MAENAD

Grant Agreement 260057

Model-based Analysis & Engineering of Novel Architectures for Dependable Electric Vehicles

Report type — Deliverable D5.3.1

Report name — EAST-ADL implementation in MetaEdit+

Dissemination level	**PU**
Status	**Final**
Version number	**4.0**
Date of preparation	**2014-02-17**

Authors

Editor	E-mail
Janne Luoma	janne@metacase.com

Authors	E-mail
Janne Luoma	janne@metacase.com
Juha-Pekka Tolvanen	jpt@metacase.com

Reviewers	E-mail
Sara Tucci	sara.tucci@cea.fr
Henrik Lönn	henrik.lonn@volvo.com

The Consortium

Volvo Technology Corporation (S) Centro Ricerche Fiat (I)

Continental Automotive (D) Delphi/Mecel (S) 4S Group (I)

ArcCore AB (S) MetaCase (Fi) Systemite (SE) CEA LIST (F)

Kungliga Tekniska Högskolan (S) Technische Universität Berlin (D) University of Hull (GB)

Revision chart and history log

Version	Date	Reason
0.1	28.12.2010	Initial version
0.2	2.2.2011	Updated metamodel, references
0.3	26.4.2011	Revision, minor updates
0.4	12.5.2011	Minor updates based on feedback from VTEC
0.5	10.8.2011	Updated for EAST-ADL2 M.2.1.9 METAEDIT20110622
1.0	30.8.2011	Finalized for deliverable
2.0	29.8.2012	Updated version
3.0	28.2.2013	Updated version with latest metamodel
4.0 prel	15.1.2014	Updated for EAST-ADL M2.1.12, added Section 4
4.0	17.2.2014	Updated based on reviews

Approval	Date
Henrik Lönn	2014-02-20

Table of contents

1 Introduction

MetaEdit+ is a mature, platform-independent language workbench for domain-specific modelling, supporting graphical, matrix and table-based modelling languages. In MAENAD project MetaEdit+ has been used to implement the metamodel of EAST-ADL, along with constraints, notation and various generators.

MetaEdit+ for EAST-ADL provides graphical editors, matrix editors, table editors, various browsers and related generators for EAST-ADL. MetaEdit+ scales to cases with hundreds of users and gigabytes of models. Repository of MetaEdit+ is used to store both EAST-ADL metamodels and models, including both conceptual and representational (abstract and concrete syntax) data for both. A repository can consist of an unlimited number of projects, each of which can hold over 4 billion persistent objects.

This document describes the implementation using Hardware Architecture of EAST-ADL (version M2.1.12 [1]) as an example. First we show the sample of the language in use (Section 2) and then (Section 3) describe its implementation details, covering:

- Metamodel and related rules
- Consistency checks and checking reports
- Notation
- Generators

Finally Section 4 outlines other modeling capabilities and generators available and developed by MAENAD partners.

For starting to use EAST-ADL, there is a separate tutorial describing the use of EAST-ADL in MetaEdit+ [2]. This guide covers also other parts of EAST-ADL than modeling Hardware Architecture, such as functional architectures, requirements models, feature models, dependability, environment models etc. There are also generators for tracing, checking, and various exports like Simulink, HIPHOPS, ReqIF etc.

In addition to EAST-ADL manuals there are also User Guides on using MetaEdit+ tool itself [3]. This User Guide's describe how to create and modify the metamodel of EAST-ADL, its notations and generators.

1.1 Downloading and installing MetaEdit+ for EAST-ADL

For exploring EAST-ADL tutorial thoroughly, you need MetaEdit+ tool and EAST-ADL repository. See EAST-ADL tutorial at http://www.metacase.com/papers/MetaEditPlus_Tutorial_for_EAST-ADL.pdf for instructions.

After installing MetaEdit+ tool you should extract the EAST-ADL repository file into the same folder you have installed MetaEdit+. As default MetaEdit+ working directory should contain '*demo*' repository coming with the standard MetaEdit+ installation. In Windows this working directory is under your '…*YOUR USERNAME*\My Documents\MetaEdit+ 5.0'. After extracting the zipped repository you should have two separate folders like: '*demo*' and '*EAST-ADL*' repositories on the same level (both of these repositories should have subfolders named as '*areas*', '*backup*' and '*users*' and two files '*manager.ab*' and '*trid*'). Make sure you have read and write rights to the directory. In case that you already have existing EAST-ADL repository (downloaded earlier), see Section 7 in [2] for details.

