Editor Specification

Introduction

This document provides an overview of all diagrams and elements of the EAST-ADL editor. The editor is based on EATOP and contains graphical diagrams implemented with the help of Graphiti.

Intention of the editor is to show the main elements and relationships of EAST-ADL. The structure of EAST-ADL is considered as much as possible. Sometimes this decreases the usability of the tool. But the editor should act as a concept show case to understand the concepts only.
It is not intended to provide a full featured tool with high usability for the use in a series production. Especially it is not in competition to any commercial tool. Instead, with the help of this show case one may create requirements for the commercial tools to improve their usability with respect to the EAST-ADL modeling.

EAST-ADL Editor Specification

1) Features

This chapter introduces the main principles of the editors, i.e. which main features they include and how they behave.

1) Features
This chapter introduces the main principles of the editors, i.e. which main features they include and how they behave.

Features

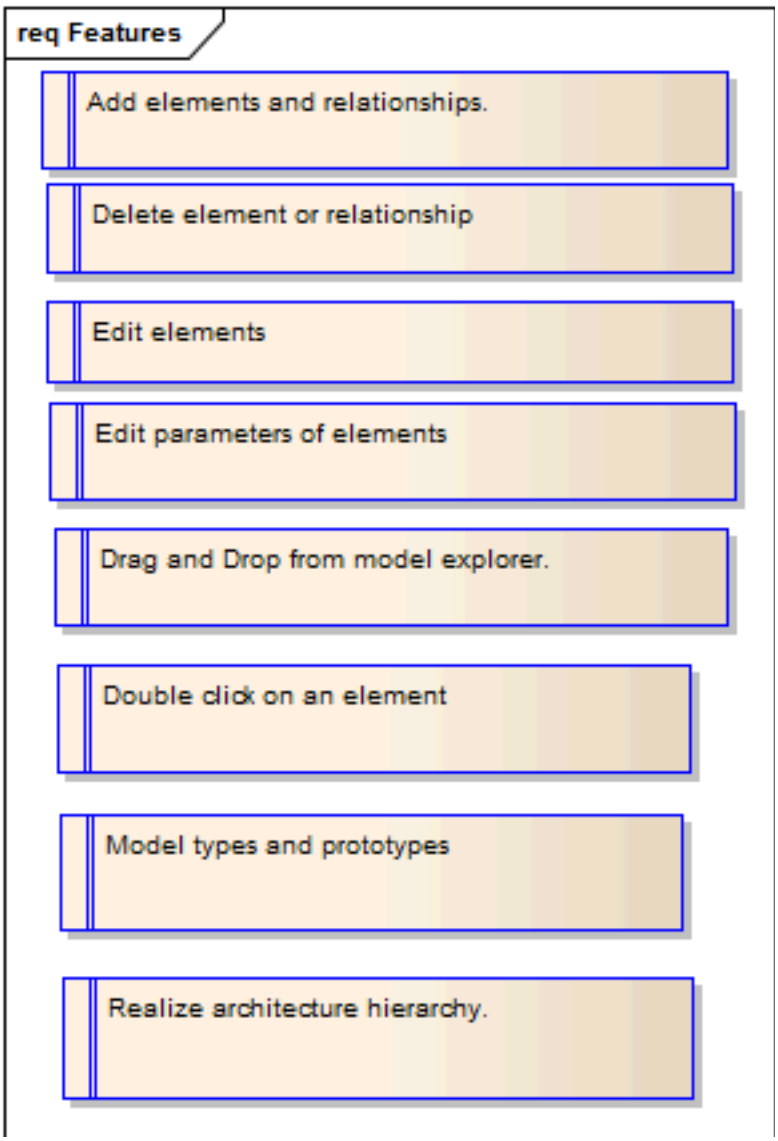

Figure: 1

Add elements and relationships.

