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 SystemWeaver

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

Authors

Editor	E-mail
Nigsti Ayele	nigsti.ayele@systemite.se
Peter Lindqvist	peter.lindqvist@systemite.se
Jan Söderberg	jan.soderberg@systemite.se

Authors	E-mail
Nigsti Ayele	nigsti.ayele@systemite.se
Peter Lindqvist	Peter.lindqvist@systemite.se
Jan Söderberg	jan.soderberg@systemite.se

Reviewers	E-mail
Sara Tucci-Piergiovanni	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	2011-05-31	Initial version
0.2	2011-09-01	Updated metamodel mapping
0.3	2011-09-14	Updated metamodel elements mapping
0.4	2011-09-26	Updated to full metamodel implementation status
1.0	2011-09-28	Updated of installation information and User Instructions
2.0	2012-06-15	SystemWeaver tutorial for EAST-ADL, version 1.0
2.0	2012-08-15	Updated the meta-model to 2.1.10 implementation
2.0	2012-12-04	Dependability implementation presented for review on MS6.5
3.0	2013-03-28	M30 Release
4.0	2014-02-17	Final Release for review updated with final development status
4.0	2014-02-21	Final release, updated after review

Approval	Date
Henrik Lönn	2014-02-21

Table of contents

1 Introduction

This document describes the implementation of the complete EAST-ADL meta-model in SystemWeaver and the model transformation implemented.

1.1 Background

There are modeling-tools now cooperating in the MAENAD project which have been used by customers (OEM's, subcontractors etc.) for many years in the automotive market. The tools will be used to verify and validate the EAST-ADL meta-model. SystemWeaver is one among these tools.

1.2 Purpose

This document constitutes one part of the Deliverable D5.3.1 on Tool adaptations for EAST-ADL. The tools in the MAENAD project in the purpose of making the EAST-ADL language more useable to OEM's and subcontractors in business projects.

This part of D5.3.1 consists of a full EAST-ADL meta-model implementation, of SystemWeaver. It also describes the status of the bidirectional model transformation.

Understanding that the purpose to have the chosen tools in MAENAD, is to make the EAST-ADL more useable to OEM's and subcontractors in business projects. To realize interoperability among tools model transformation is implemented.

1.3 Abbreviations

SW – SystemWeaver

EAXML – EAST-ADL xml

EAST-ADL – Embedded Automotive SysTems ADL

ATL – Atlas transformation language

SWxml – SystemWeaver XML

MTT – Model Transformation Technology

MDE – Model Driven Engineering

RSQ – Research Question

2 SystemWeaver supporting East-ADL

2.1 Introduction to SystemWeaver

In short description SystemWeaver is:

- a generic system modeling platform
- a collaborative environment
- a commercial tool now used by customers on the automotive market

2.2 How to install

Using click once

- Download the application from http://apps.systemite.se/swMAENAD/setup.exe.
- Start the application by clicking the shortcut found in Windows Start menu

