of the connected ARElements.
- [5] The throughADLEntity should only be chosen when an ErrorPropagation metaclass is between two ADLEntity. The throughADLEntity should not be one of the connected ADLEntity. The throughADLEntity should be an ADLFunctionPrototype or an ADLConnectorPrototype.

OCL:

context ErrorPropagation

inv: not self.to.direction = OUT and self.from.direction=OUT

inv: not self.otherRelatedPropagations = self

inv: (self.throughADLEntity->isEmpty xor self.throughARElement->isEmpty) xor self.throughADLRealization->isEmpty

inv: (self.throughARElement->notEmpty) implies (self.from.portOwner.targetARType->notEmpty and self.to.portOwner.targetARType->notEmpty) and (self.throughADLEntity <> self.to.portOwner.targetARType and self.throughARElement <> self.from.portOwner.targetARElement)

inv: (self.throughADLEntity->notEmpty) implies (self.from.portOwner.targetADLType->notEmpty and self.to.portOwner.targetADLType->notEmpty) and (self.throughADLEntity <> self.to.portOwner.targetADLType and self.throughADLEntity <> self.from.portOwner.targetADLType) and (self.throughADLEntity.oclIsKindOf(ADLFunctionPrototype) or self.throughADLEntity.oclIsKindOf(ADLConnectorPrototype))

No additional constraints specified in profile.

12.5 «ErrorToHazard» (from ErrorBehavior)

Description

Error depends on Hazard.

Extensions

- Dependency

Generalizations

None

Properties

- base_Dependency : Dependency [1]
- failureEvent : PropagationPort [0..*]
- hazard : Hazard [0..*]

Semantics

TBD

Constraints

No additional constraints

12.6 «Hazard» (from ErrorBehavior)

Description

The Hazard represent a condition or state in the system that may contribute to accidents. It is usually a failure of some kind, but may also be a result of nominal operation. The Hazard metaclass is contained in the context, as Hazard specializes ADLTraceableSpecification. The context describes the element of the system where this hazard occur.

Extensions

- Class

Generalizations

- ADLEntity
- ADLTraceableSpecification

Properties

- base_Class : Class [1]
- controllability : String [1]
- environmentFunction : ADLContFunctionType [0..*]
- exposure : String [1]
- severity : String [1]

Semantics

TBD

Constraints

No additional constraints

12.7 PropagationDirection (from ErrorBehavior)

Generalizations

None

Enumeration Literals

- IN
- OUT

Semantics

TBD

Constraints

No additional constraints

12.8 «PropagationPort» (from ErrorBehavior)

Description

The propagationPort metaclass represents the ports through which an external/internal error event propagates IN/OUT. The OUT propagation is also used to derive Hazards.

Extensions

- Class
- Port

Generalizations

None

Properties

- base_Class : Class [1]

- base_Port : Port [1]
- direction : PropagationDirection [0..1]
- event : String [1]
- portOwner : ErrorBehavior [1]

Semantics

TBD

Constraints

No additional constraints

Part IV Cross-Cutting Constructs

13 EAST-ADL2 extensions for Requirements

This section contains the UML-profile specification, specifying stereotypes in the UML-profile, defined from the metaclasses in the Requirements package. It includes specification details for each stereotype. If the stereotype has properties, which may be referred to as tag definitions, or if the stereotype has constraints, this section also includes specification details for these properties and constraints.

13.1 «ADLRequirement» (from Requirements)

Description

The ADLRequirement will be used to unite the common properties of specific requirement types.

The ArchivedEntity allows any ADLRequirement to be versioned and archived. Each version will carry information like author, creation date etc.

An ADLRequirement may either be directly contained in an ADLContext (by inheriting from ADLTraceableSpecification) or it may be contained in an ADLRequirementContainer, which represents a larger unit or module of specification information. The first possibility is used in trivial cases or often for specialized requirements.

The traceability between ADL requirement entities and other specification or design entities will be ensured by the relationship dependencies described earlier in the ADLCoreConstructs part of this specification.

Extensions

- Comment
- Class

Generalizations

- ArchivedEntity
- ADLTraceableSpecification
- SYSML::Requirement

Properties

- applicability : String [0..1]
- representation : RequirementDescription [0..*]

Semantics

The ADLRequirement metaclass applies to the ADLEntity that is associated to the ADLRequirement through the ADLSatisfy relation.

The optional Applicability attribute defines the conditions under which the requirement should hold.

