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	**Intermediate**
Version number	**2.0**
Date of preparation	**2012-08-29**

Authors

Editor	E-mail
Janne Luoma	janne@metacase.com

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

The Consortium

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

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

MetaCase (Fi) Pulse-AR (Fr) 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

Table of contents

1 Introduction

MetaEdit+ modeling and code generation tool supports different models of EAST-ADL, such as feature modeling, function architecture modeling, error modeling, dependability modeling etc. as well as various model checking, error annotation, and generator features.

This document describes the implementation of Hardware Architecture Modeling Language of EAST-ADL M2.1.10 [1]. First we show the sample of the language in use (Section 2) and then in section 3 describe its implementation details, covering:

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

There is a separate tutorial describing the use of EAST-ADL in MetaEdit+ [2] as well as User Guides on using MetaEdit+ tool itself [3]. These User Guide's describe both how to use modeling languages like EAST-ADL as well as how to create and modify the metamodels, notations and generators defining MetaEdit+ based modeling tool support.

2 Hardware Architecture Description: Modeling with EAST-ADL

MetaEdit+ provides tool support for EAST-ADL language with graphical modeling editors 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.

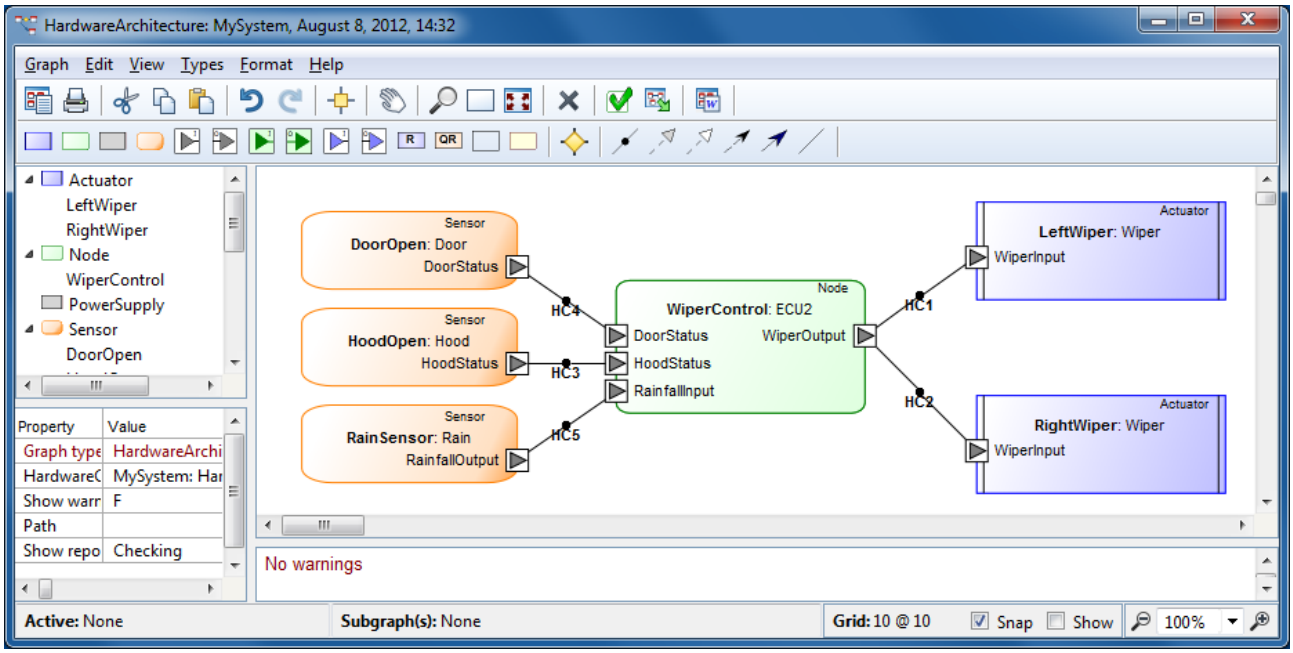


Figure 1: Example of Hardware Architecture Model in MetaEdit+.

MetaEdit+ provides also other tools for modeling work including:

- Editing models (Matrix Editor, Table Editor)
- Browsing models (Graph, Type, Object Browsers)
- Generators for 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 Metamodel and related rules

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 describes the main objects and their properties of the metamodel as been implemented in MetaEdit+.

