EAST-ADL_logoBlue

USING PAPYRUS UML FOR EAST-ADL:

A TUTORIAL

Version M2.1.10

Hans Blom, Henrik Lönn (Volvo Technology, SE), Carl-Johan Sjöstedt, De-Jiu Chen (KTH Royal Institute of Technology, SE)

Abstract

This White Paper gives an overview to EAST-ADL. The document is intended for engineers that need a short introduction to the language, through descriptions and examples.

EAST-ADL is an Architecture Description Language (ADL) initially defined in the ITEA project EAST-EEA around 2000. Subsequently, several national and international funded projects have refined the language, and it is now aligned with the more recent AUTOSAR automotive standard. It provides a comprehensive approach for describing automotive electronic systems through an information model that captures engineering information in a standardized form. Aspects covered include vehicle features, requirements, analysis functions, software and hardware components and communication. The representation of the system’s implementation is not defined in EAST-ADL itself but by AUTOSAR. However, traceability is supported from EAST-ADL’s lower abstraction levels to the implementation level elements in AUTOSAR. In this article we describe EAST-ADL in detail, including a case study to show how it relates to AUTOSAR as well as other significant automotive standards and present current research work on using EAST-ADL. in the context of fully-electric vehicles, the functional safety standard ISO 26262 and for multi-objective optimization.

Table of contents

[Abstract 2](#_Toc346882726)

[Table of contents 3](#_Toc346882727)

[1 Introduction 4](#_Toc346882728)

[2 Introduction 5](#_Toc346882729)

[3 Case Overview 7](#_Toc346882730)

[3.1 Tutorial Outline 7](#_Toc346882731)

[4 Part A: Developing A Simple EAST-ADL model. 9](#_Toc346882732)

[4.1 Getting Started 9](#_Toc346882733)

[4.1.1 Enterprise Architect 13](#_Toc346882734)

[4.2 Defining Vehicle Features 14](#_Toc346882735)

[4.2.1 Summary – Feature Modeling 20](#_Toc346882736)

[4.3 Specifying Requirements 21](#_Toc346882737)

[4.4 Developing the Analysis Architecture 24](#_Toc346882738)

[4.5 Using the extension for timing and relating requirements with functions 38](#_Toc346882739)

[5 Part B: First Steps with EAST-ADL Error Modeling 42](#_Toc346882740)

[5.1 Overview 42](#_Toc346882741)

[5.2 EAST-ADL Error Modeling Concepts 42](#_Toc346882742)

[5.3 Tutorial 43](#_Toc346882743)

[5.4 Glossary (on error-modeling) 46](#_Toc346882744)

[6 Part C: Transformation Between EAST-ADL And MATLAB/Simulink® 48](#_Toc346882745)

# Introduction

EAST-ADL represents an Architecture Description Language (ADL) initially defined in the European ITEA EAST-EEA project. It was subsequently refined and aligned with the more recent AUTOSAR automotive standard **Error! Reference source not found.** in national and international funded projects including the ATESST and MAENAD projects **Error! Reference source not found.**, **Error! Reference source not found.**. It is maintained by the EAST-ADL Association **Error! Reference source not found.**.

EAST-ADL is an approach for describing automotive electronic systems through an information model.

# Introduction

This document provides an introduction to the use of the UML editor PapyrusUML for system modeling with EAST-ADL. Papyrus has been used as the reference implementation platform for the EAST-ADL UML profile during the ATESST and MAENAD projects, and several plug-ins have been developed for connection with other tools. Papyrus is based on the Eclipse platform, and open source under the Eclipse Public License. Papyrus and the EAST-ADL profile are freely available for download from the [www.papyrusuml.org](http://www.papyrusuml.org) website.

It should be noted that Papyrus is not the only platform where EAST-ADL is available. Since it can be implemented as a UML profile, it can in principle be used with any UML editor, there is today e.g. an implementation in Magic Draw and Enterprise Architect. Moreover, other EAST-ADL can also be represented using tools which are not based on UML such as MetaEdit+ from MetaCase (<http://www.metacase.com/news/FESA2010.html>) or SystemWeaver. Another possibility is to use the EATOP Eclipse platform.

The objective of the tutorial is to:

* Learn basic concepts of EAST-ADL and the tool environment by creating a first EAST-ADL model using Papyrus with the EAST-ADL UML profile.
* Learn how different EAST-ADL extensions (Timing and Environment in case of this particular tutorial) can be used.
* See how safety analysis can be carried out using the plug-in developed for the HiP-HOPS tool.
* Learn about model transformation between Simulink and EAST-ADL showing how the information can be managed between different tools.

For an introduction to Papyrus the reader is referred to the documentation on the [Papyrus](../Hämtade%20filer/www.papyrusuml.org) website[[1]](#footnote-2). The reader should be familiar with basic UML concepts and diagrams such as class diagrams, package diagrams. The EAST-ADL specifications, available at www.east-adl.info, can also be used for reference.

The tutorial presumes that you have a working installation of the MAENAD Papyrus bundle. The bundle and the instructions for download are available at the MAENAD website[[2]](#footnote-3).

# Case Overview

The whole tutorial is based on the example of a brake-by-wire system as shown in Figure 1.

Controller

Actuator

Brake Pedal Sensor

Wheel Speed Sensor

Vehicle Speed Sensor

Vehicle Dynamics

Wheel Dynamics

Brake Pedal

ABS (Anti-lock Braking System)

Brake Calculator

Environment Model

Figure 1: Brake-by-wire system block diagram

Brake-by-wire is one of the emerging x-by-wire technologies, where electromechanical actuators and human-machine interfaces replaces traditional mechanical or hydraulic interfaces. In this example, the complete by-wire mechanism, which would include e.g. tactile feedback to the driver, is not modeled. The working principle is the same as for a conventional ABS system.

An ABS controller regulates the *slip* to a desired value. The slip is defined as:



where  is the vehicle speed divided by wheel radius.

The slip is zero when the vehicle speed and wheel speed are equal and 1 when the wheels are locked. A typical desired value of the slip is 0.2 for ABS brakes. The vehicle speed is in general not measured directly; only the wheel speed is sensed, so the vehicle speed has to be estimated. As a consequence, the slip value will also be estimated. However, in this example a vehicle speed sensor is modeled, so the slip value will have better fidelity.

## Tutorial Outline

A model of the brake-by-wire system will be created in EAST-ADL, which is Part A of this tutorial, starting at the following page.

The brake function of a car is indeed a safety-critical feature, and EAST-ADL supports error-modeling. To make an analysis of the system, the tool HiPHOPS can be used through a plugin. This is explained in Part B starting on page 42.

To predict the performance of the system, it can be modeled in MATLAB/Simulink. A plugin to the PapyrusUML environment supports bi-directional transformation between EAST-ADL and MATLAB/Simulink, this is demonstrated in Part C (page 48).

# Part A: Developing A Simple EAST-ADL model.

In this part we will go through the following steps, to model the two highest abstraction levels of our system:

1. Defining features.
2. Specifying requirements and associating the requirements with the features.
3. Developing components and associating the components with the requirements.
4. Developing the Analysis Architecture.



Figure 2: The EAST-ADL model structure. In this part, we will develop a model at the two top-most abstraction levels.

## Getting Started

In MAENAD, a pre-packaged Eclipse installation was released. However, to make sure you have the latest version of the profile and all plugins, you can also follow these steps:

* Install Eclipse M2M
* Install Papyrus using the Papyrus update site
* Install the EAST-ADL profile
* Ensure that the selected perspective is Papyrus

The installation instruction for the above items is described more in detail at the Papyrus website.

The Papyrus content is organized using projects. One project can contain many different models and files of different kinds.

1. Create a new project. “File ->New->Papyrus Project” as shown in Figure 3.

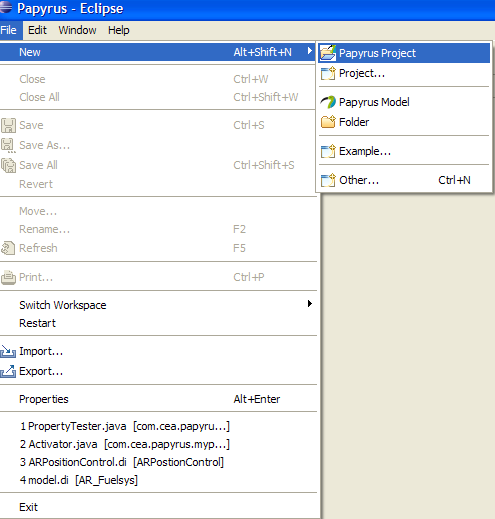


Figure 3: Start a new project

1. Write a suitable name for the project, for example MyFirstEAST, and click on “Finish” (Figure 4).

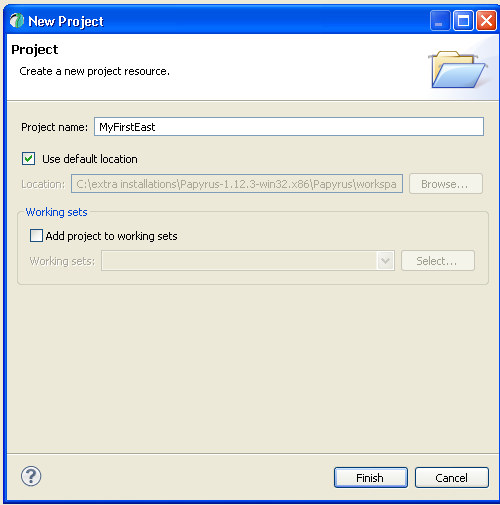


Figure 4: Select an appropriate name for your project

1. Select the initialization information as shown in the following figure (Figure 5)

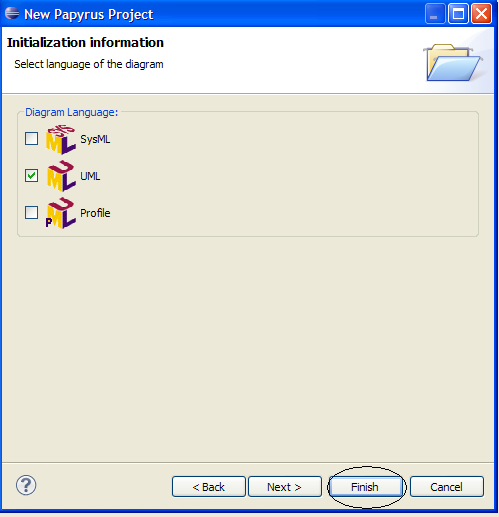
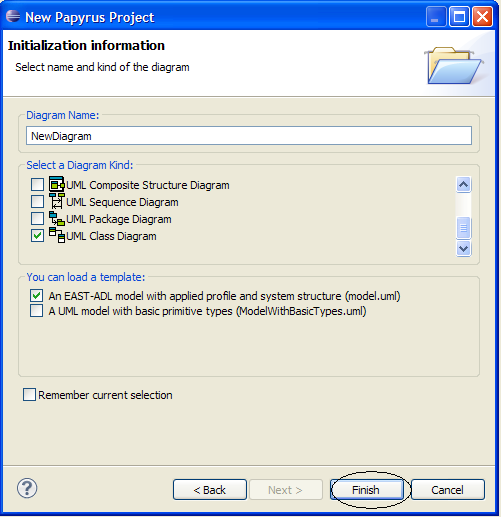
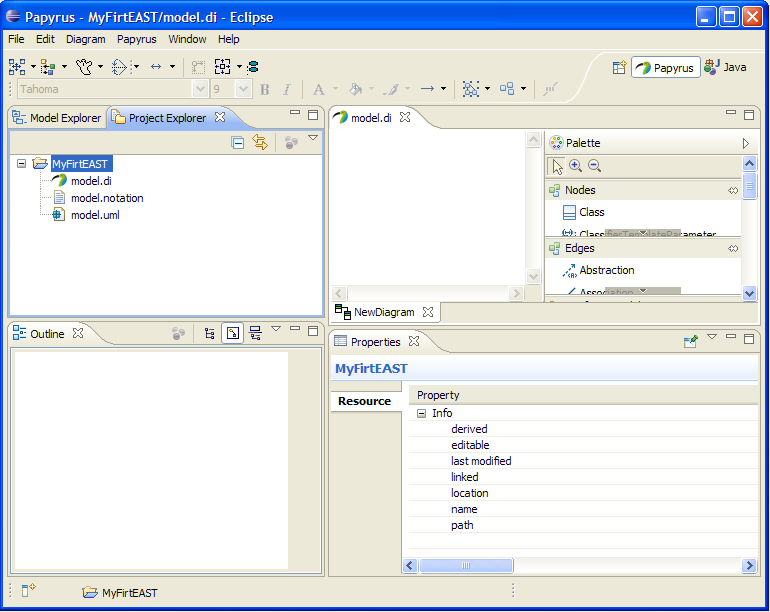
 

Figure 5: Two steps to create a new EAST-ADL model

You have just created a new project and an EAST-ADL model. The resulting window is shown in Figure 6.



Properties

Outline view

Palette

Navigator view

Figure 6: An empty EAST-ADL model inside your project

1. In the Project Explorer pane of the navigator view, three files have been created, the .di file, the .uml file and .notation file. The .di2 file is the graphical interface to the .uml file, where all model data is stored. In this part of the tutorial, we will only work with the .di file.
2. The diagram in Figure 7 shows the Model Explorer pane in its part b, and the EAST-ADL abstraction levels in its part a, which are created automatically. The terms faa, fda and had refer to “functional analysis architecture”, “functional design architecture” and “hardware design architecture” respectively. Also note that the VehicleFeatureModel is not created, this will be our first task.
3. On the right side of the window is the palette providing direct access to EAST-ADL elements is displayed. These elements are UML elements with assigned stereotypes. They can also be created manually by inserting UML elements and applying EAST-ADL stereotypes. We will practice both these techniques throughout this tutorial.
4. The diagrams will appear in the center. In the figure there is only one diagram named “New diagram” marked with an oval.