Constraints

No additional constraints

13.2 «ADLRequirementContainer» (from Requirements)

Description

The ADLRequirementContainer metaclass is used to aggregate several ADLRequirements which are semantically related to each other. It can be seen as a larger unit or module of specification information.

By inheriting from ADLTraceableSpecification, an ADLRequirementContainer may itself be contained in an ADLContext.

In addition to aggregating related ADLRequirements, the ADLRequirementContainer allows to define relations between its contained requirements and also a grouping of them.

Furthermore, the ADLRequirementContainer allows to introduce additional user attribute definitions by way of UAElementTypes or UATemplates which are valid only locally inside this ADLRequirementContainer. These are additional in that they are used in addition to the user attribute definitions which are provided globally for the entire EAST-ADL2 repository.

Extensions

- Class

Generalizations

- ADLTraceableSpecification

Properties

- base_Class : Class [1]
- containedTraceableSpecification : ADLTraceableSpecification [0..*] { ordered }

Semantics

TBD

Constraints

No additional constraints

13.3 «AllocateableElement» (from Requirements) {abstract}

Description

The AllocateableElement is a superclass for ADL entities that are allocateable.

Extensions

None

Generalizations

None

Properties

No additional properties

Semantics

TBD

Constraints

No additional constraints

13.4 «AllocationConstraint» (from Requirements)

Description

The AllocationConstraint metaclass denotes the requirement language relationship that may be used to constrain the allocation of either a Feature or an ADLFunction in a target ECU.

The same constraint could be expressed in a textual design constraint.

Extensions

- Comment
- Class

Generalizations

- DesignConstraint

Properties

- allocatedAutosarComponent : Element [1]
- allocatedElement : AllocateableElement [1] The entity to be allocated.
- target_ECU : Element [1]

Semantics

TBD

Constraints

[1] Either an ADL requirement on an AllocateableElement or on an AUTOSAR ComponentType.

No additional constraints specified in profile.

13.5 «DelayRequirement» (from Requirements)

Description

The DelayRequirement metaclass is a language entity that constrains the delay between two or more events. It is used for defining requirement on end-to-end delays, periods and synchrony.

Target events are primarily ports.

The applicability attribute, which is inherited from ADLRequirement is a definition of the condition under which the requirement must hold. Can be used to define several timing requirements for the same measure.

Extensions

- Comment
- Class

Generalizations

- QualityRequirement
- TimingRestriction

Properties

- from : ADLEntity [0..*] This is the port where a signal arrival or writing marks the beginning of computation for the end to end delay.
- probability : Bound [0..1]
- reference : ADLEntity [0..*]
- until : ADLEntity [0..*] This is the port where a signal arrival or writing marks the ending of computation for the end to end delay.

Semantics

Delay 1..* From 1..* Until

Time from first From event to last Until event

Delay From Inport / Until Outport corresponds to the time from when event ha occurred on inport until even occurs on outport

Delay From Inport / Until Inport corresponds to the time from when event ha occurred on From inport until

event occurs on the Until Inport

Delay From Output / Until Inport corresponds to the time from when event ha occurred on From Output until event occurs on the Until Inport. If referred ports are directly connected, this is a data transfer delay.

Join 1 reference 2..* From 0 Until

Time from first From event to last From event. Reference identifies the event that is the result of the From events.

Fork1 Reference 0 From 2..* Until

Time from first Until event to last Until event. Reference identifies the event that is the origin of the Until events.

Rate: 1 From and 1 Until association associated to the same entity

Time between consecutive event occurrences

Semantics for the target entity

Outport:

An Until association to an Outport refers to the event

A From association to an Outport simply refers to the event on that port

A Reference association is not applicable

Inport:

An Until association to an Inport means that the event is available to the receiving ADLFunction

A From association to an Inport simply refers to the event on that port

A Reference association is not applicable

ADLFunction:

From and Until associations both refers to the triggering event of the ADLFunction

A From association

A reference association

Constraints

[1] From, Until and Reference should reference ports only, From may reference ADLFunction, end the AUTOSAR entities Runnable, AtomicSoftwareComponent.

[2] Multiplicities in the associations for From, Until and Reference must comply with the semantics of DelayRequirement, see OCL below:

context DelayRequirement inv:

((self.from->size())>=1) and (self.until->size())>=1) and (self.reference->size()=0)) or

((self.from->size()=0) and (self.until->size())>2) and (self.reference->size()=1)) or

((self.from->size())>=2) and (self.until->size()=0) and (self.reference->size()=1)) or

((self.from->size()=1) and (self.until->size()=1) and (self.reference->size()=0))

No additional constraints specified in profile.