The metamodel is illustrated here in separate figures just for representation purposes as factually it is one single definition. The metamodeling language used here to present the metamodel is described in [4].

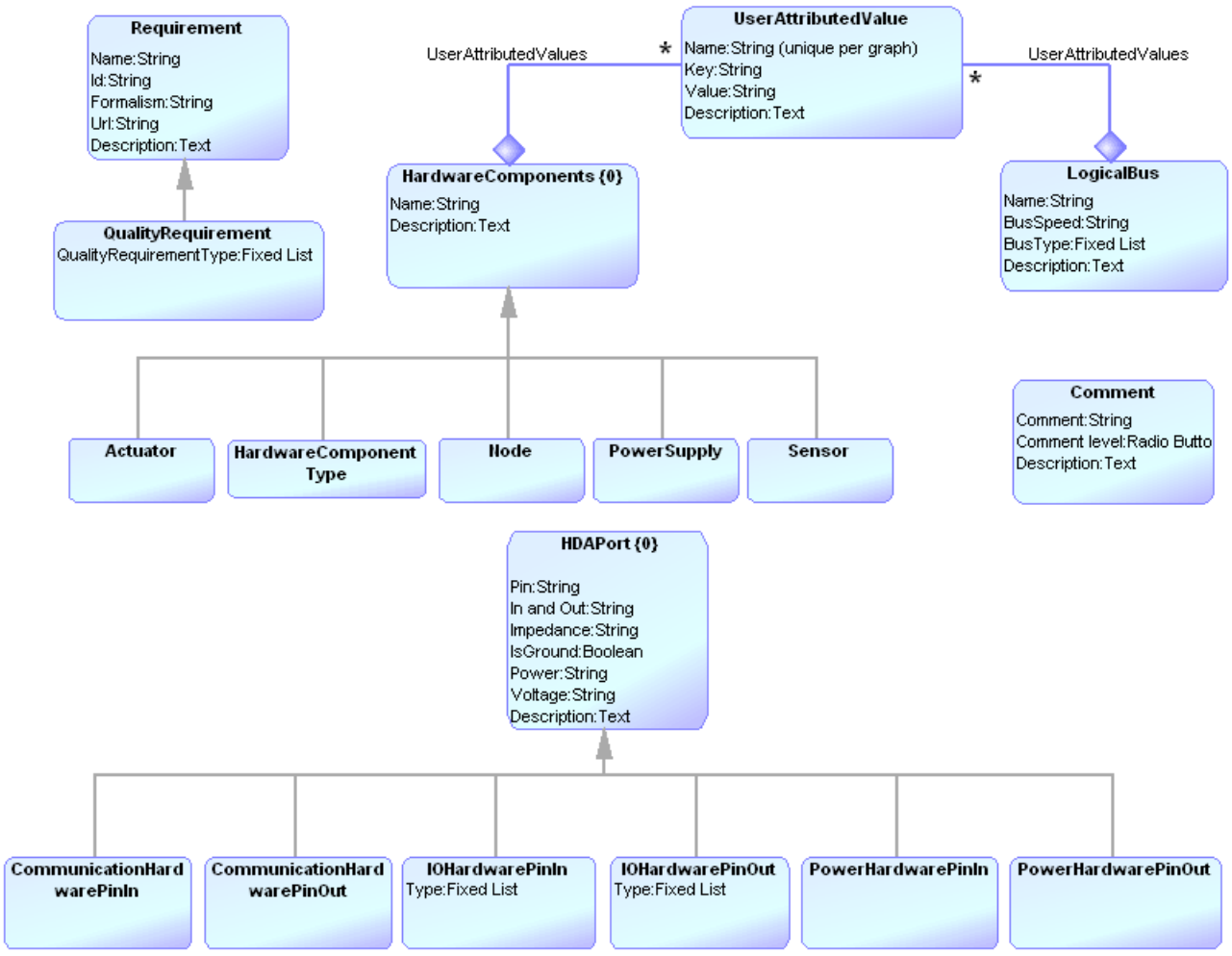


Figure 2: Main modeling objects in Hardware Architecture Diagram

In addition to the modeling of prototypes, each graph, which is usually presented as a diagram, has property that refers to type definition: aka to Hardware Component Types as follows:

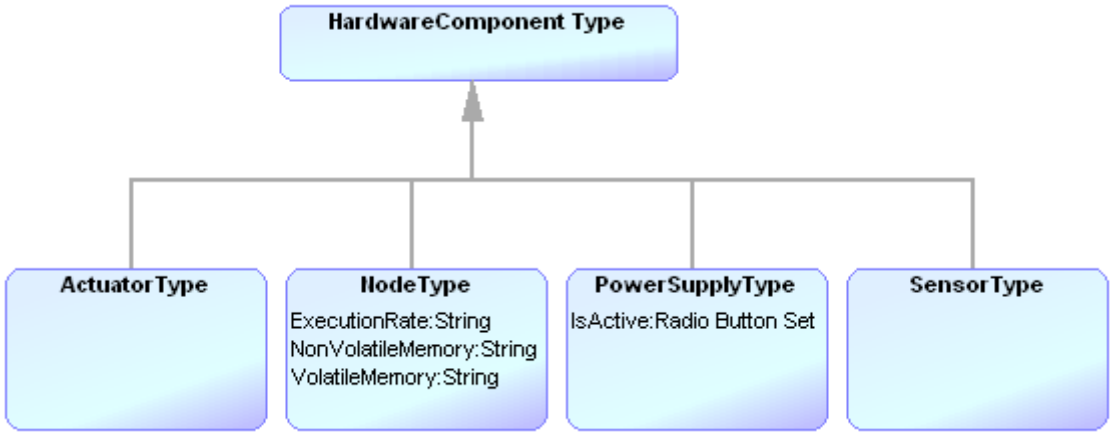


Figure 3: Metamodel elements for type definition

The EAST-ADL specification [1] also includes various connections between the model elements. These are also specified in MetaEdit+ directly into the metamodel. Figure 4 shows the bindings that are related to requirements, logical bus and comments. Binding shows how different objects for example are connected, like that requirements can be derived from other requirements as shown below.

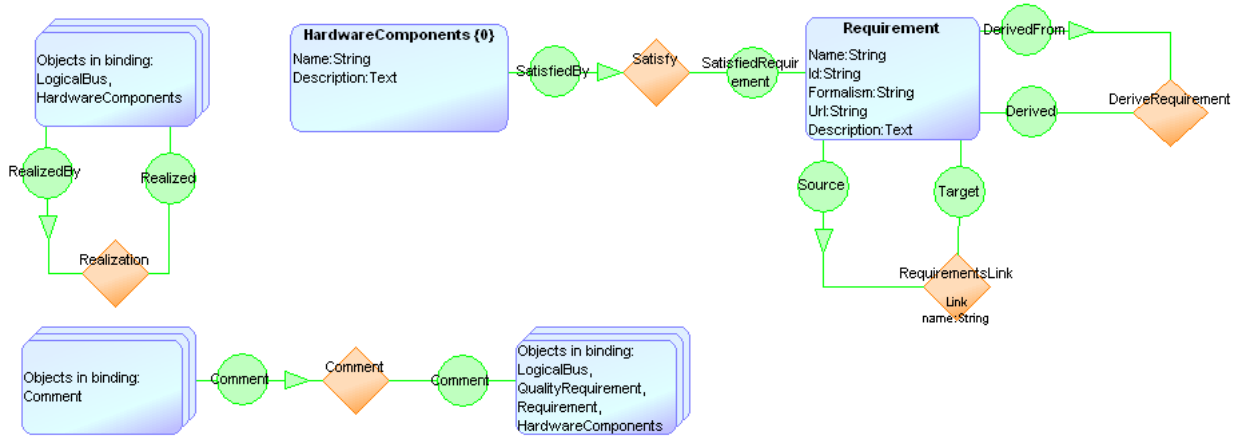


Figure 4: Bindings among other language concepts

EAST-ADL specification [1] includes also rules and constraints that specify which kind of models are complete and correct. In [1] most of these rules and constraints are written in English and are thus not processable by tools. Examples of such rule include uniqueness, such as Actuator must have a unique name matching certain syntax etc and direction of connections, such as that client-client connection is not possible among the functional prototypes.