2 Modeling Hardware Architecture with EAST-ADL

MetaEdit+ provides tool support for EAST-ADL language with graphical editors, matrix editors, table editors, various browsers and related generators. Figure 1 shows a sample of hardware architecture diagram of EAST-ADL in MetaEdit+.

The modeling editor provides basic editing features (move, zoom, grid etc) and editing functionality related to modeling languages such as copy-&-paste, paste special, refactoring, trace, unlimited redo/undo, auto layout etc.

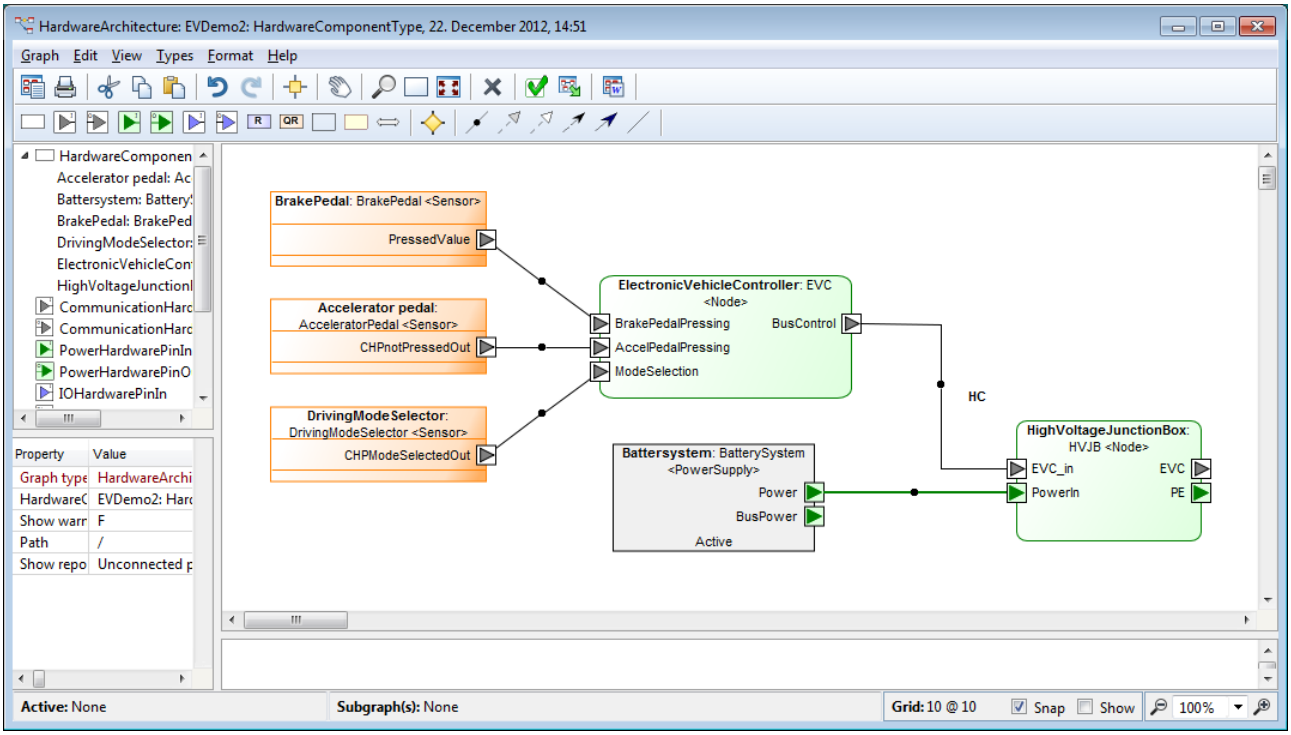


Figure 1: Example of Hardware Architecture Model in MetaEdit+.

MetaEdit+ provides also other tools for modeling work including:

- Editing models (Matrix Editor, Table Editor)
- Browsing models (Graph, Type, Object Browsers)
- Generators for configuration, code, documentation, metrics, etc.
- Multi-user support supporting simultaneous users (even within the same diagram)
- Multi-platform support
- API, import and export tools

Functionality of these tools is described in detail in [3].

3 Hardware Analysis Description: metamodel

This section describes the implementation of hardware architecture language of EAST-ADL.

3.1 Language concepts

EAST-ADL language is specified as a metamodel, covering the language concepts (objects, relationships, roles, ports and properties) as well as their connections. Figure 2 shows the main objects of the metamodel in Hardware Modeling as been implemented in MetaEdit+.