Package: 1) Features
Specification:
Elements and relationships from the palette can be added to the diagram. For each element or relationship there is exactly one diagram on which it can be created.
In case one adds an element, a popup window comes up and the short name can be added.

Delete element or relationship

Package: 1) Features
Specification:
Elements and relationships can be hidden and deleted. With the wastebasket symbol the element or relationship is

deleted from the diagram and from the model. With the eraser symbol the element or relationship is deleted from the actual diagram only but still remains in the model.

Double click on an element

Package: 1) Features
Specification:
Double click on an element opens another diagram which enables the editing of the internal of the selected element. This is only possible for elements in case the diagram specification defines the handling of double click on the element.

Drag and Drop from model explorer.

Package: 1) Features
Specification:
Elements can be added by drag and drop from the Model explorer to a dedicated diagram. These dragged elements cannot be edited on the diagram. This feature is only available in case the diagram defines the handling of the dragged element.

Edit elements

Package: 1) Features
Specification:
Elements can only be edited on the diagram the element is created on.
In cases the element is added by drag and drop from the model explorer, the element cannot be edited.

Edit parameters of elements

Package: 1) Features
Specification:
Parameters of an element that are visible on the diagram can be edited in an adapted properties view. Element parameters cannot be edited on the diagram itself.

Model types and prototypes

Package: 1) Features
Specification:
Prototypes have two relationships with types:

- A prototype can be part of a type.
- A prototype has the relationship "is of Type".

Types
There are diagrams that handle types. On these diagrams ports or pins (dependent of the element) can be added to types. If one double clicks on the type, a diagram opens on which the "parts" of that type can be added, i.e. prototypes.
In such a diagram, the ports (or pins) of the type are added, too. This is to enable the creation of delegation connectors.

Prototypes
There are diagrams that handle prototypes. In case one creates a prototype, one has to choose the property "is of Type". In such case the ports (or pins) are inherited from the types and shown on the diagram.

Realize architecture hierarchy.

Package: 1) Features
Specification:
The architecture hierarchy is not defined in one editor only. The structure of the EAST-ADL is considered as close as possible. This means.

The system model is the root of a model.
The system model encloses prototypes that act as the architectures.
As a prototype "is of Type" of a type, this type contains the next hierarchy level, the "parts".
The parts are again prototypes which "are of Type" of a type.

The hierarchy is now defined by the chain of prototypes. With each step one goes into a further hierarchy level.

There is no diagram, that follows exactly this chain. The types are modeled as reusable entities. Therefore, the library of types with their parts are defined first. At the end one only has to define one of the types as the root. This is done with the system model.

2) Editor structure

This chapter specifies the diagrams and their relationships to each other. Each description follows a fixed pattern.

- Purpose
  Short description of what the diagram should do
- Elements
  List of elements that can be added to the diagram. Adding can either be done by creating it from the palette, or by selecting it in the model explorer and drag and drop it to the diagram.
- Relationships
  List of EAST-ADL relationships that can be drawn on the diagram.
- Related diagrams
  Some of the elements may be associated with other diagrams, e.g. to model the internal of a function type. Double click on the element activates the diagram.

2) Editor structure
This chapter specifies the diagrams and their relationships to each other. Each description follows a fixed pattern.

- Purpose
  Short description of what the diagram should do
- Elements
  List of elements that can be added to the diagram. Adding can either be done by creating it from the palette, or by selecting it in the model explorer and drag and drop it to the diagram.
- Relationships
  List of EAST-ADL relationships that can be drawn on the diagram.
- Related diagrams
  Some of the elements may be associated with other diagrams, e.g. to model the internal of a function type. Double click on the element activates the diagram.