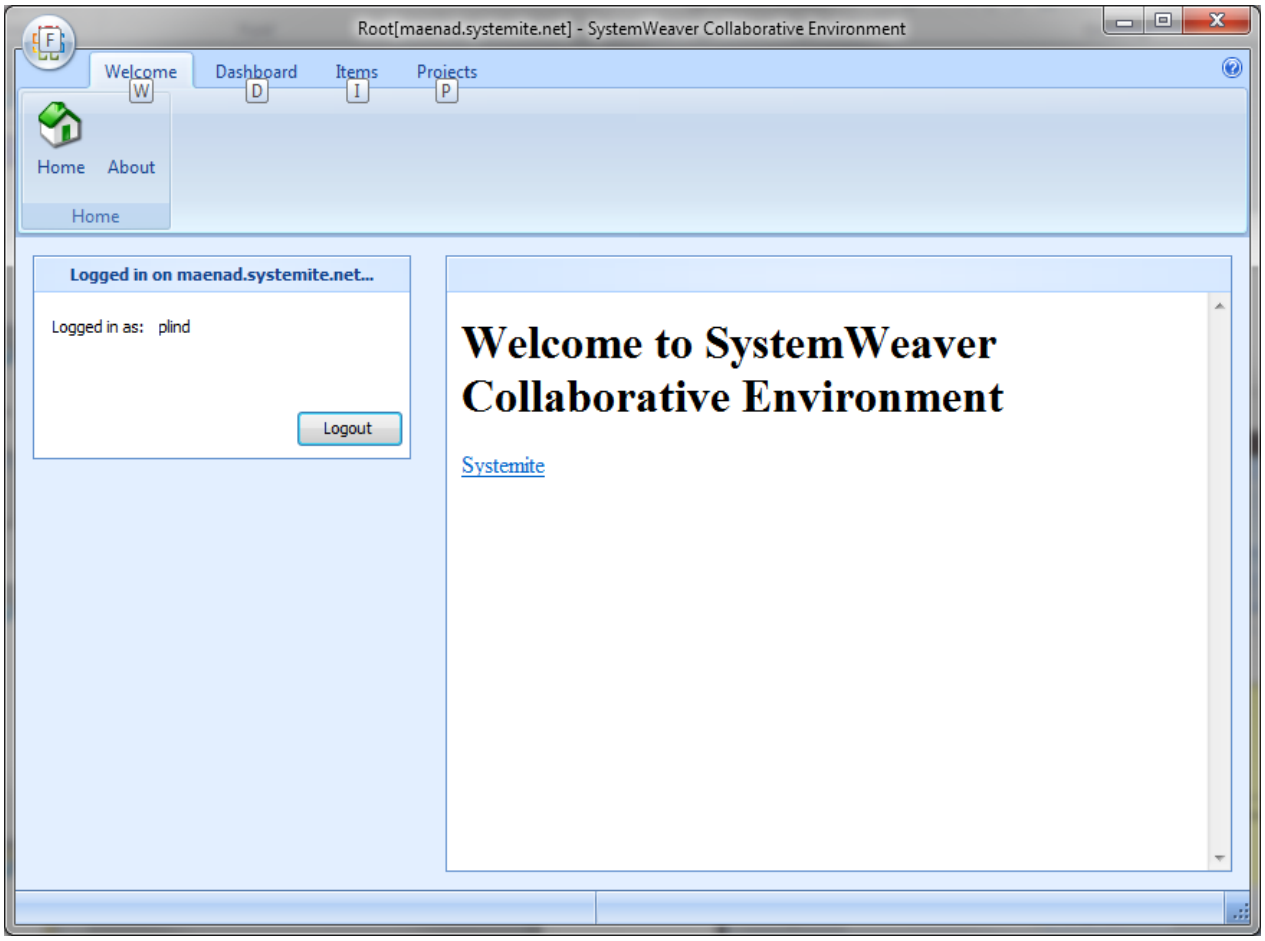

Figure 1: Welcome screen

How to login?