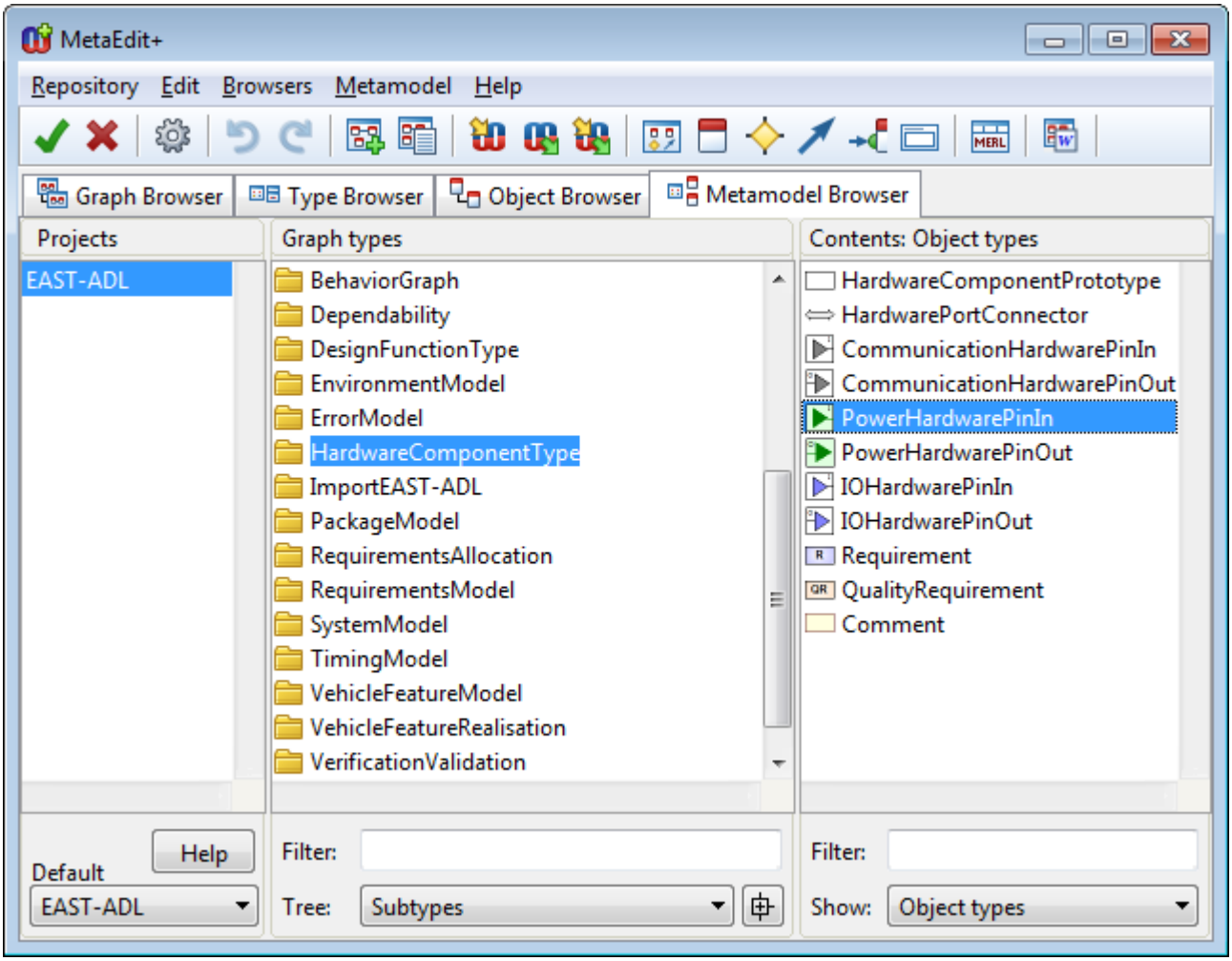


Figure 2: Main modeling objects in Hardware Architecture

For each concept of the metamodel, such as for selected ‘PowerHardwarePinIn’ there is a separate definition describing its properties, constraints and notation. Figure 3 shows the definition of the ‘PowerHardwarePinIn’ such as that Pins have four properties and that ‘IsGround’ is specified as Boolean property whereas ‘Name’ as identifier is specified as a string.