13.6 «DesignConstraint» (from Requirements)

Description

The DesignConstraint metaclass denotes the ADL entity that is used to constrain the solution space of Feature realizations.

The attribute type allows a more specific classification.

Extensions

- Comment
- Class

Generalizations

- ADLRequirement

Properties

- designConstraintKind : DesignConstraintKind [0..1]

Semantics

TBD

Constraints

No additional constraints

13.7 *DesignConstraintKind (from Requirements)*

Description

-

Generalizations

None

Enumeration Literals

- Allocation
- Cost
- Legacy
- Other
- Physical
- PowerConsumption
- Process
- ReferenceArchitecture
- Resource
- Reuse
- Standard

Semantics

TBD

Constraints

No additional constraints

13.8 «*FunctionalRequirement*» (from Requirements)

Description

The FunctionalRequirement metaclass denotes the requirement language entity used to refine a Feature requirement by specifying what the Feature is required to do.

It focuses on internal properties of the Feature rather than interaction with other actors.

The relationship used to link a FunctionalRequirement with a Feature is the ADL Satisfy relationship.

Extensions

- Comment

- Class

Generalizations

- ADLRequirement

Properties

No additional properties

Semantics

TBD

Constraints

No additional constraints

13.9 «PrecedenceConstraint» (from Requirements)

Description

-

Extensions

- Dependency
- Comment
- Class

Generalizations

- DesignConstraint

Properties

- base_Dependency : Dependency [1]
- preceding : ADLStructurePrototype [1]
- successive : ADLStructurePrototype [1..*] { ordered }

Semantics

The semantics for the PrecedenceConstraint metamodel class is to define an association relationship in-between ADLFunctions, indicating the association relationship such that all predecessors have completed before the successors are started.

Note: Without a precedence relation, ADLFunctions are executed according to its data dependencies, if these are uni-directional. For bi-directional data dependencies, execution order is not defined unless the Precedence Dependency relationship is used.

Constraints

No additional constraints

13.10 «QualityRequirement» (from Requirements)

Description

The Quality Requirement metaclass denotes the ADL entity that is used to introduce externally visible property of the system considered as whole. The system could be the entire vehicle or its subsystem.

The QualityRequirement is a stereotyped element, which extends the Requirement stereotype.

The attribute type allows a more specific classification.

Extensions

- Comment
- Class

Generalizations

- ADLRequirement

Properties

- qualityRequirementKind : QualityRequirementKind [1]

Semantics

TBD

Constraints

No additional constraints

13.11 QualityRequirementKind (from Requirements)

Description

-

Generalizations

None

Enumeration Literals

- Configurability
- Dependability
- Ergonomy
- HMI
- Other
- Performance
- Safety
- Security

Semantics

TBD

Constraints

No additional constraints

13.12 «RequirementDescription» (from Requirements)

Extensions

- Comment

Generalizations

None

Properties

- base_Comment : Comment [1]
- formalism : String [1]
- representedRequirement : ADLRequirement [0..1]
- url : String [1]

Semantics

TBD

Constraints

No additional constraints

13.13 «SafetyRequirement» (from Requirements)

Description

The Safety Requirement class denotes the ADL entity that is used to introduce safety related requirements. ISO26262 WD

Extensions

- Comment
- Class

Generalizations

- QualityRequirement

Properties

- derivedFromHazard : Hazard [0..*]
- SIL : SILLevelKind [1]

Semantics

TBD

Constraints

No additional constraints

13.14 SILLevelKind (from Requirements)

Generalizations

None

Enumeration Literals

- SILLevelKind1
- SILLevelKind2
- SILLevelKind3
- SILLevelKind4

Semantics

TBD

Constraints

No additional constraints

13.15 «TimingRestriction» (from Requirements) {abstract}

Description

The TimingRestriction metaclass denotes a requirement entity that allows giving bounds on system timing attributes, i.e. end-to-end delays, periods, etc.

A TimingRestriction can specify one or several of an upper bound, a lower bound or a nominal value. The bound will be measured in a given Unit.

Extensions

None

Generalizations

None

Properties

- jitter : Bound [1] This is the maximum allowed run-time variation for a value within the allowed range.