(a)

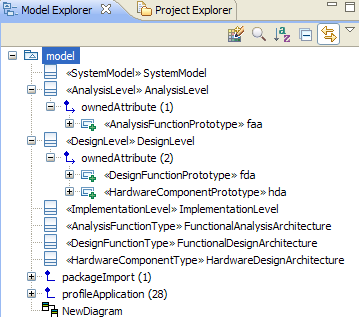


Figure 7: The outline window below, and the corresponding EAST-ADL abstraction levels above

### Enterprise Architect

A similar model can be created in Enterprise Architect.

Begin by creating an new project and save it to anly local file. Select “Basic UML Technology” in the dialogue.

Open the Resource palette (Alt-6) and right click “UML Profiles” to import “EnterpriseArchitectProfile\_EAST-ADL2.1.10.xml”

Add a view under the project Root and call it RootView. In this package a top level package can be created, call RootPackage. Add a CompositeStructureDiagram in the RootPackage and call it SystemModel. Add packages VehicleLevelElements, AnalysisLevelElements, DesignLevelElements and ImplementationLevelElements.

## Defining Vehicle Features

The Vehicle Feature Model is the topmost abstraction level of EAST-ADL, (see



Figure 2 on page 9 for a recap of the abstraction levels). A feature represent the properties/functionality/traits e.g. Brake, Wiper, Collision Warning. These are the main characteristics of a feature:

* Features are stakeholder-requested functional or non-functional characteristics of a vehicle
* A Feature describes that "what", but shall not fix the "how"
* A Feature might be refined by further requirements

In Figure 8, we see the feature tree for the ABS system that we are about to model.

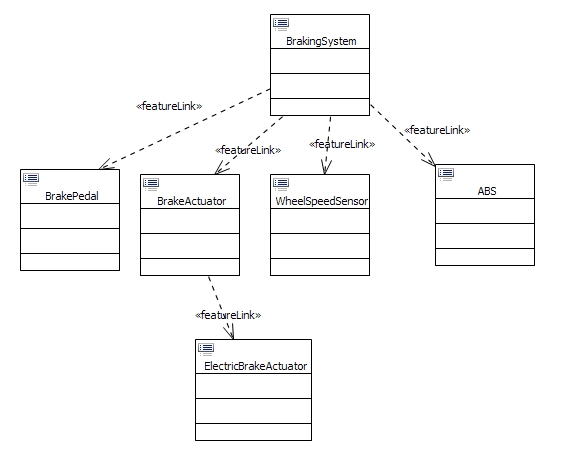


Figure 8: The feature tree we are about to model

In UML, packages are used to organize the model, so we start out by creating a feature package.

1. Creating a new package for vehicle features.
   1. Right click on model in the ModelExplorer window.
   2. Select “New Child” followed by “Create a new Package”
   3. Click on “package” (Figure 9).

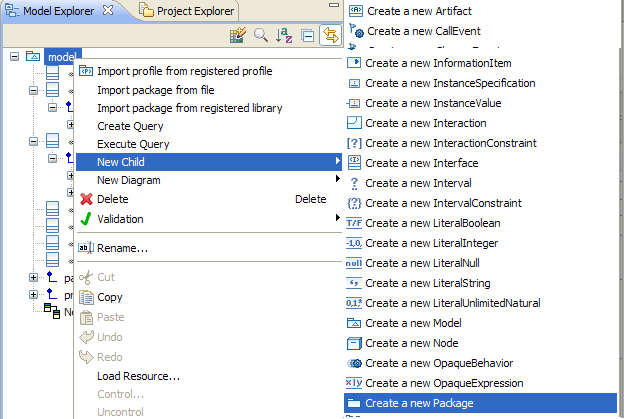


Figure 9: Create a new package

* 1. A new package will appear (Figure 10).

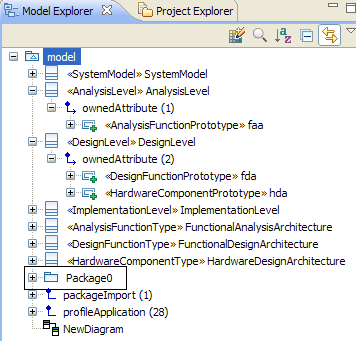


Figure 10: The newly created package

* 1. Click on the newly created package and change the name to “FeaturePackage” in the properties window as shown in Figure 11.

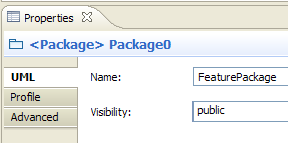
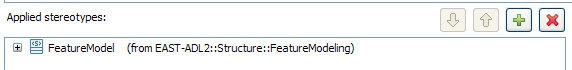


Figure 11: To rename objects, use the General tab in the properties window

* 1. Apply a “Feature Model” stereotype.
     1. Click on “Profile” in the properties window followed by the  button.



* + 1. In the new window select the “FeatureModel” stereotype . The final output should look like the following figure.



1. Create a new class diagram for the feature package (Figure 12).

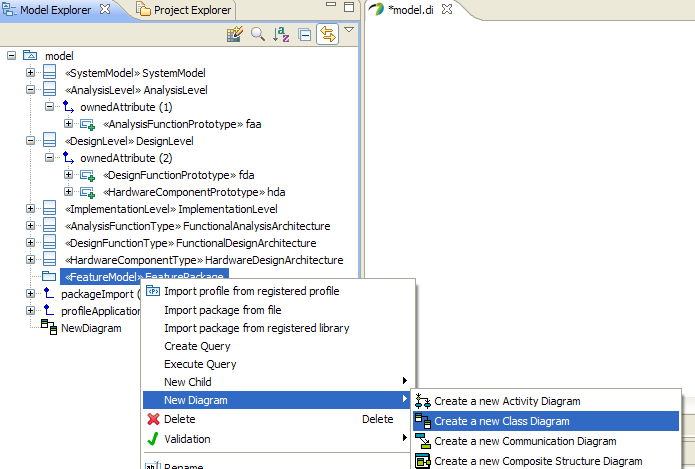


Figure 12: Add a class diagram

* 1. Right click on FeaturePackage in the outline window.
  2. Select “Add a diagram”.
  3. Click on “Create a new Class Diagram”.
  4. Name the new diagram as “FeatureModel” when asked for its new name.
  5. A new class diagram will appear. (Figure 13).

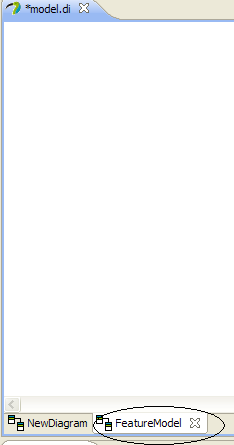


Figure 13: Rename the newly created class diagram

1. Add a new Feature.
   1. Click on the “VehicleFeature” to select from the palette (Figure 14).

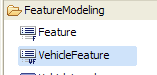


Figure 14: Add a new feature

* 1. Click on the newly created diagram. A class should appear in the diagram.
  2. Change the name of the class (VehicleFeature0) to “BrakingSystem” (see Figure 11 for how to rename objects).
  3. The resulting Model Explorer window and the diagram should look like Figure 15.

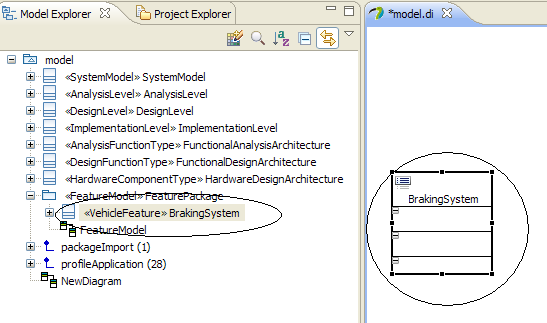


Figure 15: A new feature in model explorer

* 1. Click on the BrakingSystem feature, then “profile” in the properties window to view the supported parameters of a vehicle feature by EAST-ADL (Figure 16).

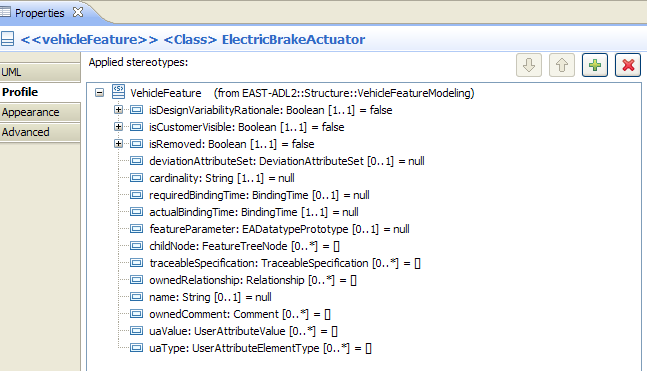


Figure 16: The supported parameters of a vehicle feature – accessed in the stereotype

1. Add the following five more vehicle features:
   1. BrakePedal
   2. BrakeActuator
   3. WheelSpeedSensor
   4. ABS
   5. ElectricBrakeActuator
2. The resulting diagram should be similar to Figure 17.

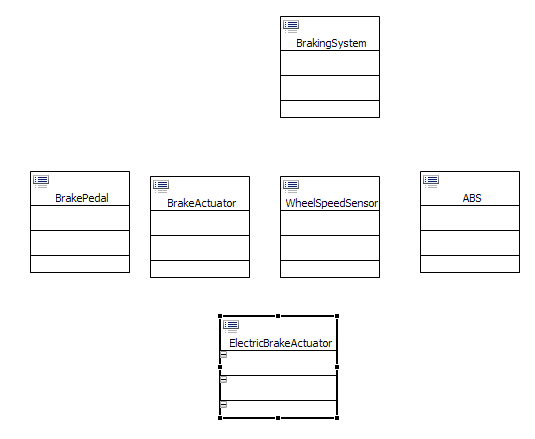


Figure 17: The features of the ABS system

1. Linking different features.
   1. Select dependency connection from the UML Edges of the Palette and connect BrakingSystem with BrakePedal feature. The link should start from the BrakingSystem (Figure 18).

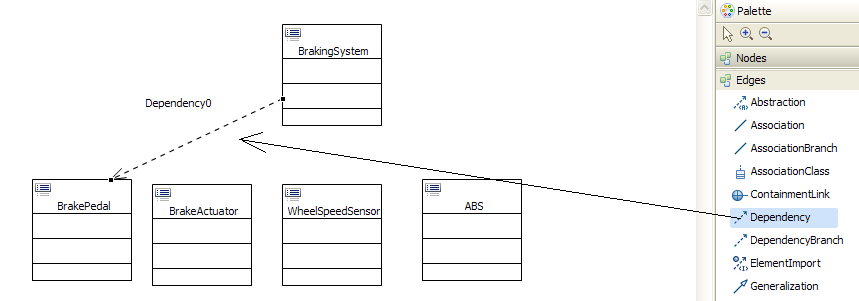


Figure 18: Create links between BrakingSystem and BrakePedal features

* 1. Select the newly created link and add the FeatureLink Stereotype (Figure 19).

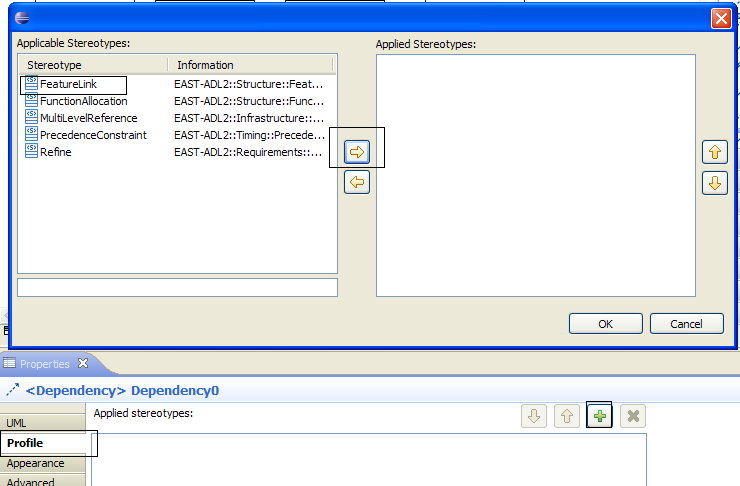


Figure 19: Apply Featurelink stereotype

1. Repeat step 8 and link different features as follows:
   1. BrakingSystem -> BrakeActuator
   2. BrakingSystem -> WheelSpeed
   3. BrakingSystem -> ABS
   4. BrakeActuator -> ElectricBrakeActuator

Go back to Figure 8 on page 14 to verify your model.

1. Observe the existence of the newly created features and their links in the Model Explorer (Figure 20).

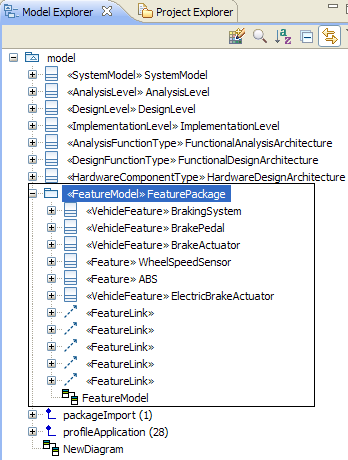


Figure 20: The features, as displayed in the model explorer

### Summary – Feature Modeling

We have now started to create our first EAST-ADL model, and learned the basics of the Papyrus UML modeler.

The feature tree of a brake system has been modeled, using featureLinks to define the relationship between different features. Note that we have not defined the semantics of these featureLinks in the tree; this must be made in a real case.

The feature model is a showcase of the functionality that is realized by the system represented by the EAST-ADL model. It is also the origin of many configurations affecting the variable contents of an EAST-ADL model.