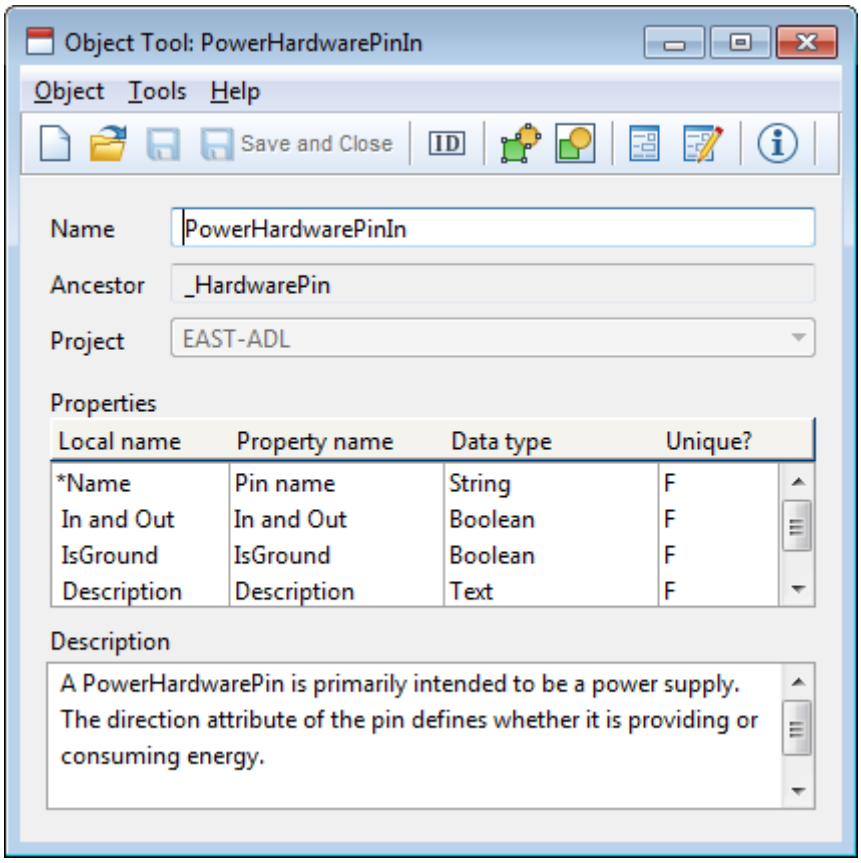


Figure 3: Definition of PowerHardwarePinIn

3.2 Notation

Every language, like EAST-ADL [1] includes also notational definitions and they are specified for each concept of the language. Below a notational symbol is specified for the PowerHardwarePinIn.