Editor structure

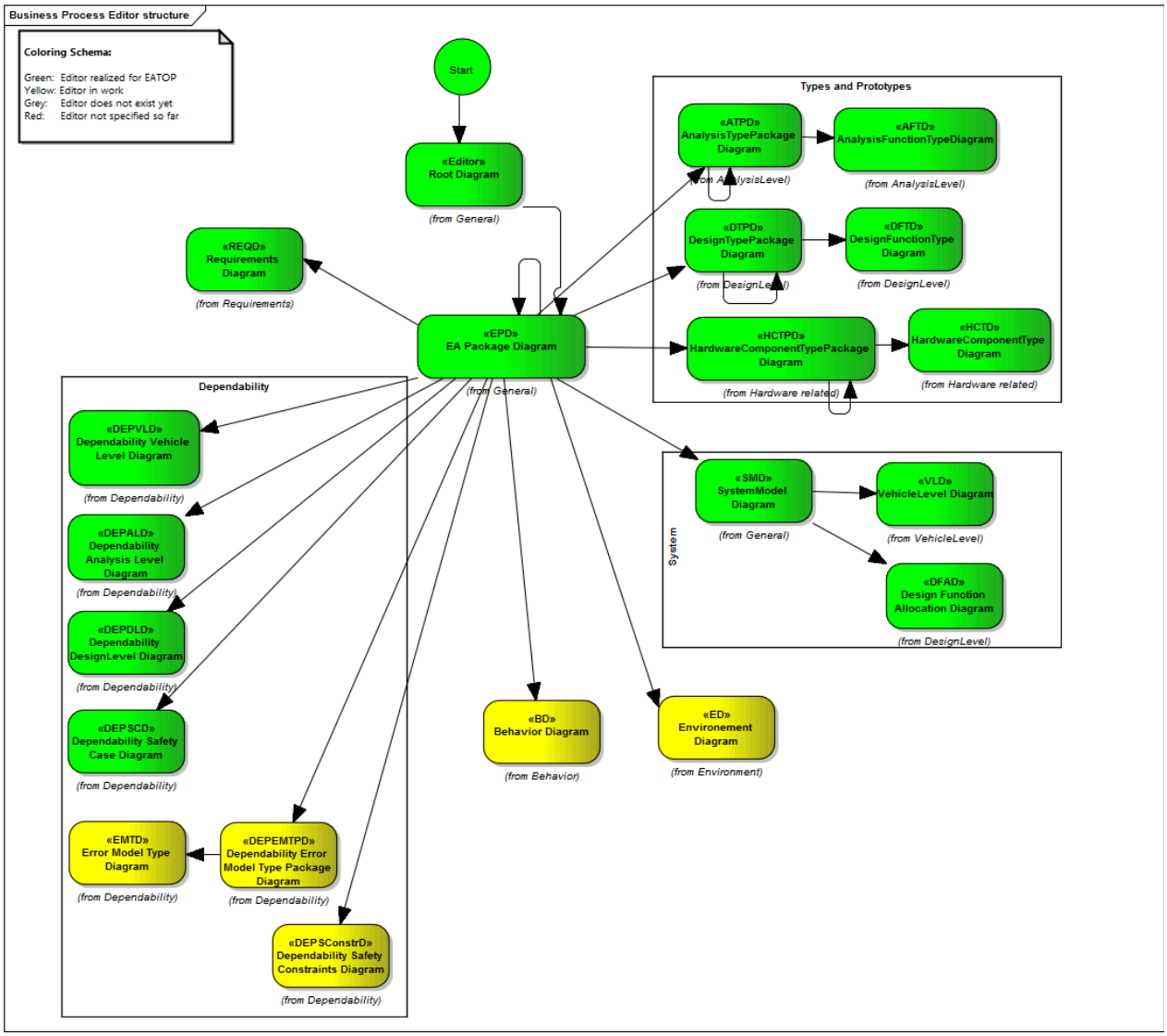

Figure: 2

Dependability

Package: 2) Editor structure
Specification:
Container for all diagrams related with "Dependability" package of EAST-ADL

System

Package: 2) Editor structure
Specification:

Thesis

Package: 2) Editor structure
Specification:
Scope of master thesis from Mourad Najd.