## Specifying Requirements

Requirements are specified, and linked to objects in the architecture using *Satisfy, Refine, DeriveRequirement, or Verify* relations depending on the need.

1. Create a new “RequirementsPackage” and a class diagram followed by the application of “Requirements Model” stereotype, just like you did for the FeaturePackage in the previous section. The result is shown in Figure 21:

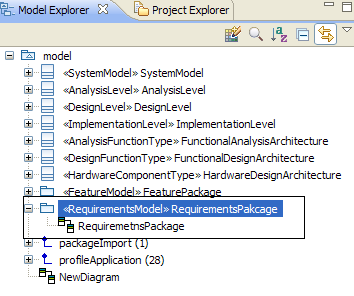


Figure 21: Create a RequirementsPackage and a class diagram

1. Add a new quality requirement “Robustness” using the palette to the newly created class diagram as shown in Figure 22:

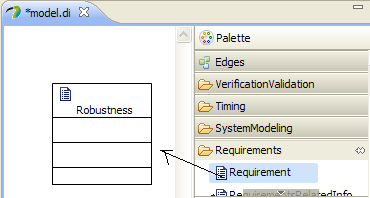


Figure 22: Add a QualityRequirement

* 1. Set the qualityRequirementType to “other” as shown in Figure 23

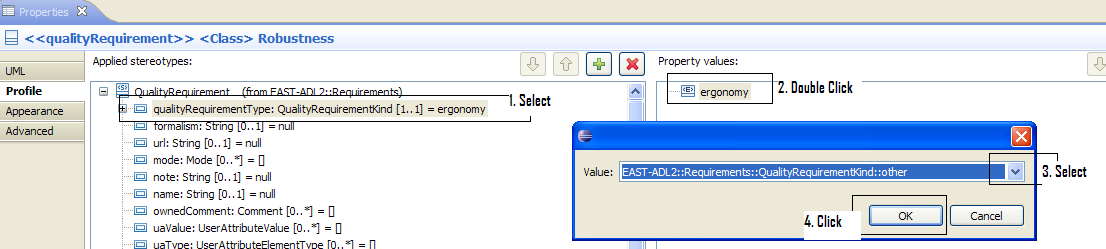


Figure 23: Set the Quality Requirement Type

* 1. Add the following requirement as a string as shown in Figure 24.

“The system should be robust for both low and high slippery conditions”

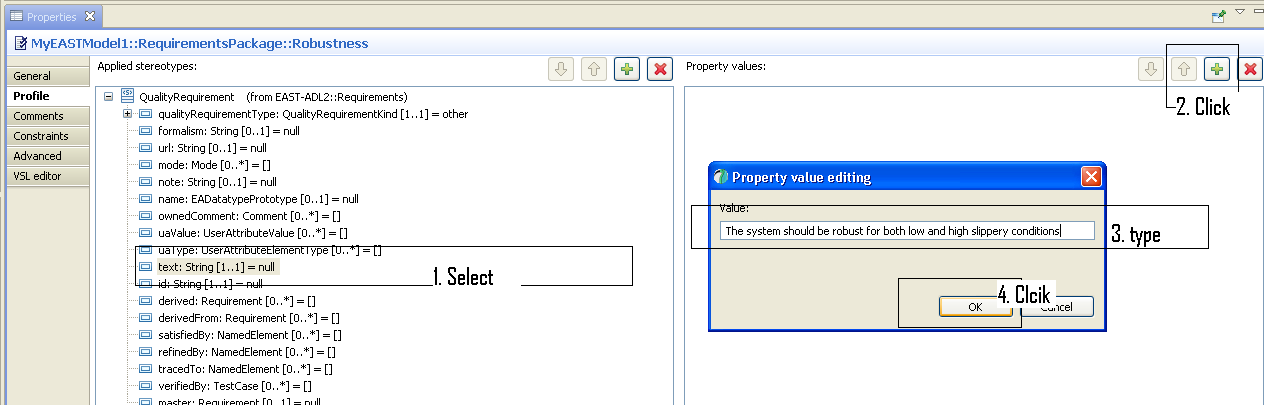


Figure 24 : Storing the textual description of a requirement

Check the other properties and refer to the EAST-ADL specifications for details.

1. Repeat step 2 to add the following quality requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO.** | **QualityRequirementKind** | **Name** | **Comments** |
| 1 | Timing | OverallTimingRequirement | The total execution time should not exceed 10ms. |
| 2 | Safety | Safety | It should be possible to warn the driver about failures. |
| 3 | Other | DrivingConditionA | It should be possible to use the system on ice. |
| 4 | Other | DrivingConditionB | It should be possible to use the system on asphalt. |

1. Select DeriveRequirement from the palette and connect Robustness to DrivingConditionA and DrivingConditionB as shown in Figure 25.

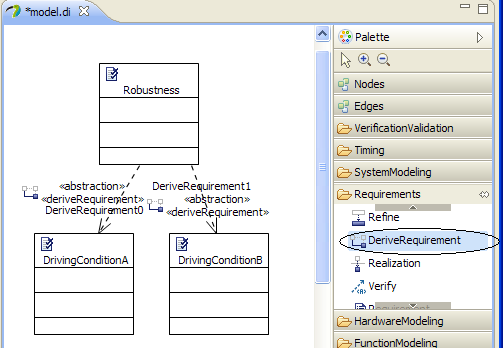


Figure 25: Using derived requirements

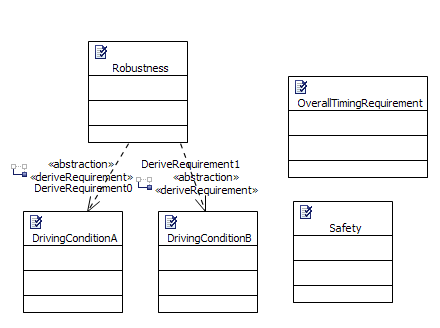


Figure 26: Using derived requirements

## Developing the Analysis Architecture

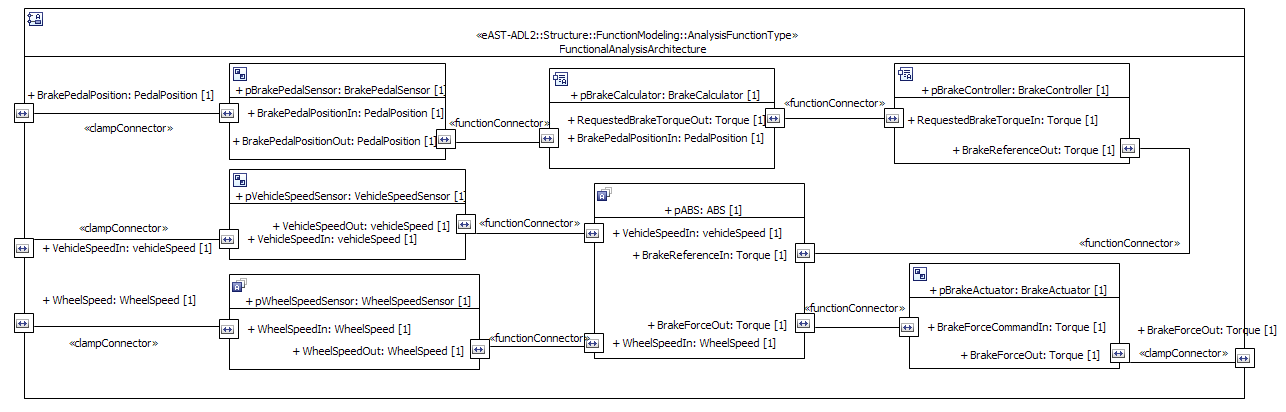


Figure 27: A complete functional analysis architecture of the brake system

In this section, the Analysis Architecture will be created, containing the Functional Analysis Architecture (Figure 27) and the Environment Model. Abstract functions are created, which have inports and outports, which are typed by datatypes, also defined at this level. The environment model is modeled in a similar fashion, and it is connected to the functional analysis architecture. The requirements are

*For engineers working in block model software, e.g. MATLAB/Simulink, the analysis architecture will be familiar, with blocks connected to each other through directional ports. The Simulink exchange plugin, described in Part C of this tutorial, transforms between Simulink subsystems and AnalysisFunctions.*

1. Create a package named “FunctionalComponents” and add a class diagram (as done in previous sections).
2. The AnalysisFunctionType is available inside the FunctionComponents folder (Figure 28). Add the following AnalysisFunctionTypes:
   1. BrakeCalculator
   2. BrakeController
   3. ABS
   4. WheelDynamics
   5. VehicleDynamics
   6. BrakePedal

*A FunctionType is the main building block when modeling functions in EAST-ADL. AnalysisFunctionType is used here at the analysis level, and in the environment model, and DesignFunctionType is used at the design level.*

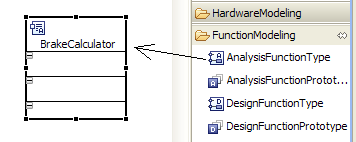


Figure 28: Adding an AnalysisFunctionType

*A FunctionalDevice is a specialized AnalysisFunctionType that represents the interface between the embedded system and the controlled system. Typically, a FunctionalDevice is a sensor or an actuator.*

1. Add the following Functional Devices, also available in the FunctionModeling folder of the palette:
2. WheelSpeedSensor
3. VehicleSpeedSensor
4. BrakePedalSensor
5. BrakeActuator
6. Create a new composite diagram at the AnalysisLevel, as shown in Figure 29.

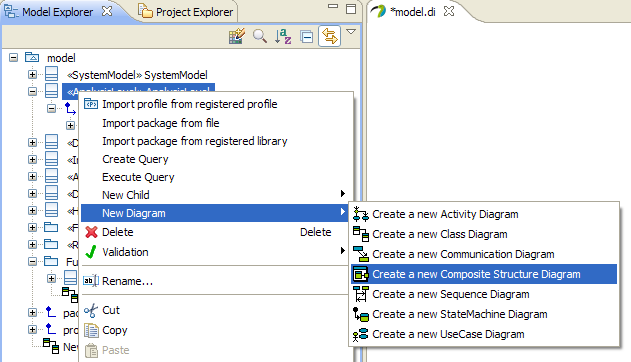


Figure 29: Create a composite diagram

1. The outcome is shown in Figure 30.

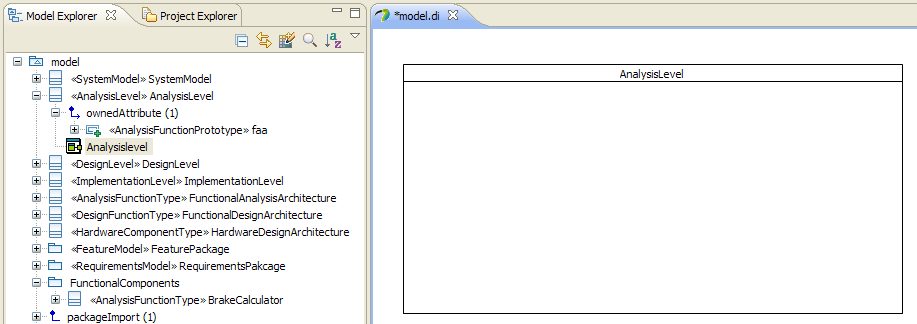


Figure 30: AnalysisArchitecture in our composite diagram

* 1. Change the appearance of the new class (Figure 31).
     1. Click on the appearance tab ,
     2. Select the AnalysisLevel in the “Applied Stereotype” box.
     3. Click on the display symbol
     4. Select “Text and Icon” in the stereotype Appearance menu

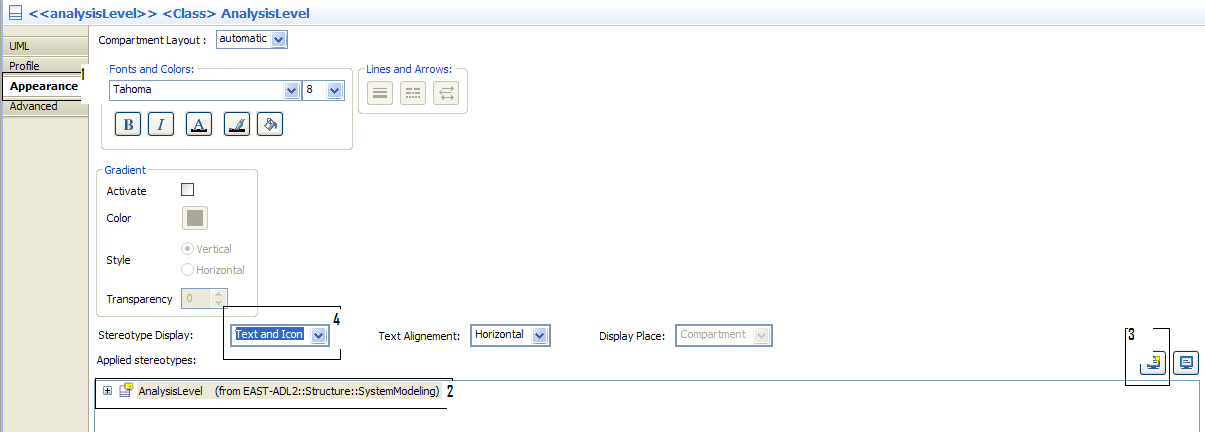
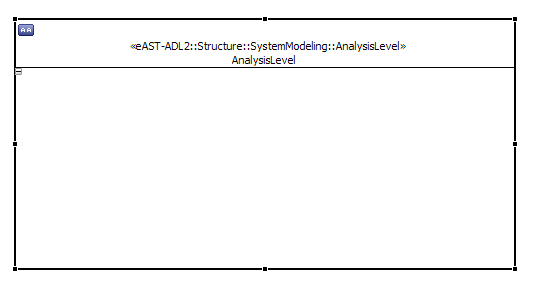