- lower : Bound [1] Minimum allowed value.
- nominal : Bound [1] Nominal, or target value.
- upper : Bound [1] Maximum allowed value.

Semantics

Upper and Lower constrain the allowed values of the measure

Jitter constrains the allowed variation of the measure

Nominal is the typical or desired value useful as target measure for design

Example:

A requirement of Lower=5 Upper 10 jitter 2 means that actual measures of

Maximum measure of 7.5 with measured variations 2.5 is not allowed because the jitter requirement is violated

Maximum measure of 6 with measured variations 1.5 is not allowed because the lower requirement is violated

Maximum measure of 9 with measured variations 1.5 is not allowed because the upper requirement is violated

Constraints

No additional constraints

14 EAST-ADL2 extensions for ArtifactLevelVariationManagement

This section contains the UML-profile specification, specifying stereotypes in the UML-profile, defined from the metaclasses in the ArtifactLevelVariationManagement package. It includes specification details for each stereotype. If the stereotype has properties, which may be referred to as tag definitions, or if the stereotype has constraints, this section also includes specification details for these properties and constraints.

14.1 «CombinationDecision» (from ArtifactLevelVariationManagement)

Description

Every CombinationDecision metaclass is linked to a unique VariationGroup i.e. (VariationPointConfiguration or DependencyConstraint) . Hence each decision leads to a specific variability choice for a group of variable elements.

Extensions

- DecisionNode

Generalizations

None

Properties

- base_DecisionNode : DecisionNode [1]
- choiceFor : VariationGroup [1]
- owner : DecisionTree [0..1]
- resolution : Resolution [1..*]

Semantics

In order to derive the system family, several choices are proposed to the user.

Constraints

No additional constraints

14.2 «DecisionTree» (from ArtifactLevelVariationManagement)

Description

This a tree of choices and actions to do in order to obtain a specific product from the system family.

Extensions

- Activity

Generalizations

None

Properties

- base_Activity : Activity [1]
- owned : CombinationDecision [0..*]

Semantics

The decision tree is generated from the model containing VariationPointConfiguration or DependencyConstraint. It is used in order to verify the adequacy of the artifact level with the VFM in terms of products generation from the system family.

Constraints

No additional constraints

14.3 «DependencyConstraint» (from ArtifactLevelVariationManagement)

Description

The DependencyConstraint metaclass expresses various constraining dependencies between ADLVariableElements.

Extensions

- Dependency

Generalizations

- VariationGroup

Properties

- base_Dependency : Dependency [1]
- source : ADLFunctionType [1..*] Refine ADLElement
(not to be an property in the profile, already exist in dependency element)
- target : ADLFunctionType [1..*] Refine ADLElement
(not to be an property in the profile, already exist in dependency element)

Semantics

The DependencyConstraint gives a constraint between ADLVariableElements. Upon the selection of a source element the selection of the target element is influenced.

Constraints

No additional constraints

14.4 «EffectNode» (from ArtifactLevelVariationManagement)

Description

The EffectNode metaclass contains a set of actions in order to obtain a product from a system family model.

The action may be written in ATL or QVT.

Extensions

- OpaqueAction

Generalizations

None

Properties

- base_OpaqueAction : OpaqueAction [1]
- body : String [1]
- languageKind : String [1]
- resolution : Resolution [1..*]

Semantics

This is an action that modifies the model in order to obtain the model without variation.

Constraints

No additional constraints

14.5 «Resolution» (from ArtifactLevelVariationManagement)

Description

Each decision node proposes a set of choice to do. Each resolution is one of these choices.

Extensions

- ControlFlow

Generalizations

None

Properties

- base_ControlFlow : ControlFlow [1]
- choice : CombinationDecision [0..*]
- triggered : EffectNode [0..*]

Semantics

This is one choice that comes from a decision node.

Constraints

No additional constraints

14.6 «ReuseMetaInformation» (from ArtifactLevelVariationManagement)

Description

The reuse meta-information is used to describe information needed in the context of reuse. For example a specific entity is only a short-time solution that is not intended to be reused. Also a specific entity can only be reused for specific model ranges (that are not reflected in the product model). This kind of information

can be stored in this information.

Extensions

- Class

Generalizations

None

Properties

- base_Class : Class [1]
- information : String [1] The reuse information is stored in this attribute.
- isReusable : Boolean = true [1] This Boolean attributes just says if the entity itself can essentially be reused or not. Specific information or constraints on reuse are in the information attribute.
- reusableElement : ADLVariableElement [1]

Semantics

This is an information that explains if and how the respective entity can be reused.