In the metamodel of EAST-ADL implemented in MetaEdit+ also the rules and constraints are formally specified into the metamodel or to the related model checking reports. These formally defined rules and constraints enable static model checking for correctness, consistency and completeness.

The type-prototype pattern that EAST-ADL follows is implemented so that the whole architecture refers to a single type definition (to HardwareComponentType in Figure 5). Each prototype refers to its type definition via a decomposition link as shown in Figure 5.

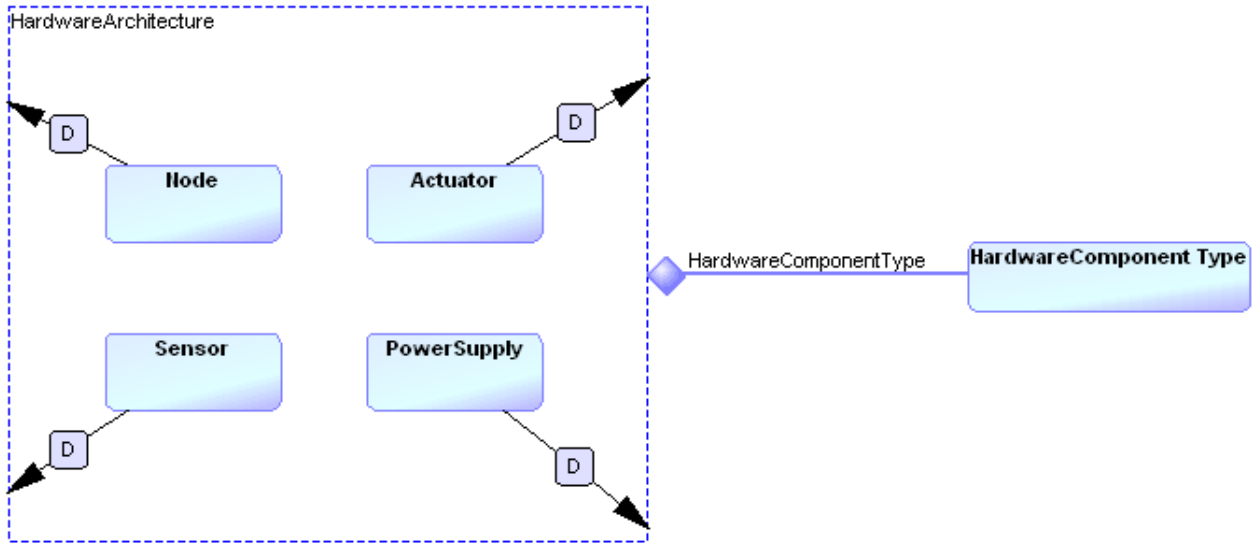


Figure 5: Model hierarchy and type-prototype structure

3.2 Checking reports

In addition to rules that are part of the metamodel definition itself, the EAST-ADL implementation in MetaEdit+ includes reports that check the correctness and completeness of the models. These checking reports 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 on empty identifying elements
- Warning on using same values for multiple objects
- Warning if prototypes are left undefined (this is also annotated in the diagram)
- Informing on incompatible type definitions among prototypes and types
- Informing on serially connected sensors / actuators
- Informing on wrong voltage definitions in power hardware connections (Voltage given at the same time when the isGround is set as True, different voltages in the same hardware connection)
- Unconnected Pin definitions
- Missing pin types in hardware connections
- Missing pin type definitions
- Informing on incompatible pin types in IOHardware based connections

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 6. This enables tracing from model elements with errors to the actual model element.