Figure 31: Change the appearance of a class

1. Repeat steps 3, 4 and 5 for the FunctionalAnalysisArchitecture. The result is shown in Figure 32:

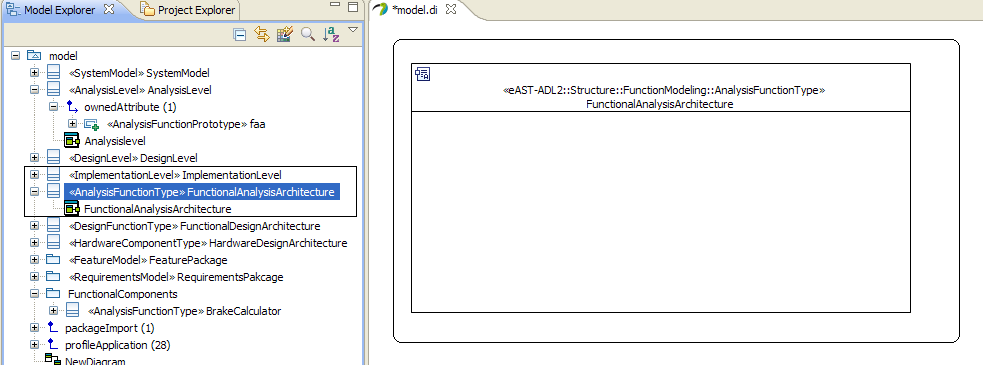


Figure 32: Create a composite diagram of the FunctioalAnalysisArchitecture

*An important concept in EAST-ADL is the use of types and prototypes. A prototype is the use of a type in a context, e.g. this composite diagram. A prototype must then refer to a defining type. This enables re-use of components.*

1. Add an AnalysisFunctionPrototype to the FunctionalAnalysisArchitecture as shown in Figure 33.
   1. Drag AnalysisFunctionPrototype from the palette.
   2. Change the name from Property0 to pBrakeController (notice naming convention for prototypes, a p before the type name)

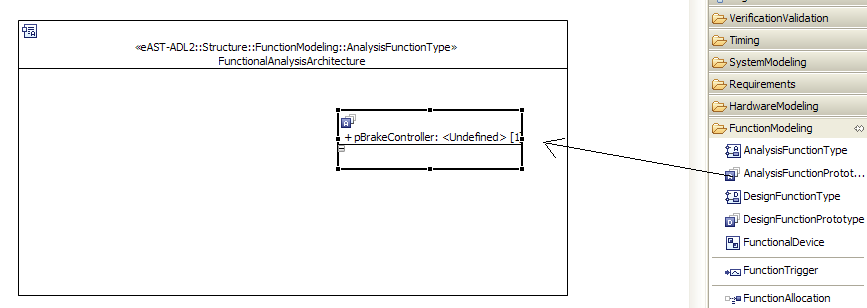
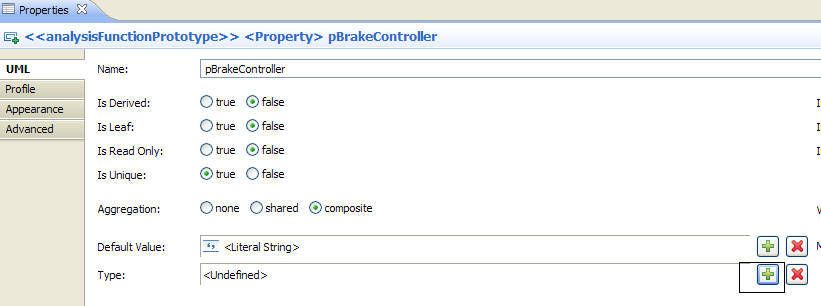


Figure 33: Adding an AnalysisFunctionPrototype

* 1. Click on the “+” sign to select the related FunctionType.



* 1. Select type BrakeController as shown in Figure 34.

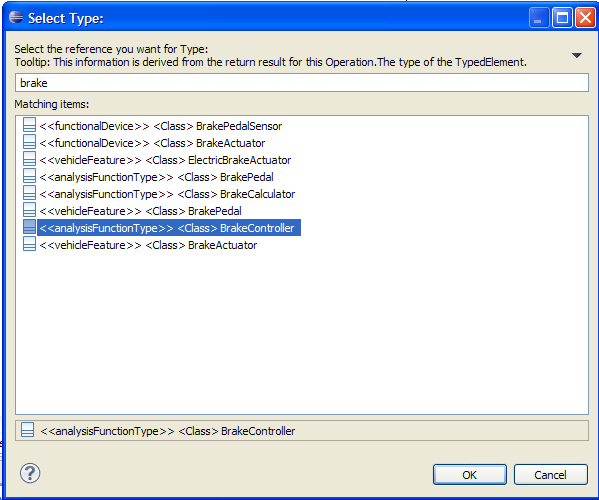


Figure 34: Setting the type of the prototype created in Figure 33

1. In a similar way, add the following FunctionPrototypes
2. pBrakePedalSensor
3. pBrakeCalculator
4. pABS
5. pWheelSpeedSensor
6. pVehicleSpeedSensor
7. pBrakePedalActuator

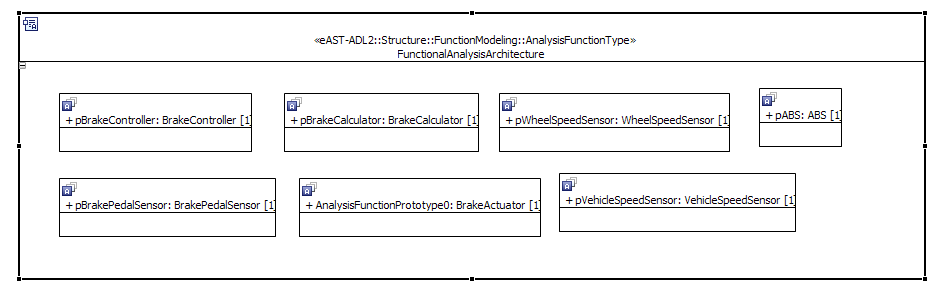


Figure 35: Function Prototypes in the FAA

1. Adding DesignDataTypes.
   1. Add a package named “DataPackage” and a class diagram for this package.
   2. Add a new DataType in the newly created package (Figure 36).

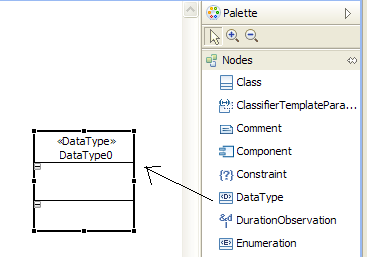


Figure 36: Add a datatype element

* 1. Apply a new stereotype named “EAInteger” to the new datatype (see Figure 22 and Figure 23 on page 21 for how to apply stereotypes).
     1. Rename the new datatype as PedalPosition. The result should look like Figure 37.
     2. Set the max and min values to 0 and 100 respectively which corresponds to 0 and 100%.

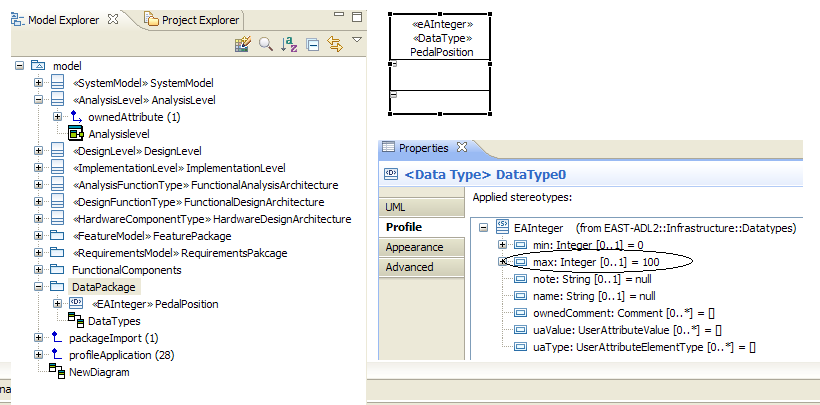


Figure 37: A design datatype

1. Repeat the last step to add the following ADLDesignDataTypes

|  |  |  |
| --- | --- | --- |
| Name | Applied Stereotype | Property Values |
| Torque | EAInteger | maxValue=1000  minValue=0  note = N.m |
| WheelSpeed | EAInteger | maxValue=50  minValue=0  note = rad/s |
| VehicleSpeed | EAInteger | maxValue=200  minValue=0  note = km/h |

1. Adding Flow ports
   1. Select FunctionFlowPort from the palette and click on pBrakeController
   2. Follow the same procedure as in step 7 to
      1. Change the name of the port to RequestedBrakeTroqueIn .
      2. Select “Torque” as type.

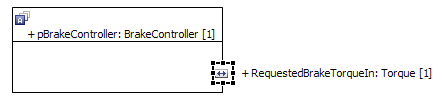


Figure 38: Adding ADLFlowPorts

* 1. Notice the addition of the port in the FunctionType definition as shown in Figure 39

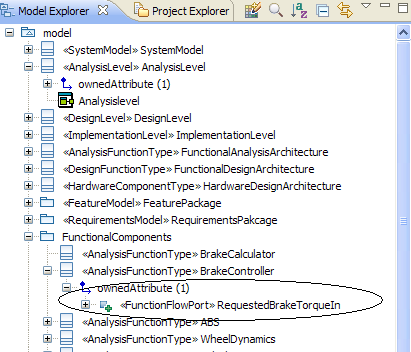
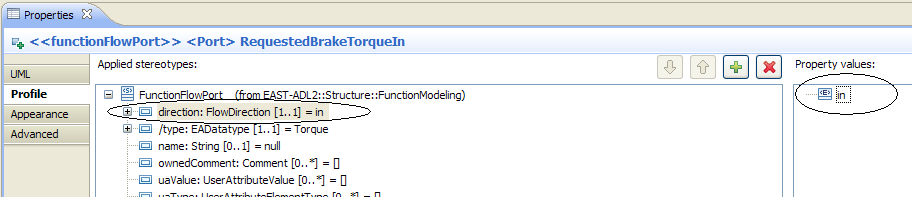


Figure 39: The port is displayed in the FunctionType

1. Set the direction of the flow to “in” as shown in the following figure.

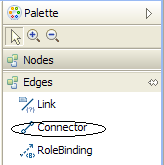


1. Repeat the last step for adding more Flow Ports according to Table 1.

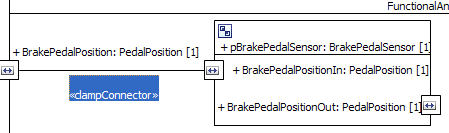
Table 1: Flow ports of the brake-by-wire system

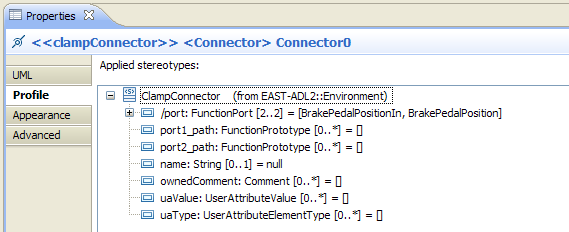
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PrototypeName** | **Type of Port** | **Direction** | **Name** | **DataType** |
| FunctionalAnalysisArchitecture | FunctionFlowPort | in | VehicleSpeedIn | VehicleSpeed |
| FunctionalAnalysisArchitecture | FunctionFlowPort | in | WheelSpeed | WheelSpeed |
| FunctionalAnalysisArchitecture | FunctionFlowPort | in | BrakePedalPosition | PedalPosition |
| FunctionalAnalysisArchitecture | FunctionFlowPort | out | BrakeForceOut | Torque |
| pBrakeCalulator | FunctionFlowPort | in | BrakePedalPositionIn | PedalPosition |
| pBrakeCalulator | FunctionFlowPort | out | RequestedBrakeTorqueOut | Torque |
| pBrakeController | FunctionFlowPort | out | BrakeReferenceOut | Torque |
| pABS | FunctionFlowPort | in | VehicleSpeedIn | VehicleSpeed |
| pABS | FunctionFlowPort | in | BrakeReferenceIn | Torque |
| pABS | FunctionFlowPort | in | WheelSpeedIn | WheelSpeed |
| pABS | FunctionFlowPort | out | BrakeForceOut | Torque |
| pPedalSensor | FunctionFlowPort | in | BrakePedalPositionIn | PedalPosition |
| pPedalSensor | FunctionFlowPort | out | BrakePedalPositionOut | PedalPosition |
| pWheelSpeedSensor | FunctionFlowPort | in | WheelSpeedIn | WheelSpeed |
| pWheelSpeedSensor | FunctionFlowPort | out | WheelSpeedOut | WheelSpeed |
| pVehicleSpeedSensor | FunctionFlowPort | out | VehicleSpeedOut | VehicleSpeed |
| pVehicleSpeedSensor | FunctionFlowPort | in | VehicleSpeedIn | VehicleSpeed |
| pBrakeAcutator | FunctionFlowPort | in | BrakeForceCommandIn | Torque |
| pBrakeActuator | FunctionFlowPort | out | BrakeForceOut | Torque |

1. Connecting ports to the external world.
   1. Select Connector from the palette



* 1. Select BrakePedalPosition port of the FunctionalAnalysisArchitecture and connect it with the input port of pBrakePedalSensor.
  2. Select the newly created connector and apply the sterotype “ClampConnector” as shown in the following figure.





1. Repeat the last step to connect
   1. VehicleSpeedIn port of FunctionalAnalysisArchitecture to VehicleSpeedIn port of pVehicleSpeedSensor.
   2. WheelSpeedIn port of FunctionalAnalysisArchitecture to WheelSpeedIn port of pWheelSpeedSensor.
   3. BrakeForceOut port of pBrakdePedalActutor to BrakeForceOut port of FunctionalAnalysisArchitecture.
2. Connecting two internal ports.
   1. Select FunctionConnector from the palette Figure 40.
   2. Select the output port of pBrakeCalulator and connect it with the input port of the pBrakeController.