Constraints

No additional constraints

14.7 «VariantLink» (from ArtifactLevelVariationManagement)

Description

The VariantLink metaclass denotes a link between a variation point and all its contained variantElements. When a variantElement is selected only the variation point is replaced by it.

Extensions

- Dependency

Generalizations

None

Properties

- base_Dependency : Dependency [1]
- variantElement : ADLVariableElement [1]
- variationPoint : VariationPoint [1]

Semantics

This link is used to express that a variant element may replace the variation point element.

Constraints

No additional constraints

14.8 «VariationGroup» (from ArtifactLevelVariationManagement) {abstract}

Description

A variation group belongs to an arbitrary set of variable elements.

It is used to express constraints between arbitrary variable elements.

Extensions

None

Generalizations

- ADLRelationship

Properties

- constraint : String [1] Only defined iff kind==customizedCombination.
- decisionNode :
CombinationDecision
[0..*]
- kind :
VariationGroupKind =
exclude [1]
- motivation : String [1] Each grouping of variable elements together with the kind of variability has a motivation.

Example: Imagine you have a watch family with optional timer functionality. The motivation for the associated variation group is: Do you want to have a timer in your watch?

In the decision model the user can then determine the decisions based on the motivation questions.

- next : VariationGroup
[0..*]
- previous :
VariationGroup [0..*]
- variableElement :
ADLVariableElement
[1..*]

Semantics

If the type is :

-exclude it means that exactly one element of the group must be selected in the configuration.

-include means that the selection of element A results in the selection of element B (while A is ordered before B in the ordered link between variationGroup and variable element). Therefore this relation is not bidirectional (cf. logical implication).

-customized combination means that the user can write an own combination. This combination is written in the constraint property. this combination is written by using logical operators.

The property motivation explains why this constraint is added in the model. It is written in literals.

Each grouping of variable elements together with the kind of variability has a motivation.

Example: Imagine you have a watch family with optional timer functionality. The motivation for the associated variation group is: Do you want to have a timer in your watch?

In the decision model the user can then determine the decisions based on the motivation questions.

The next and previous attribute are used to constraint the number of decision models. This is the sequence of choice.

Constraints

No additional constraints

14.9 *VariationGroupKind* (from *ArtifactLevelVariationManagement*)

Description

A variation group may be any one of the types defined in the *VariationGroupKind* enumeration metaclass.

Generalizations

None

Enumeration Literals

- *customizedCombination* Used to express an OCL as an own dependency constraint.
Attribute constraint (from *VariationGroup*) is used to express the OCL constraint.
- *exclude*
- *include*

Semantics

This is type of constraint that can be associated to a *VariationGroup*.

Constraints

No additional constraints

14.10 «*VariationPoint*» (from *ArtifactLevelVariationManagement*)

Description

The *VariationPoint* abstract metaclass can be an arbitrary structural entity in the model being a placeholder for all its contained variants.

Also the variation point can be used to extend the EAST-ADL variability approach to other languages and standards by adding a generalize relation from the respective (non EAST-ADL) element to the variation point, by that inheriting the placeholder property (*Extension Point*).

Extensions

- *Class*

Generalizations

- *ADLFunctionType*

Properties

- *vpConfiguration* : *VariationPointConfiguration* [1]

Semantics

This is an abstract class providing needed variation point placeholder property for concrete classes refining this class.

Constraints

No additional constraints

Notation

There is no general notation for *VariationPoint*. Further subclasses of *VariationPoint* will define their own notation.

Icon: Serialized

14.11 «VariationPointConfiguration» (from ArtifactLevelVariationManagement)

Description

The VariationPointConfiguration metaclass is a specific variation group that is used in the context of variation points. The kind is exclude and the selectionCriterion property is an expression to explain the case where the variableElement is chosen (written in ATL or QVT). Therefore the attribute constraint (from VariationGroup) is not filled.

Extensions

- Class

Generalizations

- VariationGroup

Properties

- actualBindingTime : BindingTimeAttribute [1]
- base_Class : Class [1]
- isOptional : Boolean [1]
- requiredBindingTime : BindingTimeAttribute [0..1]
- selectionCriterion : String [1]
- variationPoint : VariationPoint [1]

Semantics

This is a constraint that is associated to a variation point in order to express that we can choose only one variant and the condition to choose it.