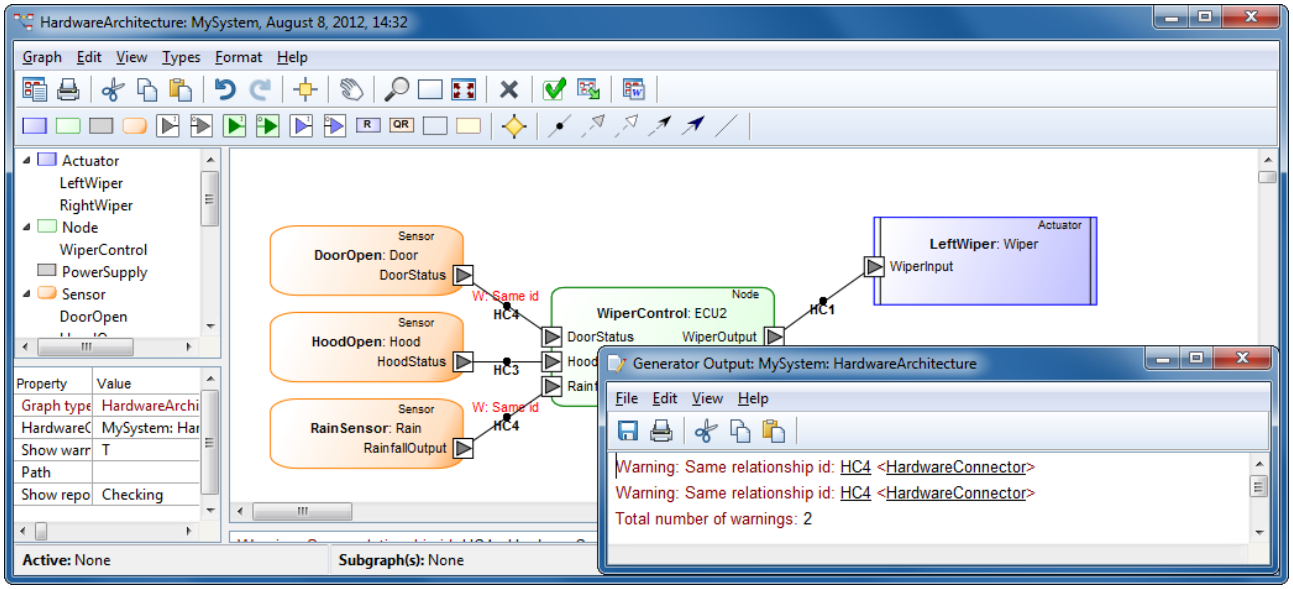
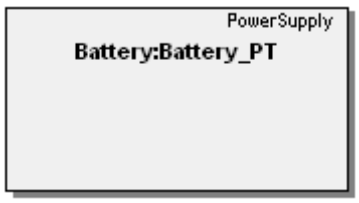
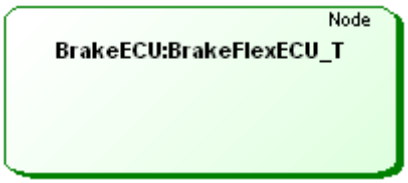
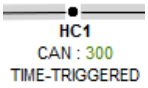
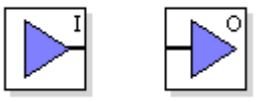
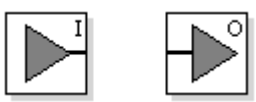


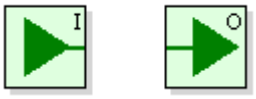
Figure 6. Error annotation and checking

3.3 Notation

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.	
PowerSupply 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.	
The LogicalBus represents a logical connection that carries data from any sender to all receivers. This information is now given in Hardware connector's dialog and its represented with wide background color behind the normal Hardware connector line	
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 power supply, 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.

Notation is used to highlight connections of hardware components as follows (see Figure 7 below):

- All available pin types, which were defined for the type, are presented on the perimeter line: inputs on the left hand side and outputs on the right hand side as illustrated in Figure 7 below.

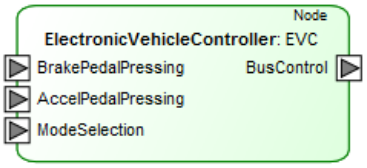


Figure 7. Visualizing pins for hardware component

Notation is also used to annotate models with errors and warnings related to for example incompleteness or missing data. The visualize warnings and possible errors the 'Show warnings' option can be selected in the graph properties. After setting this option, all places where some inconsistencies occur, red 'W' and the warning description is presented.

The warnings include:

- Missing connector names
- Use of same connector names for multiple connectors within a component
- Hardware connector which do have unconnected pins

3.4 Generators

Generators are used to produce various outputs, such as documentation, checking and metrics. These are generators are available for EAST-ADL too. In addition, support for EAXML export is implemented via generators: any hardware architecture description can be exported to EAXML by running the generator named '**ExportXML**'.

Figure 8 below shows the structure of the EAXML generator.