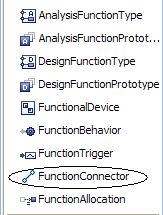


Figure 40: The FunctionConnector is used to connect two ports

1. Rearrange the blocks and repeat the last step to connect other ports as shown in Figure 41. (Notice the difference of clamp connector and the FunctionConnetor)

Note: The are two different connectors.

* 1. Clamp connectors are only used with functional devices. The functionaldevices serve as the boundary of the system. An connection with the outside world is made via a clamp connector.
  2. FunctionConnectors are used inside a system.

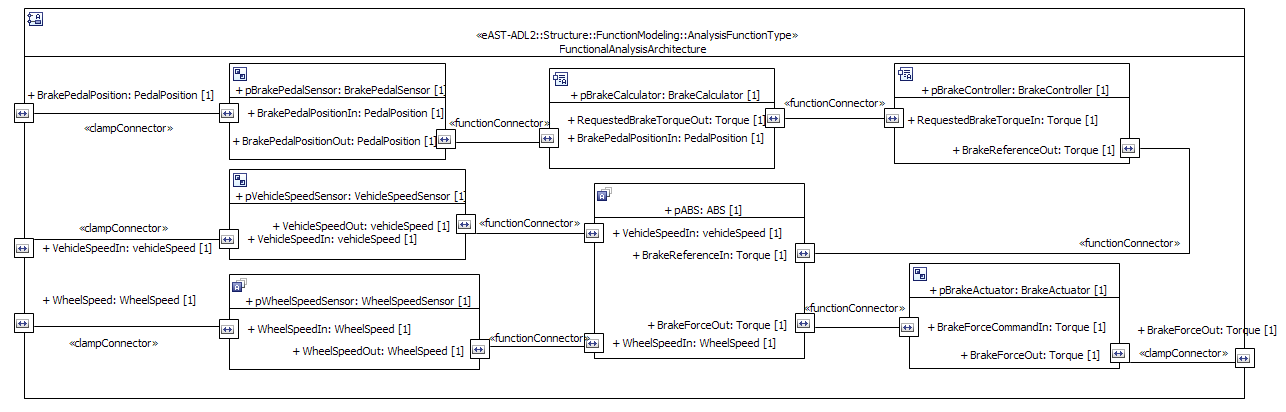


Figure 41: The completed Functional Analysis Architecture

1. Create a new package “Environment Model”.
2. Add a new composite structure diagram to the newly created package.
3. Add a new UML class from the “Nodes” section of the palette, rename it to “EnvironmentModel” and apply the stereotype “environment”.
4. Adjust the appearance. The result should look like the diagram in Figure 42.

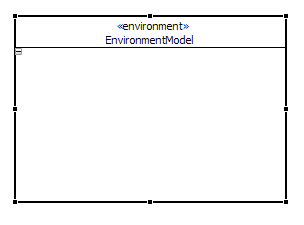


Figure 42: The EnvironmentModel

1. Add the following AnalysisFunctionPrototypes to the Environment Model (see Figure 43). (Note: You have already created the FunctionTypes)
   1. pVehicleDynamics
   2. pWheelDynamics
   3. pBrakePedal
2. Add the ports in Table 2.

Table 2: Ports of the brake-by-wire system

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PrototypeName** | **Type of Port** | **Direction** | **Name** | **DataType** |
| pVehicleDynmaics | FunctionPowerPort |  | VehicleSpeedOut | VehicleSpeed |
| pVehicleDynmaics | FunctionPowerPort |  | WheelSpeedIn | WheelSpeed |
| pVehicleDynmaics | FunctionFlowPort | out | VehicleSpeedOut | VehicleSpeed |
| pWheelDynamics | FunctionPowerPort |  | WheelSpeedOut | WheelSpeed |
| pWheelDynamics | FunctionFlowPort | in | BrakeForceIn | Torque |
| pWheelDynamics | FunctionFlowPort | out | WheelSpeedOut | WheelSpeed |
| pWheelDynamics | FunctionPowerPort |  | VehicleSpeedIn | VehicleSpeed |
| pBrakePedal | FunctionFlowPort | out | PedalPress | PedalPosition |

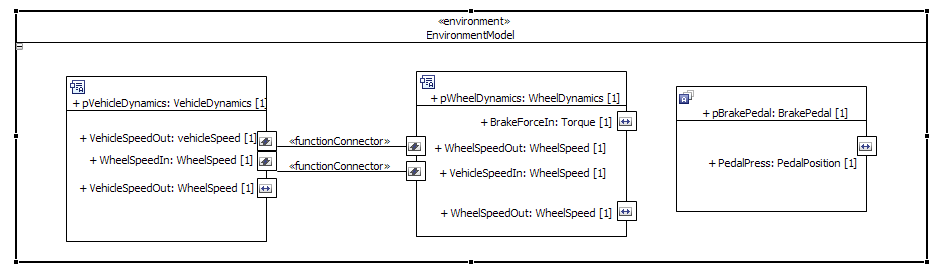


Figure 43: The completed environment model

Connect the internal ports with functionConnector and the external ports with clampConnectors as shown in Figure 43.

To connect the environment model with the functional analysis architecture, prototypes containing the complete FAA and the environment model must be modeled.

1. Create a new composite diagram for the overall model as shown in the following figure:

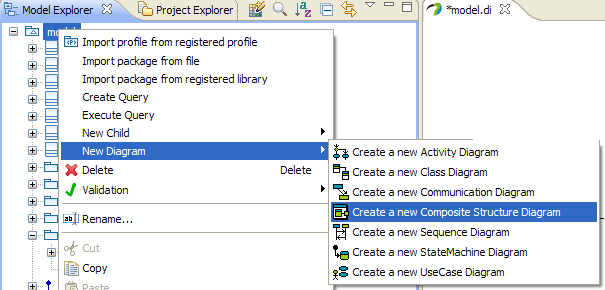


Figure 44. Adding a Composite Structure Diagram for Plant Model connection

1. Drag AnalysisLevel and EnvironmentModel Classes in the newly created diagram (step 1 and 2 in Figure 44.
2. Drag FAA inside the analysis level (step 3 in Figure 44).
3. Change the appearance as shown in Figure 44.

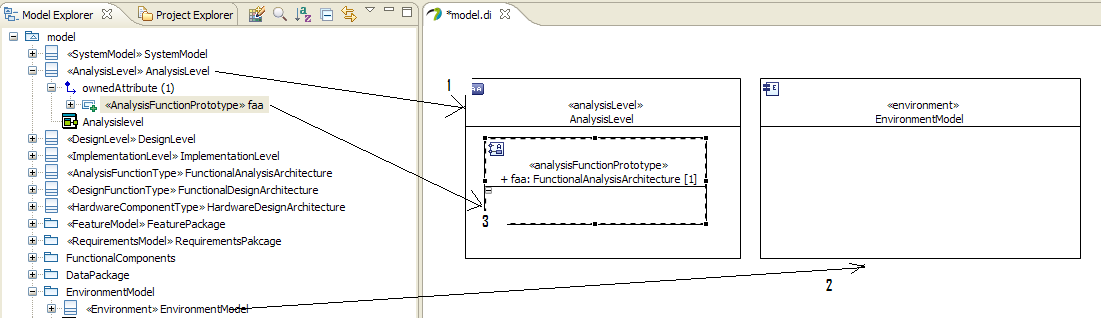


Figure 45: Creating connection between Analysis level and Environment Model.

1. In a similar way add the associated ports as shown in Figure 45.

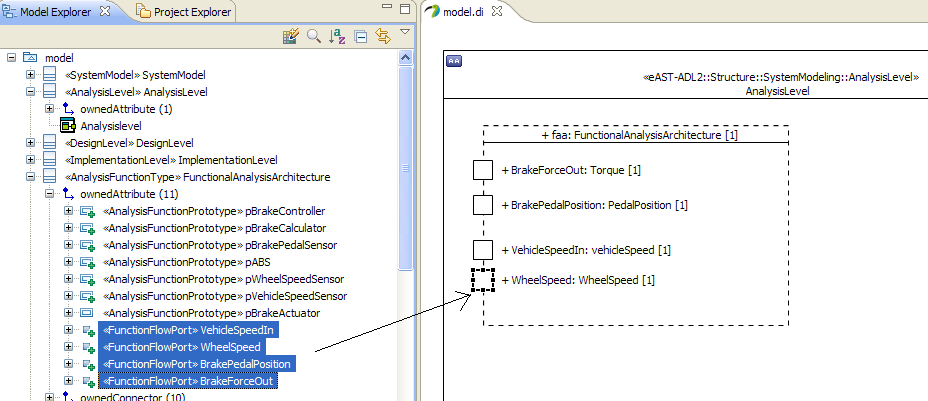


Figure 46: The AnalysisLevel, containing a prototype of the FunctionalAnalysisArchitecture

1. You can change the appearance of the ports to show their icons as done in step 5a for a class.
2. Drag the sub-components of the environment model and their associated (unconnected) functionalports to the pEnv as shown Figure 46.

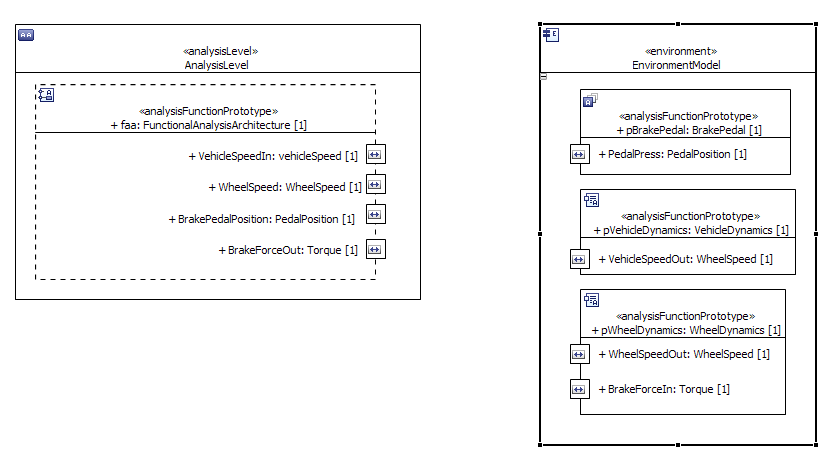


Figure 47: Adding already existing prototypes to a prototype.

1. Connect the appropriate ports using clampConnectors to get the model as shown in Figure 47.

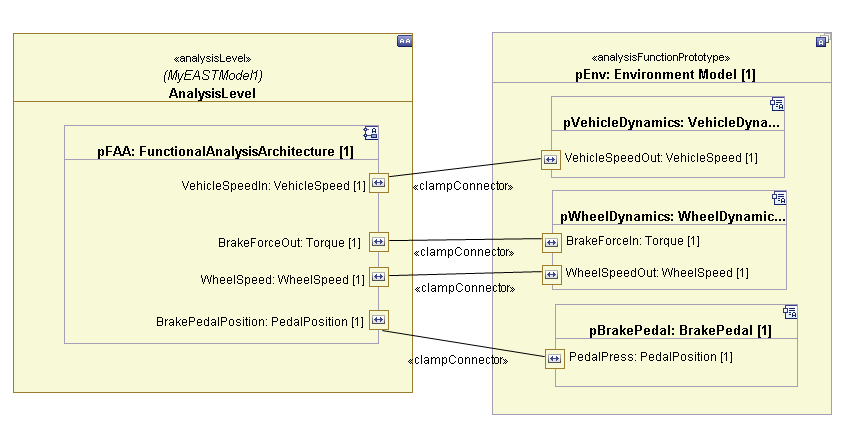


Figure 48: The complete closed-loop AnalysisArchitecture of the Brake-by-wire system

### In Enterprise Architect

Go to the package AnalysLevelElements and open AnalysisLevel diagram.

1. Drag and drop AnalysisFunctionType from the pallette (Alt-6) and call it FunctionalAnalysisArchitecture
   1. Drag and drop AnalysisFunctionType from the pallette (Alt-6) and call it BrakeControllerAFT
   2. Drag and Drop a RangeableValueType from the pallette (Alt-6) and call it TorqueType
   3. Drag and drop two FunctionFlowPort from pallette (Alt-6) and call them according to the example.
   4. Double click one of the ports and look under Property category to set type to TorqueType using the Type (…) button.
   5. Look under the EAST-ADL category to set direction by typing “in” in the direction field
   6. Do the same for the out port.
2. Drag and Drop AnalysisFunctionPrototype onto the FunctionalAnalysisArchitecture. Select “part”. Call the part pBrakeController.
3. Double-click pBrakeController and look under Property category to set type to BrakeControllerAFT using the Type (…) button
4. Right-Click the prototype and select EmbeddedElements- EmbeddedElements; Click Show Owned; Select the ports owned by the type.
5. Connect the ports by clicking FunctionConnector in the Pallete (alt-6), clicking the first port and hold, and release on the second port. Double click the connector and set direction to unspecified.

## Defining DesignLevel elements

### In Enterprise Architect

Add Package DesignLevel containing packages HardwareElements and FunctionalElements. Put stereotype EAPackage on all three. Add a CompositeStructureDiagram in each package.

Add elements corresponding to a Function Design Architecture in the FunctionalElements package. Do according to the Function Analysis Architecture, although using Design Level elements and a refined structure.