- Double click the exe file and you will see the screen in *Figure 2:*

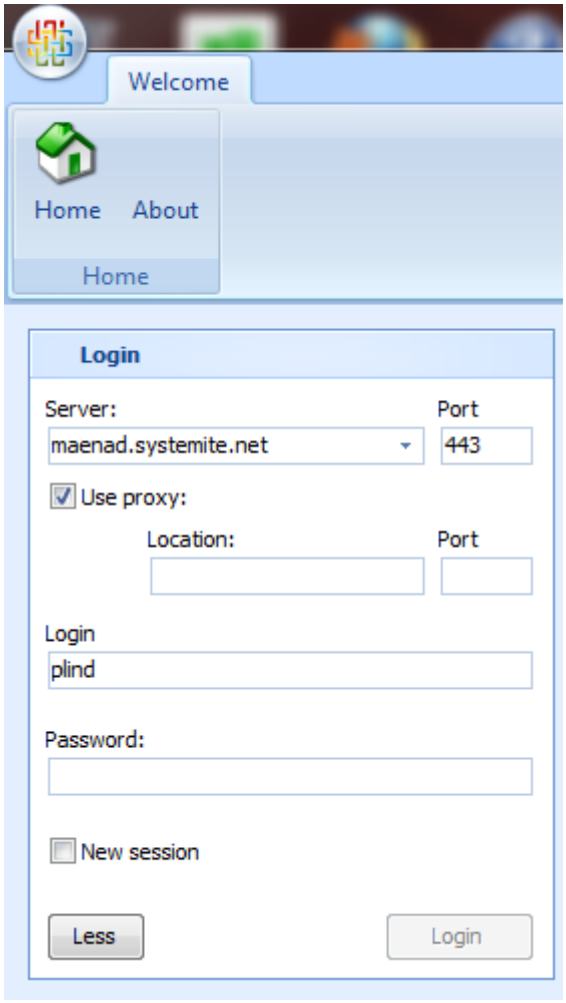

Figure 2: Login page

- Under **Login** and **Password**, enter your personal user credentials using (as distributed in the project), or the read-only account guest / guest.
- If you are sitting behind a proxy, then tick the Use proxy option and fill-in your proxy location and port[1]. You can find your proxy location either from your IT-department, or locate yourself from settings in your Internet browser. If you are using Internet Explorer you can find the proxy script under Internet Options / Connections / LAN settings / use automatic configuration script. The script will include the name of alternative proxy servers. Note that you have to browse to an external address (like google.com) in order to get the proxy activated prior to using the proxy in SystemWeaver.

[1] Note that as an alternative to using the proxy settings in SystemWeaver you could also use an external VPN application approved by your IT department.

2.3 EAST-ADL supported by SystemWeaver strong meta-model

The EAST-ADL meta-model is supported by systemweaver. Models can be extended by right clicking on an element and selecting the preferred part types from the options, guided by the meta-model.

For example see Figure 3, where “LH Window Lift Control” – is an EAST-ADL AnalysFunctionType. When right click you are given the option to “add” the elements/prototypes which is pre-defined by the East-Adl metamodel to be created by AnalysisFuntionType.