Types and Prototypes

Package: 2) Editor structure
Specification:
Container for all diagrams related with the creation and handling of types of EAST-ADL

AnalysisLevel

2) Editor structure.AnalysisLevel

AnalysisFunctionTypeDiagram

Package: AnalysisLevel
Specification:
Purpose
The AFTD shall allow to define the internals of an AnalysisFunctionType or FunctionalDevices. Internals means that the attribute "parts" of the types are added on this diagram. "Parts" of a type are prototypes.

Elements
- AnalysisFunctionPrototype

Ports (free flying, i.e. not allocated to any element)
- FunctionFlowPort with direction in
- FunctionFlowPort with direction out
- FunctionFlowPort with direction inout
- FunctionClientServerPort with kind client
- FunctionClientServerPort with kind server
- FunctionPowerPort (this port occurs in the palette only, if the element this diagram is detailing is a FunctionalDevice)

Relationships
- FunctionConnector (this needs instanceRefs between two ports)

AnalysisTypePackage Diagram

Package: AnalysisLevel
Specification:
Purpose
The ATPD shall allow to define AnalysisFunctionTypes and FunctionalDevices

Elements
- AnalysisTypePackage (i.e. a normal EAPackage but with the short name "AnalysisTypePackage" such that this package collects all AnalysisFunctionTypes)
- AnalysisFunctionType
- FunctionalDevice

Ports (allocated to an AnalysisFunctionType or a FunctionalDevice)
- FunctionFlowPort direction in
- FunctionFlowPort direction out
- FunctionFlowPort direction inout
- FunctionClientServerPort of kind client
- FunctionClientServerPort of kind server
- FunctionPowerPort (at FunctionalDevice only)

Related Diagrams
- AnalysisTypePackage related with ATPD
- AnalysisFunctionType and FunctionalDevice related with AFTD

DesignLevel

2) Editor structure.DesignLevel

Design Function Allocation Diagram

Package: DesignLevel
Specification:
Purpose
This diagram realizes the allocation of functions to hardware.

Elements
- Select DesignFunctionPrototype from model; no new creation of this element
- Select HardwareComponentPrototype from model; no new creation of this element
- (Not in initial version) Select FunctionConnector from model; no new creation of this element

Relationships
- FunctionAllocation between DesignFunctionPrototype and HardwareComponentPrototype (named by allocatedElement); this includes two instanceRef's

DesignFunctionType Diagram

Package: DesignLevel
Specification:
Purpose
The DFTD shall allow to define the internals of an DesignFunctionType, a BasicSoftwareFunctionType, a LocalDeviceManager, or a HardwareFunctionType. Internals means that the attribute "parts" of the types are added on this diagram. "Parts" of a type are prototypes.

Elements
- DesignFunctionPrototype

Ports (free flying, i.e. not allocated to any element)
- FunctionFlowPort with direction in

- FunctionFlowPort with direction out
- FunctionFlowPort with direction inout
- FunctionClientServerPort with kind client
- FunctionClientServerPort with kind server
- FunctionPowerPort (this port occurs in the palette only, if the element this diagram is detailing is a HardwareFunctionType)

Relationships
- FunctionConnector (this needs instanceRefs between two ports)

DesignTypePackage Diagram

Package: DesignLevel
Specification:
Porpose
The DTPD shall allow to define DesignFunctionTypes, BasicSoftwareFunctionTypes, LocalDeviceManagerTypes and HardwareFunctionTypes.