1. Add a HardwareComponentType in package HardwareElements and call it HardwareDesignArchitecture.
2. Add a Node in package HardwareElements and call it ECUType.
   1. Add two CommunicationHardwarePin and call them BusLo and BusHi.
3. Add two HardwareElementPrototypes in HardwareDesignArchitecture and call them pECU1 and pECU2.
   1. Double-click pECU1 and pECU2 and look under Property category to set type to ECUType using the Type (…) button
   2. Right-Click the prototype and select EmbeddedElements- EmbeddedElements; Click Show Owned; Select the ports owned by the type.

Allocate Functional elements to hardware elements:

1. Add a DesignLevel and call it myDesignLevel
2. Add prototypes pFDA and pHDA in myDesignLevel, typed by the FunctionDesignArchitecture and HardwareDesignArchitecture, respectively.
3. Add a Allocation element in myDesignLevel
4. Add FunctionAllocations in Allocation

## Using the extension for timing and relating requirements with functions

In this section we will add the timing information for different functions. Relate them with requirements and the associated functions.

* + - 1. Add a package named “TimingPackage” and its class diagram.
      2. Add an ExcutionTimeConstraint from the Pallete,
         1. Rename it from Class0 to “BrakeCaculatorExecutionTime”. This result is shown in Figure 48.

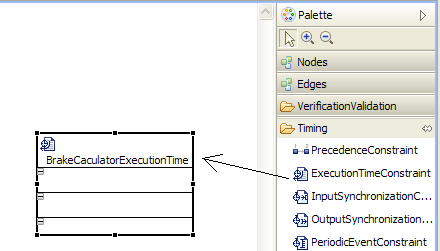


Figure 49 : Adding an execution time constraint

* 1. Add a UML data type,
     1. Name it as CalcUpper
     2. Apply the stereotype “TimeDuration”

Set the value to 3.

* + - * 1. The result is shown in Figure 50

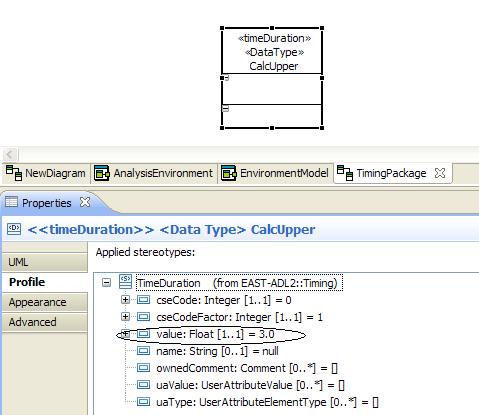


Figure 50 : Adding Time Duration

* 1. Repeat b for CalcLower with value 1
  2. Set the lower and upper time duration of the BrakeCalculationExecution time to CalcUpper and CalcLower as shown in Figure 50.

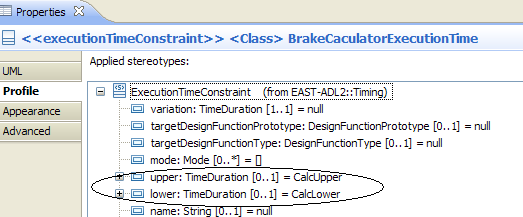


Figure 51 : Setting upper and lower time duration of an execution constraint

1. Repeat step 2 for “BrakeControllerExecutionTime” and “ABSExecutionTime” with the same lower and upper time duration values.
2. Relate Timing Constraints with the requirements.
   1. In the Class Diagram for the timing package, drag the OverallTiming requirement and connect it with the three timing constraints using the “satisfy” link (from the requirements part of the palette) of the Requirements part in the palette as shown in Figure 51.

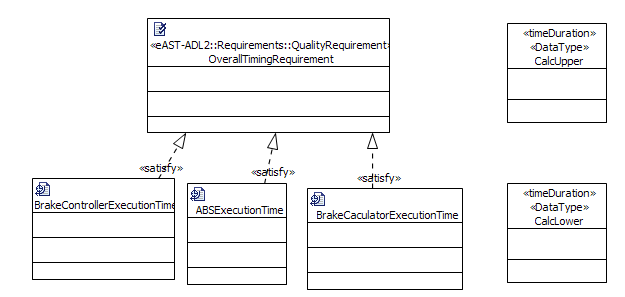


Figure 52: Satisfy links connecting requirements and timing constraints

1. Repeat step 4 for relating functions to the timing constraints as shown in Figure 52. (Note: This can be done in a separate class diagram or in the same as shown in the figure)

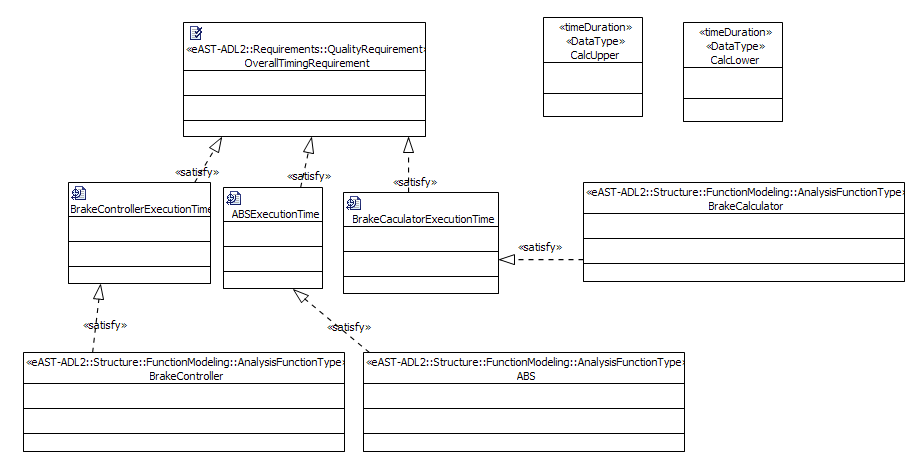


Figure 53: Linking FunctionTypes to their execution constraints.

# Part B: First Steps with EAST-ADL Error Modeling

## Overview

In this tutorial we will show some of the basic concepts of the EAST-ADL Error Model and how to use it. First we will introduce the language concepts for error modeling provided by EAST-ADL. In the second part we will take you through the first steps of creating an error model for a brake-by-wire system. We start from an existing model. You can find it in the workspace navigator under errormodel\_tutorial. We use tool support to automatically create fault trees and FMEA tables based on the EAST-ADL Error Model. In the end of this document you can find a short glossary.

## EAST-ADL Error Modeling Concepts

An Error Model is a specific non-functionality model for fault/failure description. It contains reusable definitions of error logics for a component/system, i.e. which faults may occur and how they propagate throughout the component/system. Associations to the nominal architecture identify which components are covered by the error model. The error model can have different levels of abstraction and granularity according to referenced nominal architecture definitions. For example, at the analysis level, the target components in terms of the logical functions of the application or interface device imply that only the logical failure modes are captured in the corresponding error models. Based on the type-prototype pattern found in AUTOSAR, error models can be hierarchically composed through the use of prototype constructs. This means that a prototype is the occurrence of a certain reusable type. The type also indicates what parts (structure) a given prototype is composed of. See Figure 53 for an example of adding an error model for a nominal architecture model with the EAST-ADL language.



Figure 54: Example of providing error models for each nominal architecture element.

From a system development point of view, one important modeling issue is the ability to separate the definition of faults/failures – including the related behaviors –from the definitions of nominal system behaviors. This is because the fault/failure definitions are essentially only the analytical augmentations of the nominal system architecture model. Although important for safety engineering, such information may be unnecessary during functional analysis, analysis and simulation of nominal behaviors, or code generation. Error models in EAST-ADL are maintained as separate views of the overall system architecture specification and can be traced and extracted automatically for separate safety engineering tasks.

The propagation link (PropagationLink) connects an output port with an input port for error propagation across the target architecture components. EAST-ADL allows dedicated error propagation links, which reflect the error propagation caused by communication links and other relationships in the nominal system model. This separation-of-concerns allows effective error modeling even when nominal component links are not significant, sufficient, or efficient for error propagation.

Within an error model, the syntax and/or semantics of existing external formalisms can be adopted for a precise description of the error logic (errorLogic). The specification captures what output failures of the target architecture component are caused by what faults of this component. This, together with the error propagation links, makes it possible to perform safety simulations and analyses through external analysis tools.

## Tutorial

In this example we create an EAST-ADL Error Model. It describes where errors can reveal themselves in the system and how failures propagate through the system, thus influencing other functions.

In this tutorial we will start with an existing Error Model that you will extend.

1. Open the modeling file **MyEASTModel1.di2** in the project **errormodel\_tutorial**
2. Open the existing **ErrorModelPackage** in the Papyrus Outline, as shown in Figure 54.

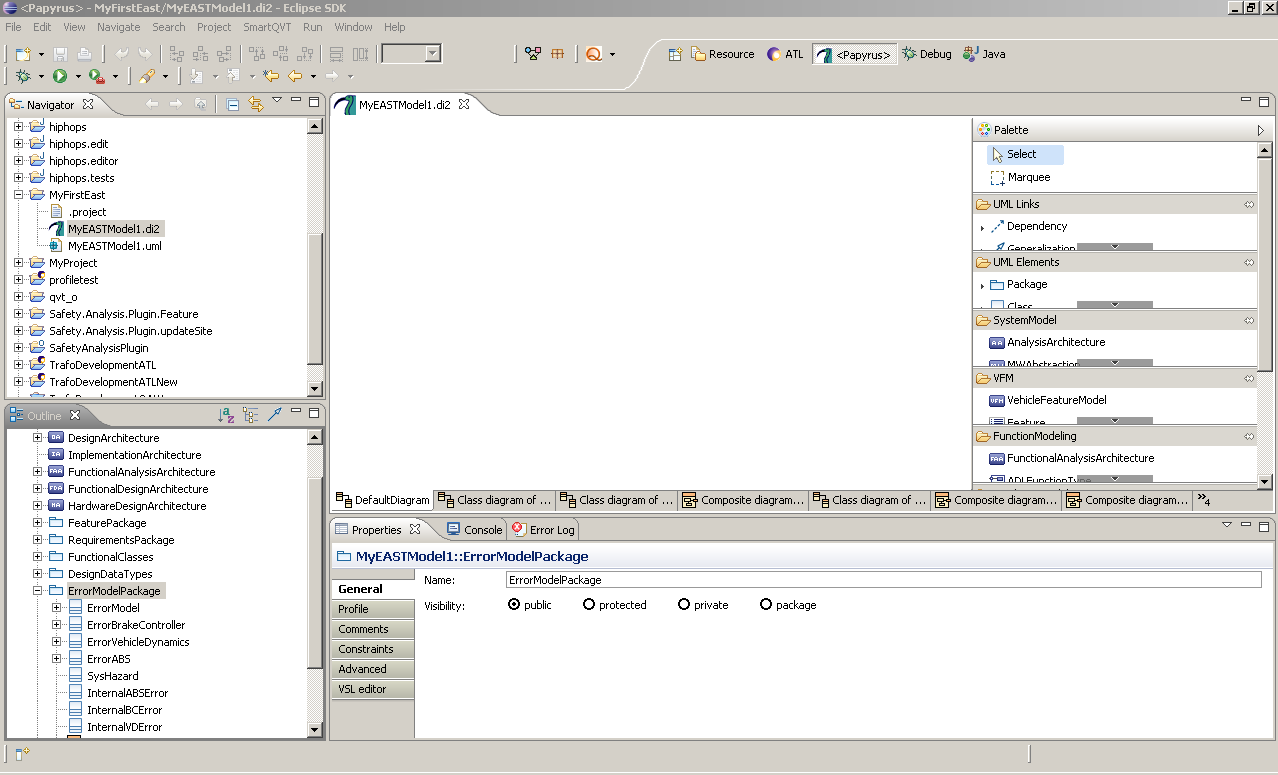


Figure 55: The ErrorModelPackage

1. Open the composite diagram contained in this package, it contains the ErrorModel for the BrakeByWire System, that is almost complete.
2. From the Palette on the right select the ErrorModeling Tools (Figure 55).
3. Create a new ErrorModelPrototype inside the yellow box called ErrorModel.
4. Double-click on the new box to change the name and type of the component to “pErrorWheelSpeedSensor: ErrorWheelSpeedSensor”.

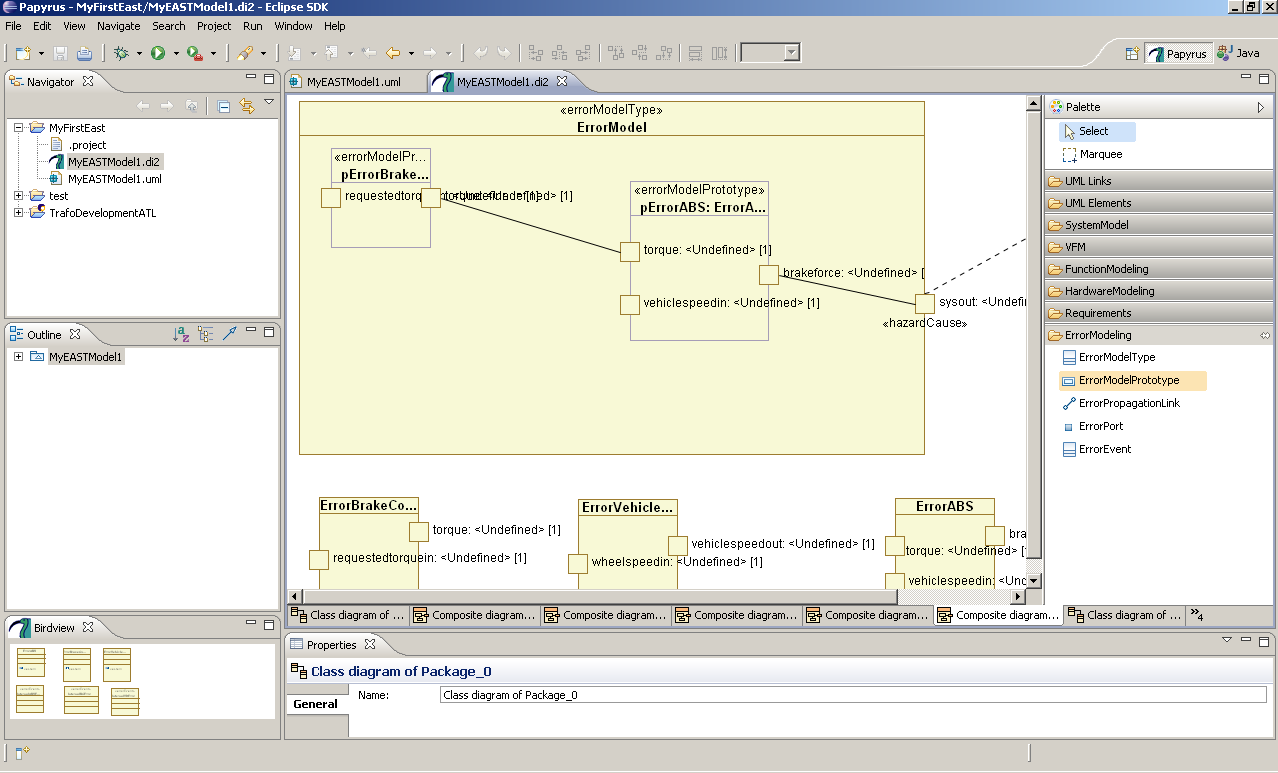


Figure 56: The ErrorModelPrototype in the palette

Ports will now become visible.