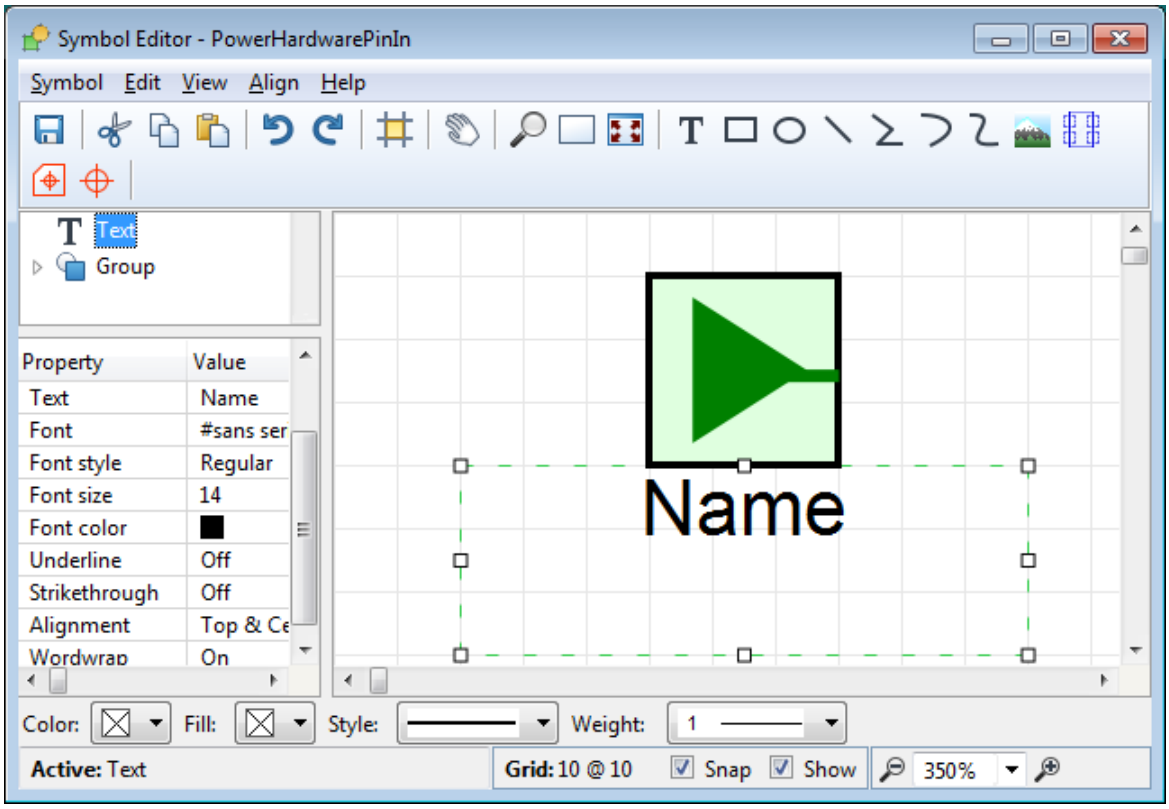


Figure 4: Symbol definition for PowerHardwarePinIn

3.3 Constraints and rules

EAST-ADL specification [1] includes also a large number of rules and constraints that specify which kind of models are complete, consistent and correct. In [1] most of these rules and constraints are written in plain English. These rules are coded and formalized so that they are followed during modeling, generators and interchange.

Examples of such rules include uniqueness, such as pins must have a unique name within a type specification, and connections between power are possible only between of different direction but same voltages. Thus for example connections between IO and power are not possible among the prototypes. These rules are also defined into the metamodel using the constraints or by implementing model checks using generators as discussed below.

In addition to rules that are part of the metamodel definition itself, the EAST-ADL implementation in MetaEdit+ includes generators that check the correctness and completeness of the models. These checks are implemented as generators and they are available to be executed when checking is needed or by annotating the checking results directly in the model or in the LiveCheck pane integrated to the Diagram Editor tool.

Checking reports include:

- Warning if prototypes are left undefined (untyped)
- Warning on using same values for multiple objects (e.g. same name for several connections)
- Warning on empty or illegal identifying elements (e.g. kind of quality requirement)
- Informing on serially connected sensors / actuators
- Informing on non-defined wire definitions applied in HardwarePortConnector
- Incompatible pin types in IOHardware based connections (e.g. analog to digital)
- Informing on duplicated wire definitions in HardwarePortConnector
- Unconnected pins

The results of the checking are shown in a separate window or directly in the hardware architecture diagram by annotating the model elements. When the result of the checking is shown textually in the output window a link is provided from each warning to the respective model element, like shown in Figure 5. This enables tracing from model elements with errors to the actual model element.