Constraints

No additional constraints

15 EAST-ADL2 extensions for Support

This section contains the UML-profile specification, specifying stereotypes in the UML-profile, defined from the metaclasses in the Support package. It includes specification details for each stereotype. If the stereotype has properties, which may be referred to as tag definitions, or if the stereotype has constraints, this section also includes specification details for these properties and constraints.

15.1 «ArchivedEntity» (from Support) {abstract}

Description

The ArchivedEntity metaclass marks instances of other ADL entities as elements for which a version management will be established.

Extensions

- Comment

Generalizations

- ADLEntity

Properties

- base_Comment : Comment [1]
- versionArchive : VersionArchive [0..1]

Semantics

TBD

Constraints

No additional constraints

15.2 «*Configuration*» (from Support)

Description

The Configuration metaclass denotes an ADL entity used for creation of a model built up of certain versions of archived entities

Extensions

- Comment

Generalizations

- ADLEntity

Properties

- base_Comment : Comment [1]
- ownedVersions : Version [0..*]

Semantics

TBD

Constraints

No additional constraints

15.3 «*Version*» (from Support)

Description

The Version metaclass denotes an ADL entity representing a snapshot of the state of an Archived Entity at a certain point in time. Will be used to track different states of an Archived Entity during the development process.

The creation of a new version has to be initiated by the user.

Extensions

- Comment

Generalizations

- ADLEntity

Properties

- base_Comment : Comment [1]

Semantics

TBD

Constraints

No additional constraints

15.4 «*VersionArchive*» (from Support)

Description

The VersionArchive metaclass is used to collect different versions of an Archived Entity. For an ADL entity marked as Archived Entity, one Version Archive will be created. If a new Version of an Archived Entity is created it will be included in the corresponding Archive.

Extensions

- Comment

Generalizations

- ADLEntity

Properties

- archivedEntity : ArchivedEntity [0..1]
- base_Comment : Comment [1]
- ownedVersions : Version [0..*]

Semantics

TBD

Constraints

No additional constraints

16 EAST-ADL2 extensions for VerificationValidation

This section contains the UML-profile specification, specifying stereotypes in the UML-profile, defined from the metaclasses in the VerificationValidation package. It includes specification details for each stereotype. If the stereotype has properties, which may be referred to as tag definitions, or if the stereotype has constraints, this section also includes specification details for these properties and constraints.

16.1 «AbstractVVCASE» (from VerificationValidation)

Description

A VVCASE is the core concept of verification and validation support in EAST-ADL2 and represents a certain verification and validation effort of varying scope and intention. The definition of VVCASEs is separated in two levels: an abstract and a concrete level, represented by the entities AbstractVVCASE and ConcreteVVCASE.

An AbstractVVCASE describes a verification and validation effort on an abstract level, i.e. without referring to a concrete testing environment and without specifying stimuli and the expected outcome on a concrete technical level.

Slightly simplified one could say: An AbstractVVCASE describes "what" needs to be done in the context of a certain verification and validation effort, but not precisely "how" this is done. Together, all AbstractVVCASEs for a certain system form sort of a "to-do"-list, which describes what needs to be done when actually testing the system with a concrete testing environment, but in a form applicable to all conceivable testing environments.

Extensions

None

Generalizations

- ADLTraceableSpecification

Properties

- description : String [1]
- environmentSpecific : ConcreteVVPProcedure [0..*]

Semantics

TBD

Constraints

No additional constraints

16.2 «AbstractVVPProcedure» (from VerificationValidation)

Description

A VVPProcedure represents an individual task in the context of an overall verification and validation effort (represented by a VVCASE), which has to be performed in order to achieve that effort's overall objective. Just as is the case for VVCASEs, the definition of VVPProcedures is separated in two levels: an abstract and a

concrete level, represented by the entities `AbstractVVProcedure` and `ConcreteVVProcedure`.

An `AbstractVVProcedure` defines such a task on an abstract level, i.e. without referring to a concrete testing environment and without specifying stimuli and an expected outcome on a concrete technical level.

Extensions

None

Generalizations

- `ADLTraceableSpecification`

Properties

- `environmentSpecific` : `ConcreteVVCASE` [0..*]

Semantics

TBD

Constraints

No additional constraints

16.3 «ConcreteVVCASE» (from VerificationValidation)

Description

A `VVCASE` is the core concept of verification and validation support in EAST-ADL2 and represents a certain verification and validation effort of varying scope and intention. The definition of `VVCASEs` is separated in two levels: an abstract and a concrete level, represented by the entities `AbstractVVCASE` and `ConcreteVVCASE`.