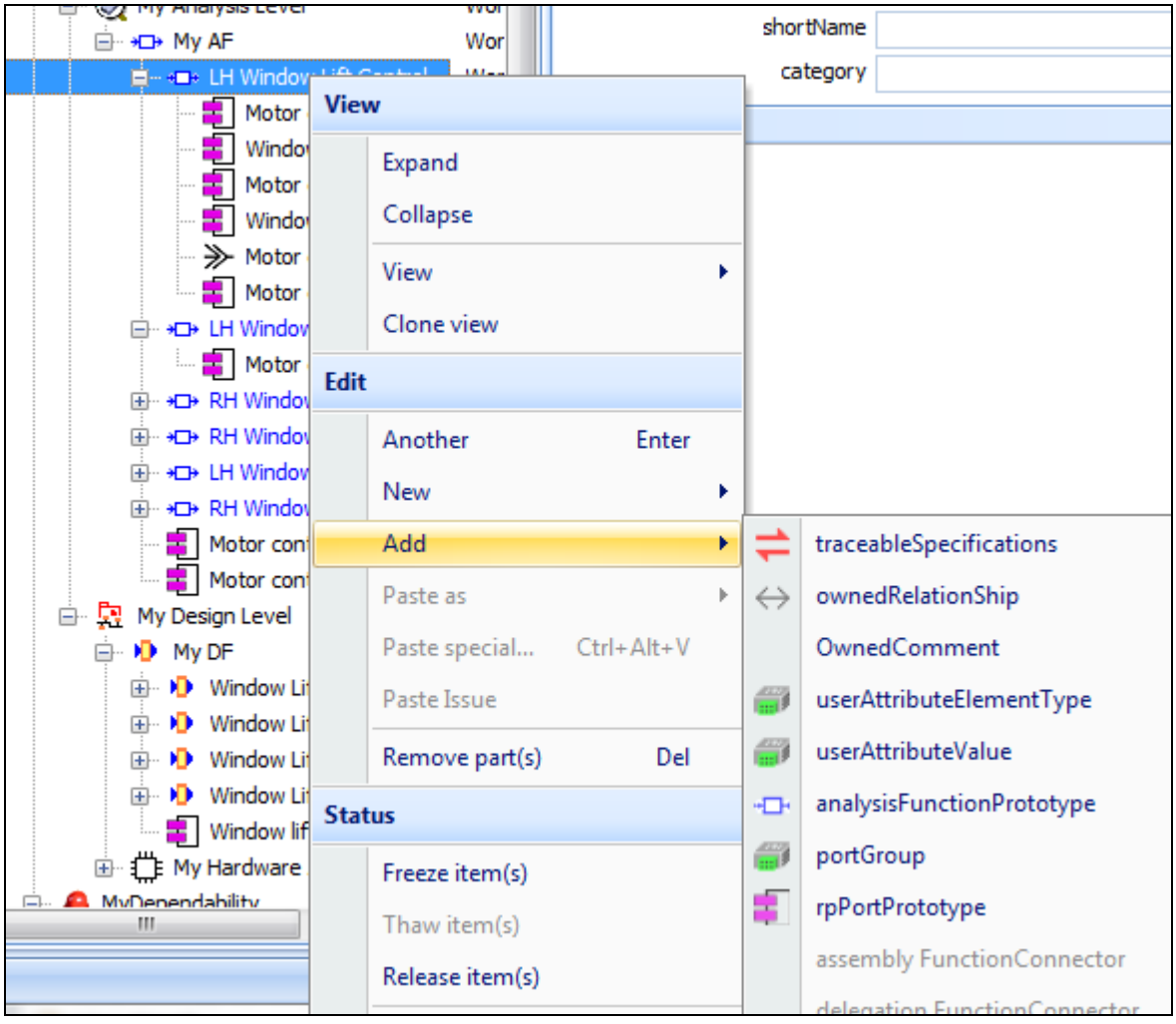

Figure 3 : Strong meta-model

2.4 Collaborative Environment for MAENAD

It is possible to create complete EAST-ADL models from your computer using the swExplorer application. It is also possible to reuse EAST-ADL model fragments created by other users and make changes without affecting the other user’s data. This means that even if a user creates another version of the item you use, it does not mean you need to update the item to work with it. It is possible to work with your old version without getting any conflicts.

3 Meta model mapping concepts for SystemWeaver

The Meta model concepts of EAST-ADL have been mapped to the set-up of SystemWeaver in a systematic matter. Ultimately this mapping could be performed automatically once the principles have been decided. However the first step is always to make an analysis of the alternative mapping principles in order to select the most appropriate, for each case.

The mapping of the SystemWeaver Meta set up to the EAST-ADL Meta-model should be based on a number of fundamental needs:

- Bi-directional Exchange between SystemWeaver and other tools based on the XML format can be done with minimal (preferably no) loss of information.
- Systemite have need for an efficient and effective implementation in SystemWeaver in terms of effective tool views, versioning, configuration and reuse, integrity of model elements, collaboration between users etc.
- Take into consideration also the limitations of the project in terms of man hours for the implementation

The investigation should detect and analyze any conflicts between the above needs, and come with conclusions on the best mapping strategy, and possibly on feedback to the EAST-ADL meta-model in case any fundamental deficiencies have been discovered.

4 Mapping Principles

4.1 Limitations and inherent properties of the SystemWeaver platform

SystemWeaver is a generic modeling platform, although it is not a general meta-modeling platform. This means that most of the needed constructs for general systems modeling are included in the platform. However, for each of the generic needs, the platform usually includes only one effective implementation concept, and that implementation concept may not support the full syntax or semantics of the formal meta-model of EAST-ADL. However, in most cases there is a suitable modeling construct in SystemWeaver, and the mapping is simple enough to support bidirectional exchange with EAXML without loss of information.

The available constructs and limitations of SystemWeaver are described below.

- SystemWeaver only supports single inheritance. In some cases multiple inheritances could be constructed by the use of inheritance structures. However such structures can only be true tree structures; “A” could inherit from both “B” and “C” only if “B” inherits from “C” (or vice versa). An example where simple inheritance is given is in the PortGroup construct where we choose to make it as a prototype (“part” in SystemWeaver). This choice has the benefit of that you know which port belongs to which port group.
- Inheritance is further only supported for classifiers (“items” in SystemWeaver), not prototypes. This means for example that the FunctionPort construct (under functionModeling) has to be implemented as inheritance for the interfaces rather than the ports, and also means that a number of generic ports are used instead (in, out, inout).
- Attribute definitions in SystemWeaver incorporates the definition of the attribute and the type of the attribute. This means that it is not possible to reuse the same type definition for different attributes. It also means that as the name of the attribute has to be chosen either the attribute name or the attribute type name.
- SystemWeaver has no inherent support for optional attributes; an attribute is either mandatory (“Default” in SystemWeaver) or exceptional (“Additional”).
- SystemWeaver has built-in support for instances, including deep instance structures such as those used in nested components models.
  Such instances (Prototypes) are managed as tree nodes and tree node references in SystemWeaver, rather than being explicitly created as objects by the user. However there is currently no support for type constraints on node references, as such constraints are usually implemented in custom type-dependent SystemWeaver tool views. (This also means that custom views have to be developed, since generic views would be very difficult to use, since the specific underlying node is not always evident from a Prototype, since there may be many possible alternative instances for a single Prototype, depending on the chosen context)
- SystemWeaver has no inherent “Package” concept, meaning that any model element can be managed without a package.
- SystemWeaver is based on formal identification rather than name binding. This means that all references in a SystemWeaver model must be correct at all times.
- SystemWeaver has no inherent support for constraints like OCL. Instead SystemWeaver uses generated reports, and structural concepts that replace the need for a class of constraints.