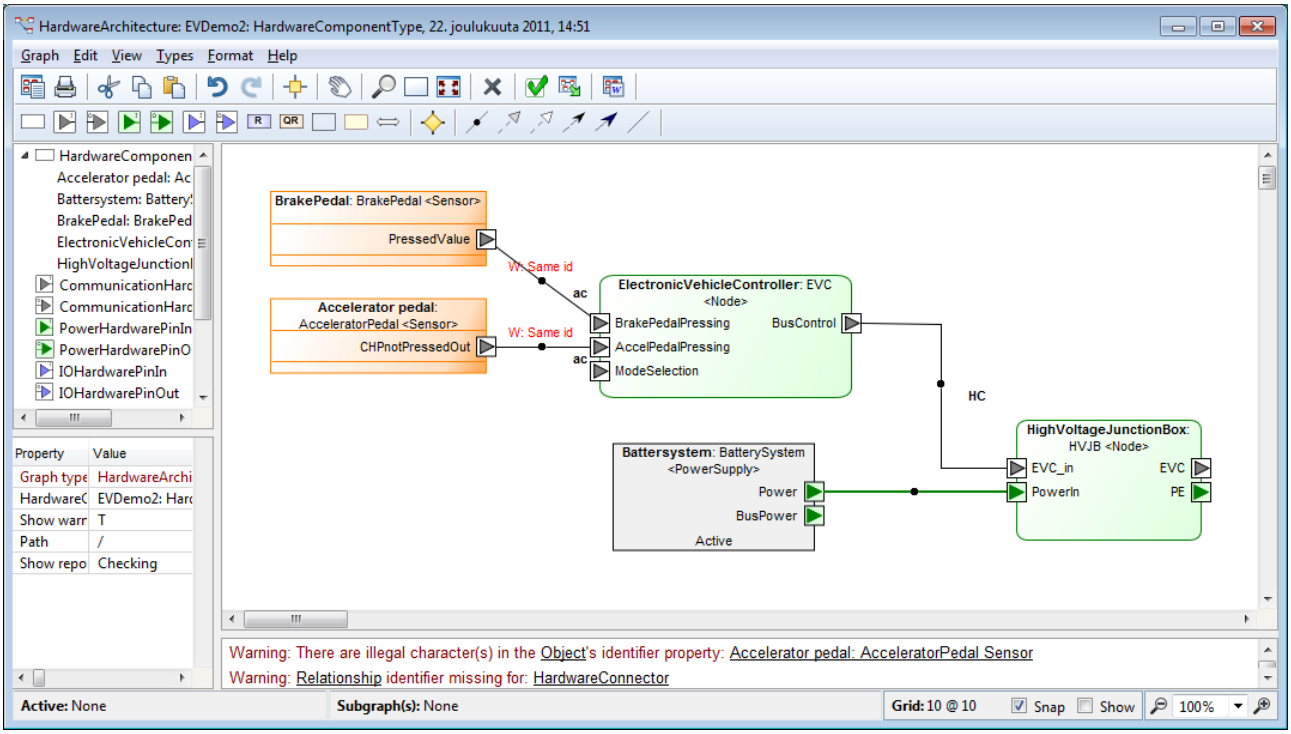


Figure 5. Error annotation and checking

Generators are also used to visualize allocations of functional prototypes to the hardware architecture. After allocation matrix is created the hardware architecture can visualize allocations (show allocated prototypes is set on in graphs properties). The figure below shows such visualization: the white small rectangles inside the hardware element are the allocated functions. This example is exported using export to PNG function.

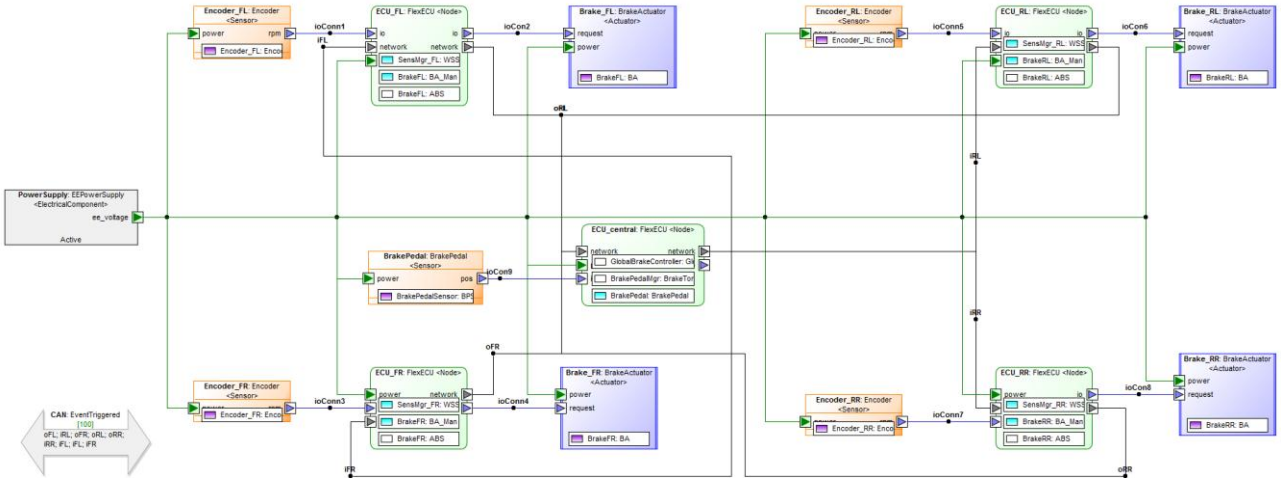
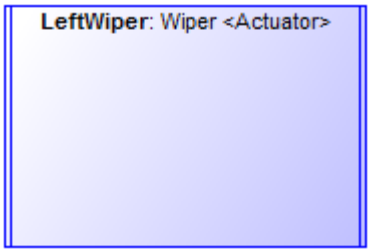
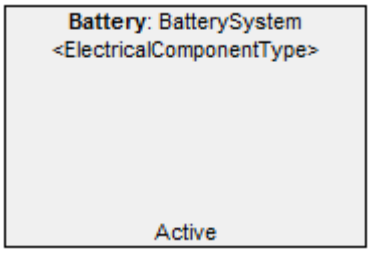
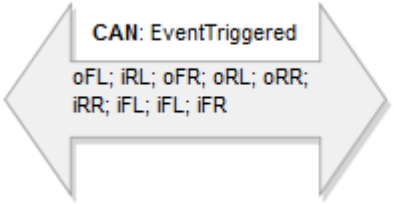