A `ConcreteVVCASE` describes a verification and validation effort on a concrete level, i.e. it specifies concrete test subjects and targets and provides stimuli and the expected outcome on a concrete technical level.

Slightly simplified one could say: A `ConcreteVVCASE` not only describes "what" needs to be done in the context of a certain verification and validation effort, but also the necessary details of "how" this is done.

Extensions

- `Class`

Generalizations

- `ADLTraceableSpecification`

Properties

- `base_Class` : `Class` [1]
- `vvEnvironment` : `ADLEntity` [0..*]
- `vvProcedure` : `ConcreteVVProcedure` [0..*]
- `vvSubject` : `ADLEntity` [1..*]
- `vvTarget` : `ADLEntity` [0..*]

Semantics

TBD

Constraints

No additional constraints

16.4 «ConcreteVVProcedure» (from VerificationValidation)

Description

A `VVProcedure` represents an individual task in the context of an overall verification and validation effort (represented by a `VVCASE`), which has to be performed in order to achieve that effort's overall objective. Just as is the case for `VVCASEs`, the definition of `VVProcedures` is separated in two levels: an abstract and a

concrete level, represented by the entities `AbstractVVProcedure` and `ConcreteVVProcedure`.

A `ConcreteVVProcedure` defines such a task on a concrete level, i.e. it is defined with a concrete testing environment in mind and provides stimuli and an expected outcome of the procedure in a form which is directly applicable to this testing environment.

Extensions

- `Class`

Generalizations

- `ADLTraceableSpecification`

Properties

- `base_Class` : `Class` [1]
- `vvActualOutcome` : `VVActualOutcome` [0..*]
- `vvIntendedOutcome` : `VVIntendedOutcome` [0..*]
- `vvStimuli` : `VVStimuli` [0..*]

Semantics

TBD

Constraints

No additional constraints

16.5 «VVActualOutcome» (from VerificationValidation)

Description

Actual output of the testing environment represented by `VVTarget` when triggered by the `VVStimuli` of the `ConcreteVVProcedure` which is defined by the association 'performedVVProcedure' of the containing `VVLog`. It should be equivalent to the `VVIntendedOutcome` defined by association 'intendedOutcome'.

Extensions

- `Class`

Generalizations

None

Properties

- `base_Class` : `Class` [1]

Semantics

TBD

Constraints

No additional constraints

16.6 «VVIntendedOutcome» (from VerificationValidation)

Description

Expected output of the testing environment represented by `VVTarget` when triggered by the corresponding `VVStimuli` of the containing `ConcreteVVProcedure`.

Since this entity only occurs on the concrete level (i.e. within the context of a `ConcreteVVCASE`), the output must be provided in a form such that it can directly be compared to the output of the `VVTarget(s)` defined for the containing `ConcreteVVCASE`.

Extensions

- `Class`

Generalizations

None

Properties

- base_Class : Class [1]

Semantics

TBD

Constraints

No additional constraints

16.7 «VVStimuli» (from Verification Validation)

Description

Input values to the testing environment represented by VVTarget in order to perform the corresponding VVProcedure.

Since this entity only occurs on the concrete level (i.e. within the context of a ConcreteVVCASE), the input values must be provided in a form such that they are directly applicable to the VVTarget(s) defined for the containing ConcreteVVCASE.

Extensions

- Class

Generalizations

None

Properties

- base_Class : Class [1]

Semantics

TBD

Constraints

No additional constraints

17 Index

AbstractVVCASE	62, 63
AbstractVVProcedure	62, 63, 64
Actuator.....	12, 29, 30
ADLBehavior.....	20, 36, 38, 39, 40
ADLClientPort	19
ADLClientServerInterface	26
ADLClientServerPort	19, 20, 23, 26
ADLConnectorPrototype	20, 23, 44
ADLContext.....	10, 11, 46, 47
ADLDeriveReq.....	11, 12
ADLDesignDataType	14, 15, 21, 24, 25
ADLDesignDataTypeKind	15
ADLDouble.....	16
ADLEntity 10, 12, 13, 14, 16, 20, 21, 23, 26, 27, 31, 33, 41, 42, 43, 44, 45, 46, 48, 60, 61, 63	
ADLFloat	16
ADLFlowPort	20, 21, 23, 24, 25, 26, 37, 38