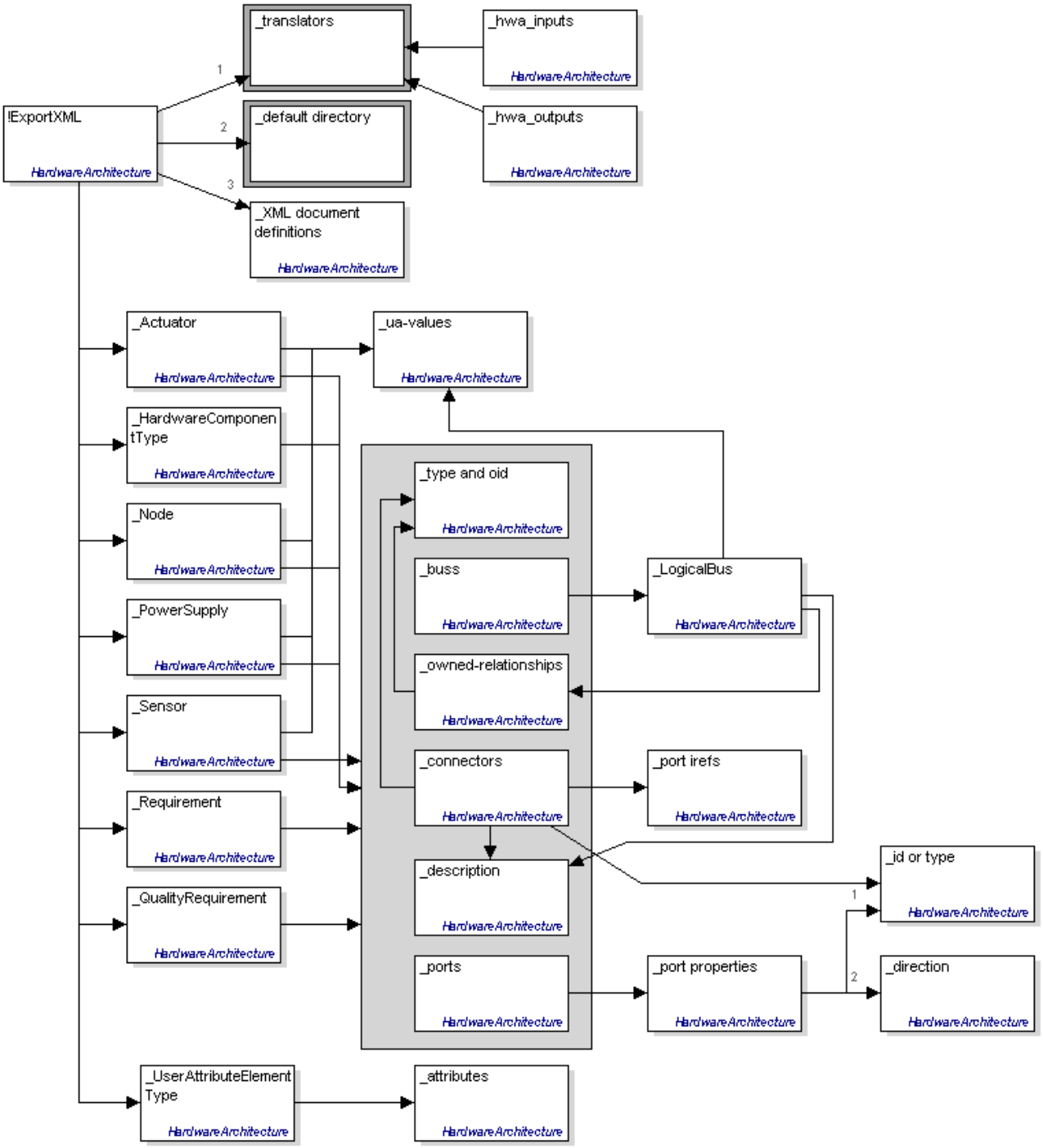


Figure 8. Structure of the Export to XML generator.

Similarly, model checking, as described in Section 3.3, are implemented as a generator. The checking generator, and its subgenerators, each checking separate aspects, is illustrated in Figure 9.