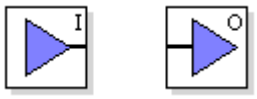
Figure 6. Visualizing allocated functions within hardware architecture description

3.4 Notation of the Hardware Architecture

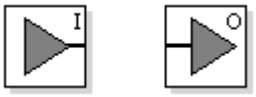
To enable creating, editing and reading the models the language is supported by a graphical notation as follows:

Language concepts	Representation of the concept
The Actuator is the element that represents electrical actuators, such as valves, motors, lamps, brake units, etc.	
ElectricalComponent denotes a power source that may be active (e.g., a battery) or passive (main relay).	
The Node element represents an ECU, i.e. an Electronic Control Unit, and an allocation target of FunctionPrototypes.	
Sensor represents a hardware entity for digital or analog sensor elements.	
Hardware connectors represent wires that electrically connect the hardware components through its ports.	
HardwarePortConnector represents a logical connection that carries data from any sender to all receivers.	

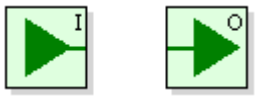
IOHardwarePin represents an electrical connection point for digital or analog I/O. Blue triangle shows the direction of the flow. Relationships between these pins are shown with solid blue line.



The CommunicationHardwarePin represents the hardware connection point of a communication bus. Grey triangles show the direction of the flow. Relationships between these pins are shown with solid black line.



PowerHardwarePin represents a pin that is primarily intended for electrical component, either providing or consuming energy. Green triangles present the direction of the energy. Relationships between these pins are shown with thick green color.



Comment object allows you to add visual description to the model and connect it with any other object in the model.

3.5 Generators

Generators are used to produce various outputs, such as documentation, checking and metrics. Some of the generators are specific to hardware architecture of EAST-ADL. These include:

- Finding out all prototypes using the current hardware architecture type
- Tracing all realization relationships from hardware to requirements
- Tracing all satisfy relationships from hardware to requirements
- Unconnected pins reports pins available but not used

A number of other generators are also accessing hardware architectures, like producing AUTOSAR based on allocations done between functional elements and hardware elements, or producing EAXML for tool interchange.

4 Modeling support for other parts of EAST-ADL

In addition to specifying hardware architectures, EAST-ADL can be used to cover several other aspects of automotive architecture, such as functions, behavior, safety etc. These models can be then used for various analysis and code/configuration generation needs.

4.1 Modeling support

MetaEdit+ provides editors for other parts of EAST-ADL. Figure 7 below shows one of the main browsers of MetaEdit+ describing different parts of EAST-ADL been covered.