Elements
- DesignTypePackage (i.e. a normal EAPackage but with the short name "DesignTypePackage" such that this package collects all DesignFunctionTypes)
- DesignFunctionType
- BasicSoftwareFunctionType
- LocalDeviceManager
- HardwareFunctionType

Ports (allocated to a DesignFunctionType, BasicSoftwareFunctionType, LocalDeviceManager or HardwareFunctionType)
- FunctionFlowPort with direction in
- FunctionFlowPort with direction out
- FunctionFlowPort with direction inout
- FunctionClientServerPort with kind client
- FunctionClientServerPort with kind server
- FunctionPowerPort (at HardwareFunctionType only)

Related Diagrams
- DesignTypePackage related with DTPD
- DesignFunctionType (and subtypes) related with DFTD

Hardware related

2) Editor structure.DesignLevel.Hardware related

HardwareComponentType Diagram

Package: Hardware related
Specification:
Purpose
The HCTD shall allow to define the internals of an HardwareComponentType and its subtypes. Internals means

that the attribute "parts" of the types are added on this diagram. "Parts" of a type are prototypes.

Elements
- HardwareComponentPrototype

Pins (free flying, i.e. not allocated to one of the elements)
- CommunicationHardwarePin of direction in
- CommunicationHardwarePin of direction out
- CommunicationHardwarePin of direction inout
- PowerHardwarePin of direction in
- PowerHardwarePin of direction out
- PowerHardwarePin of direction inout
- IOHardwarePin of direction in
- IOHardwarePin of direction out
- IOHardwarePin of direction inout
-

Relationships
- HardwareConnector (this needs instanceRefs between two Pins)

HardwareComponentTypePackage Diagram

Package: Hardware related
Specification:
Purpose
The HCTPD shall allow to define HardwareComponentTypes, Nodes, Sensors, Actuators and ElectricalComponents.

Elements
- HardwareComponentTypePackage (i.e. a normal EAPackage but with the short name "HardwareComponentTypePackage" such that this package collects all HardwareComponentTypes)
- HardwareComponentType
- Node
- Sensor
- Actuator
- ElectricalComponent

Pins (allocated to one of the elements)
- CommunicationHardwarePin of direction in
- CommunicationHardwarePin of direction out
- CommunicationHardwarePin of direction inout
- PowerHardwarePin of direction in
- PowerHardwarePin of direction out
- PowerHardwarePin of direction inout
- IOHardwarePin of direction in
- IOHardwarePin of direction out
- IOHardwarePin of direction inout

Related Diagrams
- HardwareComponentTypePackage related with HCTPD
- HardwareComponentType (and subtypes) related with HCTD

Extensions

2) Editor structure.Extensions

Behavior

2) Editor structure.Extensions.Behavior

Behavior Diagram

Package: Behavior
Specification:
Purpose
This diagram realizes the behavior organization.

Elements
- FunctionBehavior
- ModeGroup
- Mode
- FunctionTrigger
- Select FunctionType (i.e. one of the derived elements) from model; no new creation of this element type
- Select FunctionPrototype (i.e. one of the derived elements) from model; no new creation of this element type
- Select FunctionPort (i.e. one of the derived elements) from model; no new creation of this element type

Relationships
- From FunctionTrigger to Mode (named by mode)
- From FunctionBehavior to Mode (named by mode)
- From ModeGroup to Mode (named by mode; note that this is a containment relationship)
- From FunctionTrigger to FunctionPrototype (named by functionPrototype)
- From FunctionTrigger to FunctonPort (named by port)
- From FunctionTrigger to FunctionType (named by function)

Dependability

2) Editor structure.Extensions.Dependability

Dependability Analysis Level Diagram

Package: Dependability
Specification:
Purpose
This diagram realizes use cases related with the functional safety concept. These use cases are related to the abstraction layer "Analysis Level" of the EAST-ADL.

Elements
- Create FunctionalSafetyConcept
- Create Satisfy
- Select Requirement or QualityRequirement from model; no new creation of this element type
- Select AnalysisFunctionPrototype from model; no new creation of this element type

Relationships
- Satisfy to Requirement (named by satisfiedRequirement)
- Satisfy to AnalysisFunctionPrototype (named by satisfiedBy)
- FunctionalSafetyConcept to Requirement (named by functionalSafetyRequirement)

Dependability DesignLevel Diagram

Package: Dependability
Specification:
Purpose
This diagram realizes use cases related with the technical safety concept. These use cases are related to the abstraction layer "Design Level" of the EAST-ADL.