- The multiplicities in SystemWeaver is limited to “0..1” and “*”, since other multiplicity ranges have not proven useful in past development projects using SystemWeaver. Note that these multiplicities does not tell which multiplicities that are meaningful in a completed model, but rather which multiplicity that should be allowed at all times in the SystemWeaver database, which are two related, but slightly different things. Other cases should be covered by custom consistency checks, if needed. Note: SystemWeaver is a system modeling tool rather than a Meta-modeling tool, meaning that multiplicity ranges that do not translate into real development cases are not supported. For example how would a multiplicity range of “1” be used in a development tool? The “1” is clearly a rule that there should be a “link” with that cardinality, but the question is when, and what should the enforcement mechanisms be. Such constraints are usually implemented in the same way as other constraints; with generated reports.

4.2 Concept Mapping

The general mapping between EAST-ADL modeling constructs and the selected construct in SystemWeaver is described below.

EAST-ADL concept	SystemWeaver concept	Comment
Attribute	Attribute	In SystemWeaver the attribute name and the attribute type are the same. Therefore is the attribute type and name the same in the SystemWeaverExplorer (GUI for user models). One alternative solution is to prefix the attribute name to the attribute type: <name>:<type>
Attribute type	Attribute type	Since SystemWeaver does not distinguish between attribute names and attribute types, different attributes cannot share the same type definition (apart from sharing the same basic type) but separate types must be defined for such cases.
Optional attributes	-	Attributes with a multiplicity of 0..1 are not supported in Systemweaver, only mandatory attributes, with a multiplicity of 1
EAElement	Item	In SystemWeaver items have names as a property. Not an attribute as in EA.
«atpType»	Item, DefObj of a Part	The semantics of atpTypes are supported by SystemWeaver Item Types
«atpPrototype»	Part types, suffixed with a	The semantics of atpPrototypes are supported by

EAST-ADL concept	SystemWeaver concept	Comment
	"«atpPrototype»"	SystemWeaver Part types
«atpPrototype» + ports	Parts + ports.	A Part owns ports. Parts inherit its ports from its defining type (Item).
«atpStructureElement»	Items and parts in combination with its node tree. For Items see «atpType» and for parts «atpPrototype»	atpStructureElement which combines «atpType» + «atpPrototype» and have the property of "unique existential quantification" or "disambiguation" is in SystemWeaver solved with the node tree. Where Items and parts uses the node tree/instance tree. Short description for Nodes: In SystemWeaver there is an instance tree that is created automatically for each atpStuctureElement and all included prototypes. In the tree there is a unique instance node created and linked to each such element. Whenever there is a need for a reference to an element in the structure the reference is made to the corresponding node. Also keep in mind that the Item solution means that reusability is mandatory, even when not wanted. No significant drawbacks can be seen for this exception. Note that In a fine-grained versioning system like SystemWeaver even an atpStructureElement will be reused by different versions.
«instanceRef» ("Dependencies")	Nodes	Supported by SystemWeaver, with nodes and node references. SystemWeaver does not support type constraints on node references
«isOfType»	Meta model: DefType Model: DefObj	IsOfType is implemented as a standard DefType construct. "
Allocation	See comment	Maps to a node to node relation in SystemWeaver.

EAST-ADL concept	SystemWeaver concept	Comment
Multiple inheritance	Not supported (for classical reasons)	In EAST-ADL multiple inheritance is used in a few cases, like the AllocateableElement/InstanceRef case (see above), where special solutions are used anyway. The solution is to use only the most fundamental of the two (or many) inheritances. The inheritance that gives the most important properties.

4.3 Element Mapping and Implementation status

The **Impl Meta** columns describes if there is an explicit implementation in the meta model of the SystemWeaver set up. Note that sometimes there is support for a construct without any explicit implementation. In such case there is a note attached that describes details on this.

The **Impl view** means that there is some kind of dedicated or effective view support. However normally the standard views are effective. n/a for this column means that there is no identified need for a dedicated view.