* Using an ErrorPropagationLink from the Palette, connect the port wheelspeedout on the pErrorWheelSpeedSensor to the port wheelspeedin of pErrorABS.

It should look as in Figure 56:

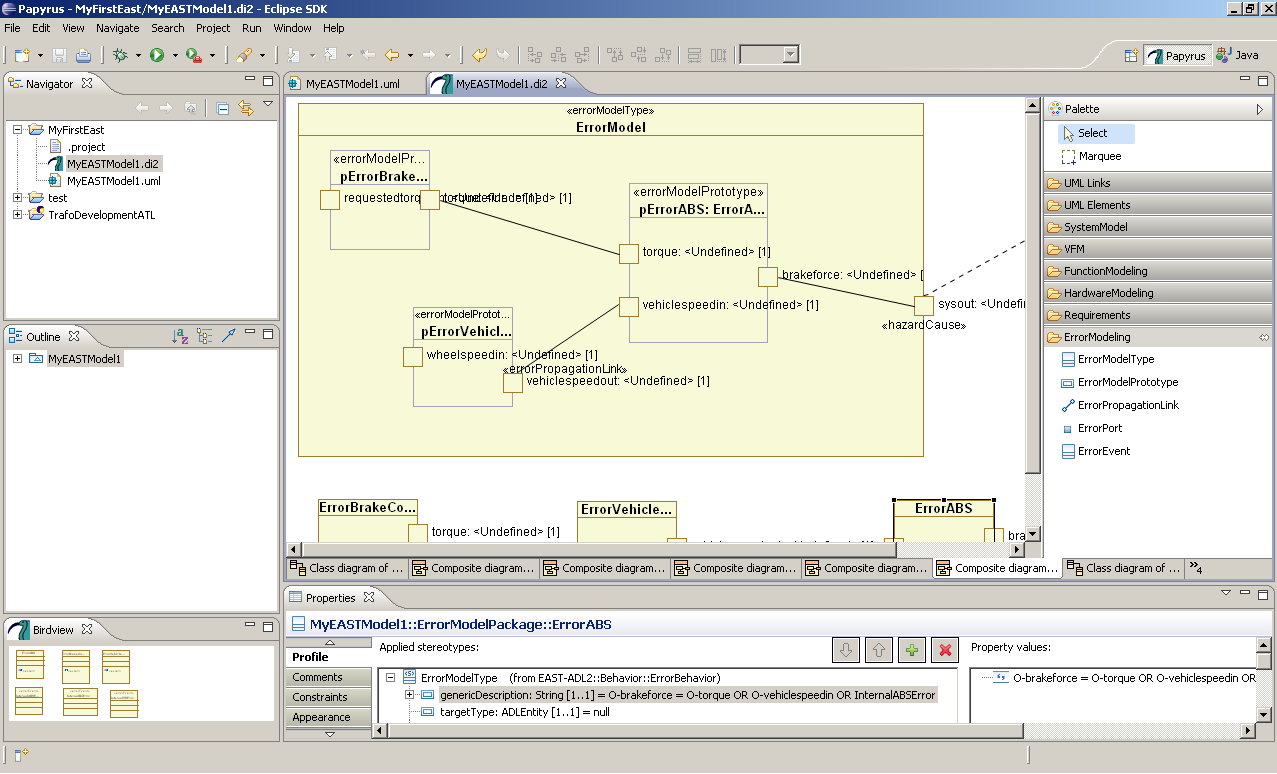


Figure 57: An error propagation connector

We will now check the ErrorPropagation concept. Each ErrorModelType contains a description of how failures propagate from the inports to the outports. The propagation is specified as a string containing logical operators (AND, OR) , EAST-ADL ErrorPorts and InternalFailures.

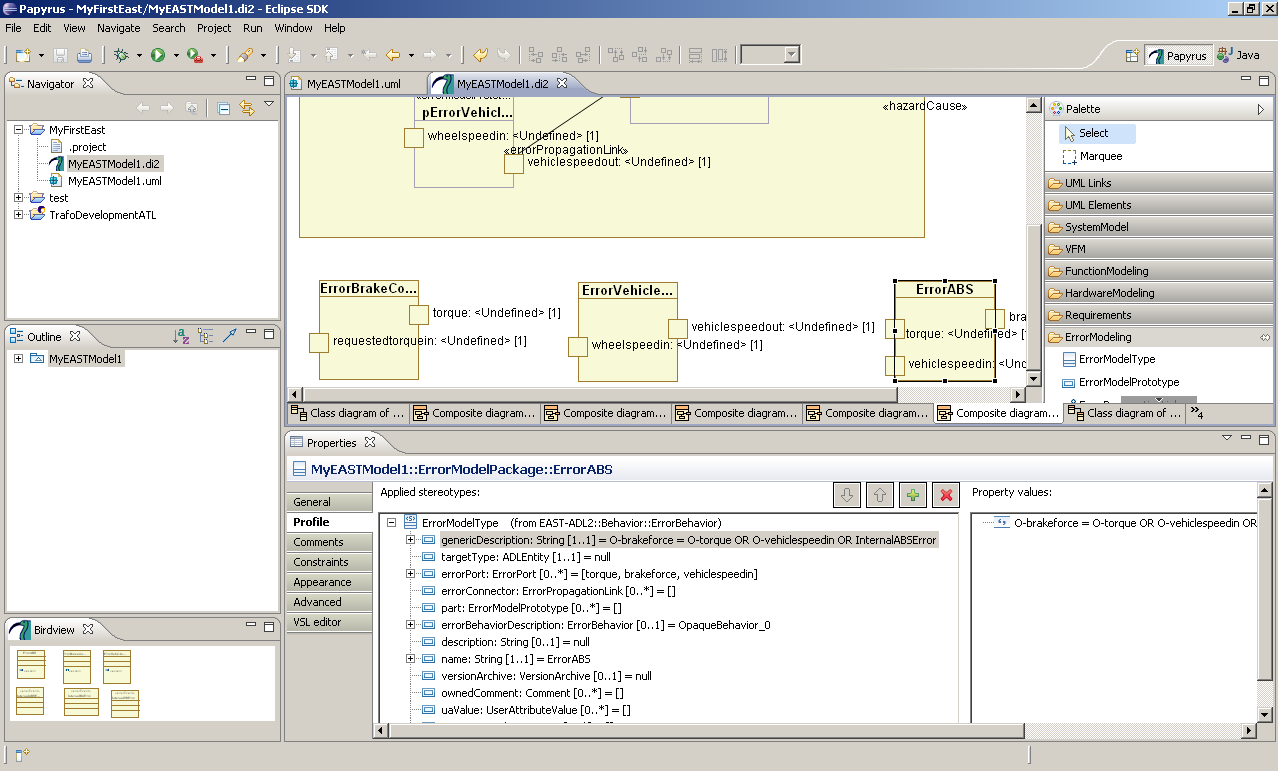


Figure 58: Here is the logic for error propagation defined

Select the ErrorModelType named ErrorABS and select the profile tab. Under genericDescription you can find the string explaining the propagation of errors (Figure 57).

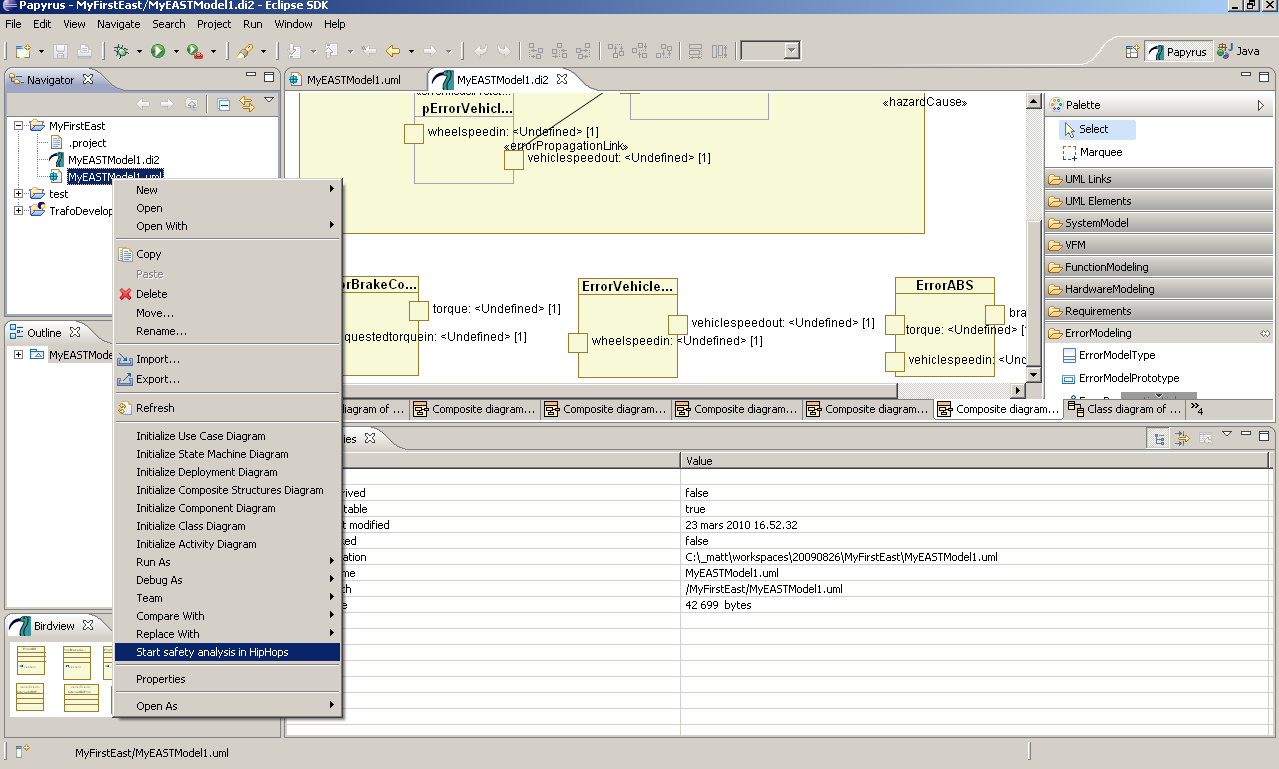


Figure 59: Starting HiP-HOPS from Papyrus

1. Save your model
2. You can analyze your model by selecting the uml file on the upper left hand side. Right click on the uml file and select StartSafetyAnalysis (Figure 58). The FaultTree and the FMEA Tables will be computed and displayed in the browser, as shown in Figure 59

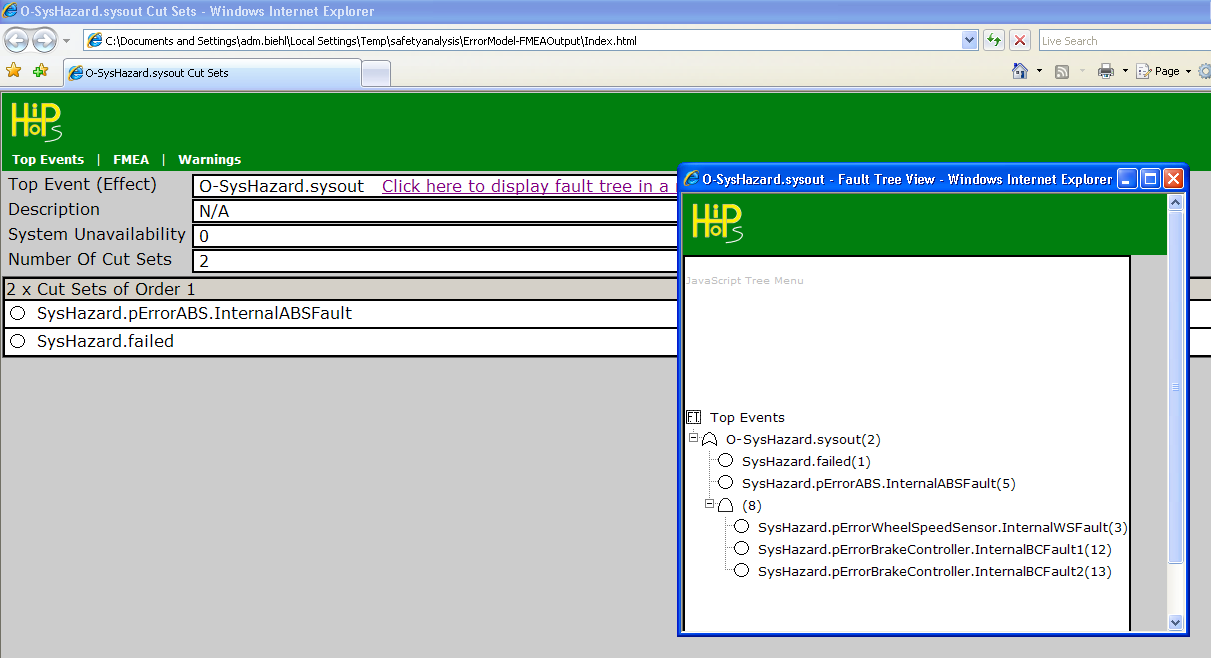


Figure 60: The output from HiP-HOPS, which is displayed inside a webbrowser

The tool has calculated the fault tree for you. The fault tree is an AND-OR tree, that shows with which combination of faults the system fails (see Glossary). The top event representing a system failure is called SysHazard.sysout in this example. You can navigate the tree and unfold its branches.