Elements
- Create TechnicalSafetyConcept
- Create Satisfy
- Select Requirement or QualityRequirement from model; no new creation of this element type
- Select DesignFunctionPrototype from model; no new creation of this element type
-

Relationships
- Satisfy to Requirement (named by satisfiedRequirement)
- Satisfy to DesignFunctionPrototype (named by satisfiedBy)
- TechnicalSafetyConcept to Requirement (named by technicalSafetyRequirement)

Dependability Error Model Type Package Diagram

Package: Dependability
Specification:
Porpose
The DEPEMTPD shall allow to define ErrorModelTypes.

Elements
- ErrorModelType
- Select FunctionType from model; no new creation of this element
- Select HardwareComponentType from model; no new creation of this element

Ports (allocated to an ErrorModelType)
- FaultInPort (enable selection of "isOfType")
- FailureOutPort (enable selection of "isOfType")

Relationships
- From ErrorModeType to FunctionType (named by target)
- From ErrorModelType to HardwareComponentType (named by hwTarget)

Related Diagrams
- ErrorModelType related with EMTD

Dependability Safety Case Diagram

Package: Dependability
Specification:
Purpose
This diagram realizes the safety case documentation.

Elements
- Claim
- Ground
- Warrant
- Select QualityRequirment from model; no new creation of this element type

Relationships
- From Warrant to Ground (named by evidence)
- From Warrant to Claim (named by decomposedGoal)
- From Claim to Group (named by evidence)
- From Claim to Warrant (named by goalDecompositionStrategy)
- From Claim to Warrant (named by supportedArgument)
- From Claim to QualityRequirement (named by safetyRequirement)

Dependability Safety Constraints Diagram

Package: Dependability
Specification:
Porpose
The DEPSConstrD shall allow to define safety constraints.

Elements
- FaultFailure
- QuantitativeSafetyConstraint
- SafetyConstraint
- Select Enomaly (i.e. one of the derived elements) from model; no new creation of this element

Relationships
- From QuantitativeSafetyConstraint to FaultFailure (named by constrainedFaultFailure)
- From SafetyConstraint to FaultFailure (named by constrainedFaultFailure)
- From FaultFailure to Anomaly (named by anomaly; this is an instanceRef); to be clarified: what is the best way to get the ErrorModelPrototype which is needed in the instanceRef; to be able to decide this, the experience with the error model type diagram is needed first.

Dependability Vehicle Level Diagram

Package: Dependability

Specification:
Purpose
This diagram realizes the hazard and risk analysis. This use case is related to the abstraction layer "Vehicle Level" of the EAST-ADL.

Elements
- Item
- FeatureFlaw
- Hazard
- HazardousEvent; display controllability, exposure, severity, harzardClassification
- Select OperationalSituation from model; no new creation of this element type
- Select UseCase from model; no new creation of this element type
- (Not in initial version) Select RequirementsRelationship from model; no new creation of this element type
- Mode
- SafetyGoal; display hazardClassification
- Select Requirement from model; no new creation of this element type
- Select VehicleFeature from model; no new creation of this element type

Relationships
- Item to VehicleFeature (named by vehicleFeature)
- FeatureFlaw to Item (named by item)
- FeatureFlaw to Requirement (named by nonFulfilledRequirement)
- Hazard to Item (named by item)
- Hazard to FeatureFlaw (named by malfunction)
- HazardousEvent to Hazard (named by hazard)
- HazardousEvent to OperationanSituation (named by traffic)
- HazardousEvent to OperationanSituation (named by environment)
- HazardousEvent to UseCase (named by operationalSituationUseCase)
- (not in initial version) HazardousEvent to RequirementsRelationship (named by externalMeasures)
- HazardousEvent to Mode (named by operatingMode)
- SafetyGoal to HazardousEvent (named by derivedFrom)
- SafetyGoal to Mode (named by safeModes)
- SafetyGoal to Requirement (named by requirement)
- (not in initial version) Requirement to Mode (named by mode)