The **Impl XML** view describes support for export and import.

n/a Not Applicable, the specific construct is not supported in the implementation platform

not No transformation support, there is a need but due to time the transformation could not be implemented

X Implemented

4.3.1 SystemModeling

EAST-ADL element	SystemWeaver concept	Impl Meta	Impl view	Impl XML	Comment
AnalysisLevel «atpStructureElement»	Item	X	X	X	Import and Export Supported
DesignLevel «atpStructureElement»	Item	X	X	X	Import and Export Supported
ImplementationLevel «atpStructureElement»	Item	*	*	*	*There is an optional support for AUTOSAR 4.0 available, including views and support for import and export of ARXML, however not installed in final setup.
SystemModel «atpStructureElement»	Item	X	n/a	X	Import and Export Supported

VehicleLevel «atpStructureElement»	Item	X	n/a	X	Import and Export Supported

4.3.2 FeatureModeling

EAST-ADL element	SystemWeaver concept	Impl Meta	Impl view	Impl XML	Comment
BindingTime	Item	X	X	X	Import and Export Supported
BindingTimeKind «enumeration»	Attribute (type)	X	X	X	Export to EAXML supported
Feature «atpStructureElement»	Item	X	X	X	Import and export supported
FeatureConstraint	Item	X	X	X	
FeatureGroup	Item	X	X	X	• Import and export supported • Multiplicity of child feature set to *
FeatureLink	Item	X	X	X	Import and export supported
FeatureModel «atpStructureElement»	Item	X	X	X	Import and Export supported
FeatureTreeNode {abstract}	Item	X	n/a	X	
VariabilityDependencyKind «enumeration»	attribute	X	n/a	X	See section 4.2 attributes for more details. Export to EAXML available

4.3.3 VehicleFeatureModeling

EAST-ADL element	SystemWeaver concept	Impl Meta	Impl view	Impl XML	Comment
DeviationAttributeSet	Item	X	n/a	n/a	No transformation support
DeviationPermissionKind «enumeration»	Attribute	X	n/a	X	Export to EAXML is available See section 4.2 attributes for more details.
VehicleFeature	Item	X	X	X	Import and export supported Attribute: isDesignVaribilityRationale is added.

4.3.4 FunctionModeling

EAST-ADL element	SystemWeaver concept	Impl Meta	Impl view	Impl XML	Comment
shortName	Attribute	X	n/a	X	• Export and import supported Used reg exp. [a-zA-Z][a-zA-Z0-9_]{0,31} instead of ([a-zA-Z][a-zA-Z0-9_]{0,31})+ for name validation. ()+ seems strange, and complexity of evaluation grows exponentially with the length of the name.
AllocateableElement {abstract}		n/a	n/a	not	In SystemWeaver only supports instance references directly from the item owning the instance/node tree, meaning that the allocation concept is not allowed in SystemWeaver. SystemWeaver also does not support multiple inheritances. AllocateableElements is therefore not implemented.
Allocation	Item	X	X	X	Export to EAXML supported
AnalysisFunctionPrototype	Part	X	X	X	• Import and Export Supported In SystemWeaver this is a part and a part does not support inheritance. Due to this reason the AnalysisFunctionPrototype is not inherited from the FunctionPrototype (Which is a part)
AnalysisFunctionType	Item	X	X	X	Import and Export supported
BasicSoftwareFunctionType	Item	X	X	X	Import and Export supported
ClientServerKind «enumeration»	Attribute	X	X	X	Import and Export is supported * See section 4.2 attributes for more details.
DesignFunctionPrototype	Part	X	X	X	Import and Export supported
DesignFunctionType	Item	X	X	X	Import and Export supported
EADirectionKind «enumeration»	Attribute	X	X	X	• Import and Export available Native tool view in SystemWeaver use parts (prototypes) corresponding to the in/out/inout values, rather than attribute types. However, a way to solve this is to use the inout part type only for all EAST-ADL ports types, and then use the DirectionKind to decorate the model with a direction.