You can now experiment with the ErrorModel and e.g. change the propagation logic. In Papyrus, open the ErrorModel diagram that we edited before, select an element of type ErrorModelType and select the error logic description as shown in Figure 59. Change the logic description and rerun the analysis tools.

## Glossary (on error-modeling)

**ErrorModel**: a model describing the propagation of failures

**Fault**: A fault is a defect within the system, like the malfunction of a sensor or the bug in the software

**Error**: An error is a deviation from the required operation of the system. A fault may (or not) lead to an error

**Failure**: A failure occurs when the system fails to perform the required function

**Hazard**: A hazard is a situation in which there is an actual or potential danger for people or environment

**FMEA (Failure Modes and Effects Analysis):** FMEA analyzes the failure behaviors of components inductively. It starts with the failures and checks for the causes this failure might lead to in the system.

**FTA (Fault Tree Analysis):** FTA is a deductive technique that starts from the hazard and works backwards to determine its cause. It is a classical top down approach, where the top event is the hazard that is split up into its causes. To model which of these causes are necessary and sufficient, they can be connected using AND and OR operators. This technique is used recursively, so a tree of causes is created, with a hazard as root and leaves representing causes. The tree can then be used for determination of the minimal cutest, the minimal combination of leaves necessary for the hazard to occur. The tree can also be used to determine the probability of the top event.

HiP-HOPS (Hierarchically Performed Hazard Origin and Propagation Studies):  is a method for safety analysis through a combination of several classical techniques (i.e., FFA, FMEA, and FTA). It relies on a hierarchical system description and provides the capability to synthesize an overall fault tree for an entire system automatically. The HiP-HOPS method aims to enable an integrated safety analysis and systems design and hence to make safety analysis up-to-date along with design refinement and evolution. The safety analysis starts with an exploratory FFA analysis that identifies system hazards based on a model of system functions. This functional failure analysis method extends the classical FFA method by also considering plausible failure combinations and suggestions for hazard control. The outcome is a table listing possible function failures, their contributing factors, effects, severity, and recommendations for error detection and recovery.

# Part C: Transformation Between EAST-ADL And MATLAB/Simulink®

A demo Simulink model of an ABS system available from the MathWorks has been used as a reference to develop this example. The original Simulink model is available by entering:

sldemo\_absbrake

at the MATLAB command-line in any recent version of MATLAB. The model is displayed below: mathworks

Figure 61: The ABS model from the MATLAB example files

In short, this model calculates a deviation from a desired relative slip, with feedback from the actual Relative Slip. The brake torque is applied using an ideal bang-bang controller, proportional to the deviation from the desired relative slip. Inside the Wheel Speed block, a hydraulic lag is modeled, making it a dynamical system. In this tutorial, the original model is restructured. Quoting from the MATLAB/Simulink documentation

In an actual vehicle, the slip cannot be measured directly, so this control algorithm is not practical. It is used in this example to illustrate the conceptual construction of such a simulation model. The real engineering value of a simulation like this is to demonstrate the potential of the control concept prior to addressing the specific issues of implementation

In the original model, the border between the controller and the environment is not very clear. The output from the Relative Slip block is used both as an environment function, to get the mu friction coefficient, and as a measured variable in the controller. Another note is that there is no such thing as a brake-pedal, or a brake-signal, the model assumes maximum possible brake force at all time.

To structure the model, subsystems are created, as in Figure 61, and brake functionality (pedals, brake calculator) is added Each of these subsystems corresponds to an AnalysisFunction. These subsystems will be converted to AnalysisFunctions.

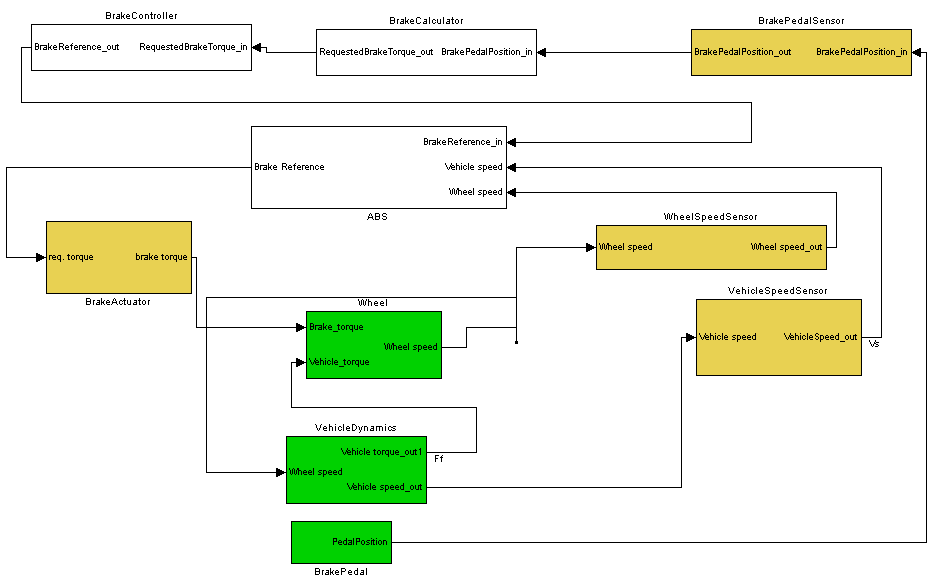


Figure 62: The model in Figure 60, structured as subsystems. Compare with the FAA in Figure 27 on page 24

Some of the subsystems represents physical entities, e.g. the brake pedal and the vehicle model. They will be modeled inside the environment model in EAST, however this partitioning is done in Papyrus. In Figure 61, the physical systems are marked green. FunctionalDevices (yellow in Figure 61) are modeled to transform between physical signals and logical signals, they are modeled inside the FunctionalAnalysis architecture in Papyrus.

When this restructuring of the model has been carried out, they need to be added to the Ecore library in Simulink. This tutorial assumes that you have installed the Simulink plugin, and run the “ecore\_init” script from the MATLAB command prompt. Right-click a subsystem, and choose “Add to Ecore library” (this option will only be available for subsystem blocks). The block will now be moved to the FunctionTypes.mdl library file. The original block will then be replaced with this library block, to show the operation has been successful; the color is changed to light-blue. It is important to add blocks to this library this way, since many other actions are performed in the background, e.g. renaming ports, adding unique ID:s (UUID:s) to blocks and ports. All the blocks in the model needs to be added this way to the Ecore library.

By encapsulating all entities into library subsystems, the model is now modified in such a way that it can be saved as an Ecore file, compliant to the meta-model that has been defined. It can then be saved as an .simulink file, this is done in Figure 63.

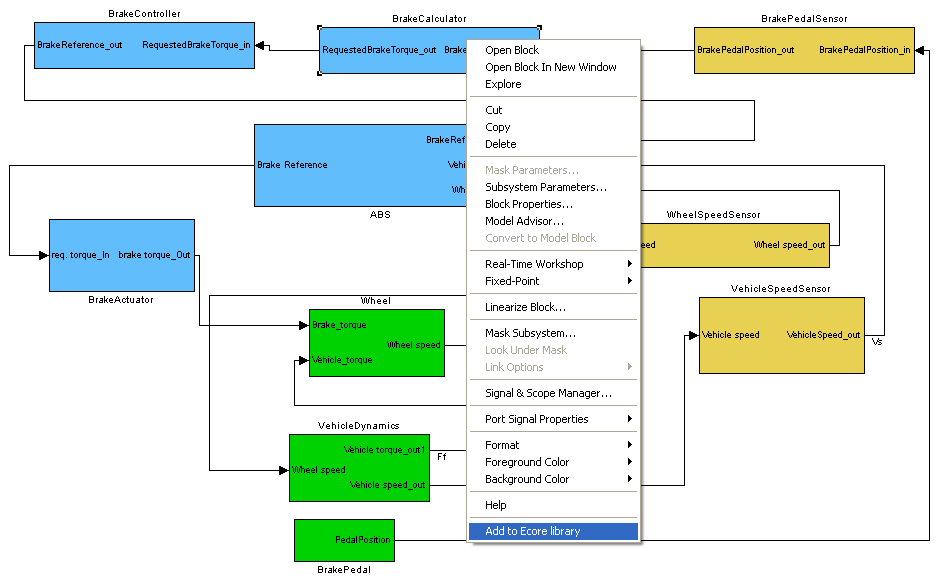


Figure 63:How to add blocks to the Ecore library

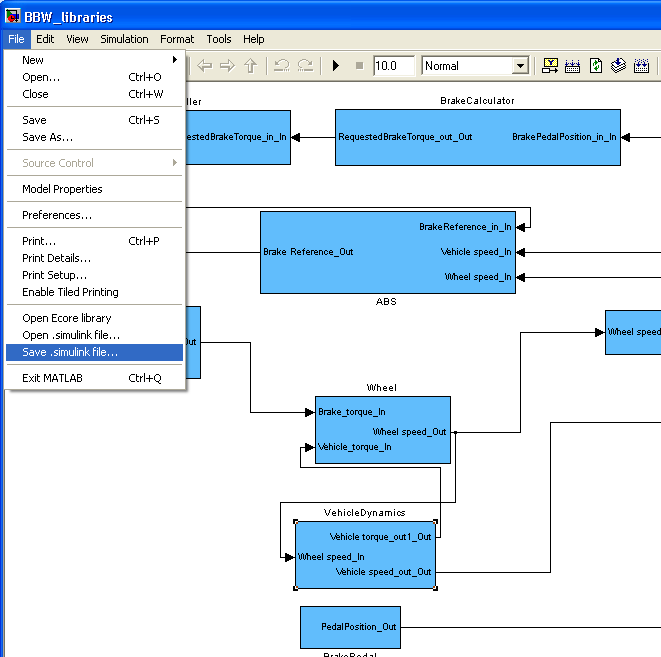


Figure 64: Saving .simulink files

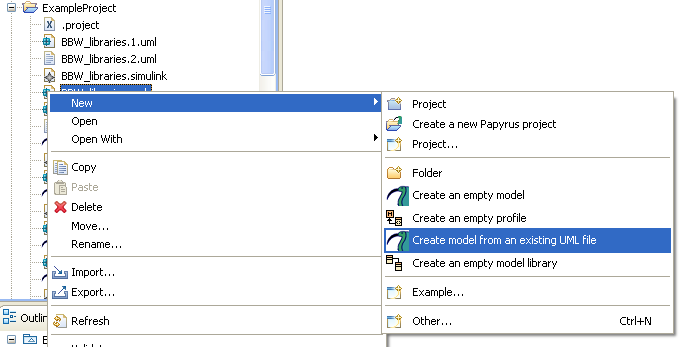
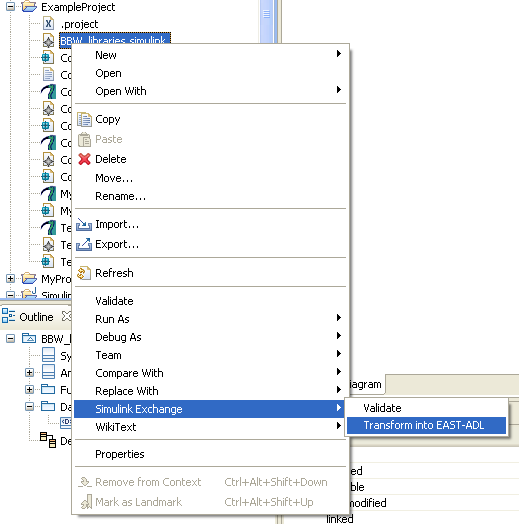


Figure 65: Two steps to create an EAST-ADL UML file from a .simulink file

To import it into Papyrus, drag the file into Eclipse, and right-click the file (see Figure 64, above). Hit refresh, and you will see that three files were created: Modelname.1.uml, Modelname.2.uml, and Modelname.uml. The two first files are intermediate files and can be ignored. By right-clicking the Modelname.uml file, and choosing “Create model from an existing UML file”, a complete Papyrus model is made, see Figure 65.

The plugin will be further developed and maintained after the ATESST2 project. The latest version of the plugin can be downloaded from the link below:

<http://code.google.com/p/kth-simulink-exchange/>

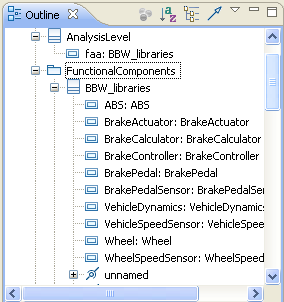


Figure 66: The model structure in Papyrus

1. http://www.eclipse.org/modeling/mdt/papyrus/ [↑](#footnote-ref-2)
2. http://maenad.eu/public\_pw/Tooling/maworkbench\_AHQ\_0.8.2-eastadl2.1.10.zip [↑](#footnote-ref-3)