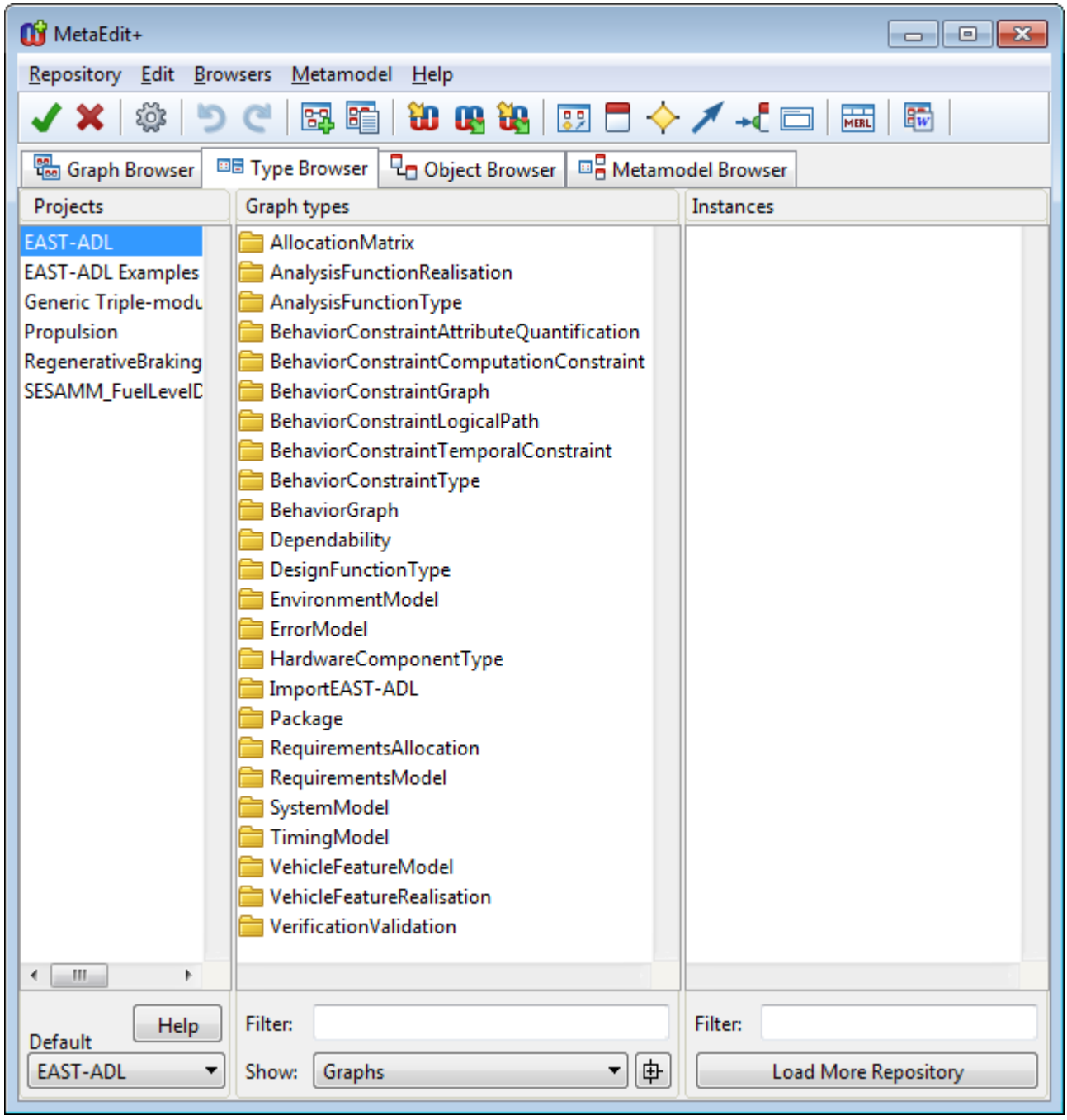


Figure 7. Modeling support in MetaEdit+ for EAST-ADL.

4.2 Generators, analysis support and interfaces

The various models created with MetaEdit+ can be used for different needs. Depending on the model different generators are been developed including:

- EAXML: generators from Package produces EAST-ADL models into EAXML format
- ReqIF: Requirements can be transformed into ReqIF format for exchange with requirements management tools
- Excel: Requirements can be managed in Excel or similar tool and then imported for EAST-ADL for further refinement (like satisfy links to functions etc).
- Simulink: Functional Design can be transformed into Simulink models and API calls
- HIPHOPS: Behavioral descriptions can be translated to HIPHOPS for safety analysis
- Error models for initial dependability analysis can be created from nominal functional architectures
- SPIN: BehaviorConstraints can be translated to SPIN
- UPPAAL: Behavioral descriptions can be translated to UPPAAL
- AUTOSAR: based on allocations between hardware and functions mappings to AUTOSAR architecture (software components, their ports and runnables) can be created
- Test cases and data can be produces from models
- Word: documentation can be generated to RTF and Word
- HTML: documentation can be generated into HTML files

All these generators are open for modifications using MetaEdit+. Users may then extend and modify any of the generators fitting to their specific needs.

In addition to generators also interfaces are available to programming environments like Visual Studio and Eclipse.

5 Conclusions

This document described MetaEdit+ support for EAST-ADL using Hardware Architecture modeling as an example. The tool support covers EAST-ADL concepts, constraints rules, notation, checks and generators. These definitions can be inspected and modified with MetaEdit+ Workbench. Any change to the language definition will update the models made and previous work will not be lost due to language evolution.

In addition to modeling editors, MetaEdit+ provides various browsers, generators, traceability, multi-user support, native UI support for various operating systems etc. To learn more about MetaEdit+ you are encouraged to study MetaEdit+ User's Guide that comes with the installation of MetaEdit+. The installation package includes also other manuals for system administration (especially for multi-user use) and for language creation using MetaEdit+ Workbench. Web pages at http://www.metacase.com provide further information: there you will find user references, discussion forums, downloadable white papers, and FAQs.

6 References

[1] EAST-ADL Domain-Model Specification, Version 2.1.12, 2013. Available at http://www.east-adl.info.

[2] MetaCase, EAST-ADL tutorial, MetaCase Document No. EAT-5.0, January, 2014, available at: http://www.metacase.com/papers/MetaEditPlus_Tutorial_for_EAST-ADL.pdf

[3] MetaCase, MetaEdit+ User Manuals, http://www.metacase.com/support/50/manuals/