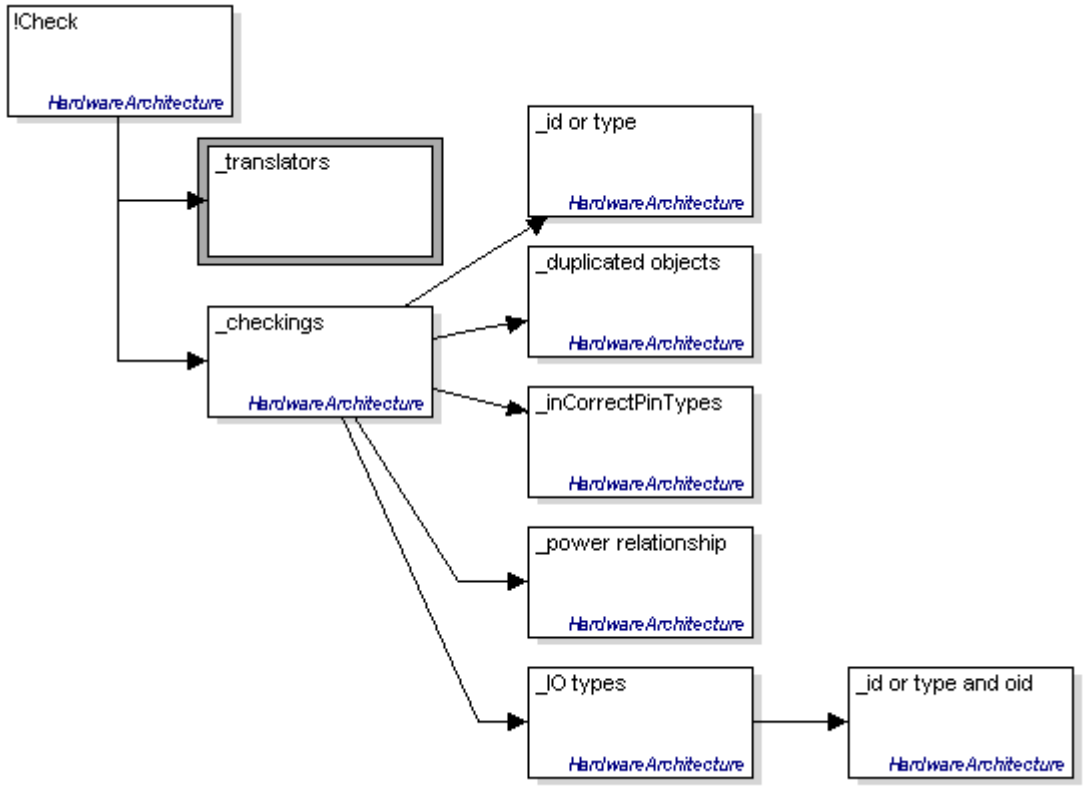


Figure 9. Structure of the checking generator.

The generators also include automated documentation production (exporting EAST-ADL models to word document) and generator to gather comments from language users.

- → The document generation applies autorun macro for Word to create the styles, headings etc. You may need to change your Word application security settings and allow running the macros when opening the generated document in Word. If macros are not accessed, the resulted file is stored and opened only as .rtf file. (You can edit your security settings by clicking the “Macro Security” in the Developer ribbon (Word 2007))
- → After you make any changes in the security, please run the generator again. If macros are accessed correctly, the resulted document should be stored in .doc format and have all the pictures, tables & hyperlinks as well as table-of-contents included
- → **Please notice: after you change the security settings in Word, then changed setting will remain to all other Word sessions too!**

In addition to EAXML support has been implemented to other tool-chain integration and artifact generation, such as Simulink integration and hardware architecture model importing by utilizing the XSLT transformation script.

Other companies have also implemented generators exporting AUTSAR models as well as data for simulation and analysis to support for example early phase design space exploration.

4 Conclusions

Hardware Architecture modeling language of EAST-ADL is documented in terms of the metamodel, notation and related generators for checking and exporting to EAXML format. This definition can be inspected and modified with MetaEdit+ Workbench.

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.

5 References

[1] EAST-ADL Domain-Model Specification, Version M2.1.10, 2011

[2] MetaCase, EAST-ADL tutorial, MetaCase Document No. EAT-4.5, August, 2012.

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

[4] MetaCase, Graphical Metamodeling Example, MetaCase Document No. GE-4.5, 2008, http://www.metacase.com/support/45/manuals/