Notes
On vehicle level we run the safety Use case "Hazard & Risk analysis". Starting point is a requirement and the outcome is the safety goal which is refined in safety requirements.
On this level extensions of the "hazard & risk analysis" editor have to be done to handle the requirements better.

Error Model Type Diagram

Package: Dependability
Specification:
Purpose
The EMTD shall allow to define the internals of an ErrorModelType.

Elements
- ErrorModelPrototype (i.e. the attribute "part" of the ErrorModelType; enable selection of "isOfType")
- ProcessFaultPrototype (i.e. the attribute "processFault" of the ErrorModelType)
- InternalFaultPrototype (i.e. the attribute "internalFault" of the ErrorModelType)
- FaultFailurePropagationLink (i.e. the attribute "faultFailureConnector of the ErrorModelType)
- ErrorBehavior (i.e. the attribute "errorBehaviorDescription of the ErrorModelType); display type (attribute of type ErrorBehaviorKind)

- Select FunctionPrototype from model (i.e. one of the derived elements); no new creation of this element
- Select HardwareComponentPrototype from model; no new creation of this element
- Select FunctionPort (i.e. one of the derived elements) from model; no new creation of this element
- Select HardwarePin (i.e. one of the derived elements) from model; no new creation of this element

Ports (free flying, i.e. not allocated to any element)
- FaultInPort
- FailureOutPort

Relationships
- From ErrorModelPrototype to FunctionPrototype (named by functionTarget; this is an instanceRef)
- From ErrorModelPrototype to HardwareComponentPrototype (named by hwTarget; this is an instanceRef)

- From (FaultInPort or FailureOutPort) to FunctionPort (named by functionTarget; this is an instanceRef)
- From (FaultInPort or FailureOutPort) to HardwarePin (named by hwTarget; this is an instanceRef)
- From FaultFailurePropagationLink to FaultInPort (named by fromPort; this is an instanceRef)
- From FaultFailurePropagationLink to FailureOutPort (named by toPort; this is an instanceRef)
-
- From ErrorBehavior to FailureOutPort (named by externalFailure)
- From ErrorBehavior to FailureOutPort (named by internalFailure)
- From ErrorBehavior to FaultInPort (named by externalFault)
- From ErrorBehavior to InternalFaultPrototype (named by internalFault)
- From ErrorBehavior to ProcessFaultPrototype (named by processFault)
-

Environment

2) Editor structure.Extensions.Environment

Environement Diagram

Package: Environment
Specification:
Purpose
The ED acts as a container for Environment modeling.

Elements
- Environment
- Select FunctionPrototype (i.e. one of the derived elements) from model; place FunctionPrototype in Environment; an Environment can contain at maximum one FunctionPrototype only; no new creation of this element type
- Select FunctionPrototype (i.e. one of the derived elements) from model; place FunctionPrototype outside of Environment; no new creation of this element type

Relationships
- ClampConnector (this needs instanceRefs between two ports) connects a FunctionPort of a FunctionPrototype within an Environment with a FunctionPort of another FunctionPrototype (either outside of an Environment or within another Environment). It never connects two FunctionPorts of FunctionPrototypes in case both FunctionPrototypes are outside of an Environment.

Requirements

2) Editor structure.Extensions.Requirements

Requirements Diagram

Package: Requirements
Specification:
Purpose
Collect requirements and provide the possibility to derive them.