ADLFunctionPrototype.....	18, 22, 28, 43, 44
ADLFunctionType.....	17, 18, 19, 20, 21, 22, 23, 24, 27, 28, 29, 31, 33, 36, 37, 41, 55, 59
ADLHwConnector.....	30, 31, 34, 35, 36
ADLHwElement.....	23, 31, 32, 33, 34, 35
ADLHwElementPrototype.....	31, 32
ADLHwPort.....	31, 32, 34, 35
ADLInFlowPort.....	21, 24, 25
ADLInOutFlowPort.....	25
ADLOutFlowPort.....	21, 25
ADLPortGroup.....	26
ADLRealization.....	10, 12, 14, 43
ADLRefine.....	13
ADLRelationship.....	10, 11, 12, 13, 14, 38, 58
ADLRequirement.....	12, 13, 14, 46, 48, 50, 51, 52
ADLRequirementContainer.....	14, 46, 47
ADLSatisfy.....	10, 13, 46
ADLServerPort.....	26
ADLSystemModel.....	16, 17
ADLTraceableSpecification.....	10, 11, 39, 45, 46, 47, 62, 63, 64
ADLUseCase.....	11, 38
ADLVariableElement.....	20, 23, 27, 57, 58
ADLVerify.....	14
AllocateableElement.....	23, 47, 48
AllocationConstraint.....	47
AnalysisArchitecture.....	17
ArchivedEntity.....	46, 60, 62
BindingTimeAttribute.....	60
Bound.....	48, 53, 54
CombinationDecision.....	54, 55, 56, 58
CommunicationHwPort.....	32, 35, 36
ConcreteVVCASE.....	14, 62, 63, 64, 65
ConcreteVVProcedure.....	14, 62, 63, 64
Configuration.....	61
DecisionTree.....	54, 55
DelayRequirement.....	48, 49
DependencyConstraint.....	54, 55
DesignArchitecture.....	17, 18
DesignConstraint.....	48, 49, 51
DesignConstraintKind.....	50
ECU.....	19, 24, 29, 33, 47, 48
EffectNode.....	56
EnvironmentModel.....	9, 17, 28, 36, 37
ErrorBehavior.....	41, 42, 43, 44, 45, 46
ErrorModel.....	41, 42
ErrorModelToTarget.....	42, 43
ErrorPropagation.....	43, 44

ErrorToHazard	44
ExternalBehavior	39, 40
ExternalBehaviorRepresentationKind	39
Feature.....	9, 12, 17, 47, 49, 50
FunctionalAnalysisArchitecture	9, 17, 18, 22, 28, 29
FunctionalDesignArchitecture	9, 17, 18, 29
FunctionalDevice	28, 29
FunctionalRequirement.....	50
FunctionFeatureModel	23
HardwareDesignArchitecture	17, 18, 30, 31, 32, 33, 34, 35, 36
Hazard	36, 44, 45, 53
ImplementationArchitecture	41
IoComponent.....	30, 33, 36
IoHwConnector.....	34
IoHwPort.....	32, 34
LocalDeviceManager	12, 18, 29
MWAbstraction.....	17, 18, 19
NativeBehavior	40
OCL.....	29, 41, 42, 43, 44, 49, 59
PowerHwConnector	34, 35
PowerHwPort.....	35
PowerSupply	35
PrecedenceConstraint.....	51
ProductDecisionModel	23, 24
PropagationPort.....	41, 43, 44, 45
QualityRequirement	48, 51, 53
QualityRequirementKind	52
Resolution	54, 56
ReuseMetaInformation	27, 56
SafetyRequirement.....	53
Sensor.....	12, 29, 35, 36
SystemBusConnector	33, 36
TimingRestriction	29, 48, 53
Trigger.....	20, 29, 30
VariantLink	57
VariationGroup	27, 54, 55, 57, 58, 59, 60
VariationGroupKind	58, 59
VariationPoint	57, 59, 60
VariationPointConfiguration.....	23, 54, 55, 59, 60
VehicleFeature	9
VehicleFeatureModel.....	17, 18, 28
Version	61, 62
VersionArchive	60, 61
VVActualOutcome	64
VVIntendedOutcome	64
VVLog	64

VVStimuli	64, 65
VVTarget	64, 65