Elements
- Requirement
- UseCase
- OperationalSituation
- QualityRequirement

Relationships
- DeriveRequirement
- RequirementLink

Specifics
- Display Type of element, ShortName of element and Text field of element
- In case of QualityRequirement display in addition the attribute kind (of type QualityRequirementKind)

Literature:
- MAENAD\SVN\WP7\WT7.1 Dissemination\ConceptPresentations\7_Requirements_EAST-ADL_Introduction_2012.pptx

General

2) Editor structure.General

EA Package Diagram

Package: General
Specification:
Purpose
The EPD is the entry point for modeling types, a system or the environment. It mainly structures the model into packages. Packages of different types are associated with special diagrams that can be navigated to from this diagram.

Elements

- EAPackage
- AnalysisTypePackage (an EAPackage with this special short name)
- DesignTypePackage (an EAPackage with this special short name)
- HardwareComponentTypePackage (an EAPackage with this special short name)
- SystemModel (an EAPackage with this special short name)
- RequirementModel
- Behavior
- EnvironmentPackage (an EAPackage with this special short name)
- Dependability-VehicleLevel (a Dependability, but the realated diagram encloses elements from vehicle level only)
- Dependability-AnalysisLevel (a Dependability, but the related diagram encloses elements from analysis level only)
- Dependability-DesignLevel (a Dependability, but the related diagram encloses elements from design level only)
- Dependability-ErrorModel(a Dependability, but the related diagram encloses elements from error model only)
- Dependability-SafetyCase (a Dependability, but the related diagram encloses elements from safety case only)
- Dependability-SafetyConstraints (a Dependability, but the related diagram encloses elements from safety constraints only)

Relationships
-

Related Diagrams
- EAPackage related with EPD
- RequirementModel related with REQD
- Behavior related with BD
- EnvironmentPackage related with ED
-
- SystemModel related with SMD
-
- AnalysisTypePackage related with ATPD
- DesignTypePackage related with DTPD
- HardwareComponentTypePackage related with HCTPD
-
- Dependability-VehicleLevel related with DEPVLD
- Dependability-AnalysisLevel related with DEPALD
- Dependability-DesignLevel related with DEPDLD
- Dependability-ErrorModel related with DEPEMTPD
- Dependability-SafetyCase related with DEPSCD
- Dependability-SafetyConstraints related with DEPSConstrD

Root Diagram

Package: General
Specification:
Purpose
Acts as the root diagram editor. A user starts from here.

Elements:
- EAPackage

Relationships
-

Related Diagrams
- If you double click on an EAPackage the EPD diagram opens.

SystemModel Diagram

Package: General
Specification:
Purpose
At the SMD one creates several system models. These act as a containers for the four abstraction levels: VehicleLevel, AnalysisLevel, DesignLevel, and ImplementationLevel.
The prototypes that are added here, act as the top level architectures.

Elements
- VehicleLevel
- AnalysisLevel
- DesignLevel
- ImplementationLevel
- AnalysisFunctionPrototype (only one allowed to add; placement in Analysislevel only; named by FunctionalAnalysisArchitecture)
- DesignFunctionPrototype (only one allowed to add; placement in DesignLevel only; named by FunctionalDesignArchitecture)
- HardwareComponentPrototype (only one allowed to add; placement in DesignLevel only; named by HardwareDesignArchitecture)
- Allocation (placement in DesignLevel only)

Relationships
-

Related Diagrams
- VehicleLevel related with VLD
- Allocation related with DFAD

Start

Package: General
Specification:

VehicleLevel

2) Editor structure.VehicleLevel

VehicleLevel Diagram

Package: VehicleLevel
Specification:
Purpose

The purpose of the Vehicle Level Diagram (VLD) is to allow the user to create feature models and a set of vehicle features.

Elements
- FeatureModel
- VehicleFeature
- FeatureConstraint

Relationships
- FeatureLink

Related Diagrams

Specifics
- In case of a VehicleFeature display attribute "isCustomerVisible"
- Display the kind attribute of a FeatureLink