EAST-ADL element	SystemWeaver concept	Impl Meta	Impl view	Impl XML	Comment
FunctionalDevice	Item	X	X	X	Import and Export supported
FunctionAllocation	Item	X	X	X	Export to EAXML supported
FunctionClientServerInterface «atpType»	Item	X	n/a	X	Import and Export supported
FunctionClientServerPort	Item	X	X	X	Import and Export supported
FunctionConnector «atpStructureElement»	Part Type (EAFN and EAFO)	X	X	X	• Import and Export supported • Part types used rather than item types for best use of the SystemWeaver Platform. Separate part types are used for the assembly and the delegation type, due to different instance tree reference mechanisms (node and part respectively)
FunctionFlowPort		X	X	X	Import and Export supported
FunctionPort {abstract} «atpPrototype»	Item {abstract}	X	X	X	Import and Export supported
FunctionPowerPort		X	X	X	Import and Export supported
FunctionPrototype {abstract} «atpPrototype»		*	X	X	*Not Supported or needed explicitly. See DesignFunctionPrototype and AnalysisFunctionProtoype for details.
FunctionType {abstract} «atpType»	Item {abstract}	X	X	X	Import and Export supported
HardwareFunctionType	Item	X	X	X	Import and Export supported
LocalDeviceManager	Item	X	X	X	Import and Export supported
Operation	Item	X	not	not	No transformation support
PortGroup	Part	X	not	not	*Since this is implemented using the Part construct in SystemWeaver, subPortGroups cannot be used. The advantage of this is to avoid ambiguity of ports (functionPorts). This means that it can be identified which port belongs to which portGroup and functionType. See ticket 50. Maybe portGroup also is in the wrong abstraction level. Should be closer to implementation level. Because the possibility of knowing group of ports is high on the less abstraction level than in the hardwareModeling.

EAST-ADL element	SystemWeaver concept	Impl Meta	Impl view	Impl XML	Comment
Actuator	Item	X	X	X	Import and Export supported
AllocationTarget	Part	X	*	*	*No meaning to implement, see multiple inheritance for SystemWeavereaver. In SystemWeaver HardwareComponentPrototype is a part and parts cannot be inherited, due to this reason it is not feasible to implement the AllocationTarget.
Cardinality	Attribute	X	X	X	Export to EAXML is supported

4.3.5 HardwareModeling

EAST-ADL element	System Weaver concept	Impl Meta	Impl view	Impl XML	Comment
Actuator	Item	X	X	X	Import and Export supported
AllocationTarget {abstract}	Item	X	X	X	• Export to EAXML is available • This is supported in systemweaver way due to the allocation support.
CommunicationHardwarePin	Item	X	X	X	Import and Export supported
HardwareComponentPrototype «atpPrototype»	Part	X	X	X	Import and Export supported
HardwareComponentType «atpType»	Item	X	X	X	Import and Export supported
HardwareConnector «atpStructureElement»	Item	X	X	X	Import and Export supported
HardwarePin «atpStructureElement»	Item	X	X	X	Import and Export supported
HardwarePinDirectionKind «enumeration»	Attribute	X	X	X	Import and Export supported
HardwarePinGroup	Item	X	X	not	No transformation support
IOHardwarePin	Item	X	X	X	Import and Export supported
IOHardwarePinKind «enumeration»	Attribute	X	X	X	See section 4.2 attributes for more details.
Node	Item	X	X	X	Import and Export supported
PowerHardwarePin	Item	X	X	X	Import and Export supported
ElectricalComponent	Item	X	X	X	Import and Export supported
Sensor	Item	X	X	X	Import and Export supported

4.3.6 Environment

EAST-ADL element	System Weaver concept	Impl Meta	Impl view	Impl XML	Comment
ClampConnector «atpStructureElement»	Part	X	not	not	No transformation support
Environment	Item	X	not	not	No transformation support

4.3.7 Behavior

EAST-ADL element	System Weaver concept	Impl Meta	Impl view	Impl XML	Comment
Behavior	Item	X	not	not	No transformation support
FunctionBehavior	Item	X	not	not	No transformation support
FunctionBehaviorKind «enumeration»	Attribute	X	not	not	No transformation support See section 4.2 attributes for more details.
FunctionTrigger	Item	X	not	not	No transformation support
Mode	Item	X	not	not	No transformation support
ModeGroup	Item	X	not	not	No transformation support
TriggerPolicyKind	Attribute	X	n/a	not	See section 4.2 attributes for more details.

4.3.8 Variability

EAST-ADL element	System Weaver concept	Impl Meta	Impl view	Impl XML	Comment
ConfigurableContainer	Item	X	n/a	not	No transformation support
ConfigurationDecision	Item	X	n/a	not	No transformation support
ConfigurationDecisionFolder	Item	X	n/a	not	No transformation support
ConfigurationDecisionModel {abstract}	Item {abstract}	X	n/a	not	No transformation support
ConfigurationDecisionModelEntry {abstract}	Item {abstract}	X	n/a	not	No transformation support
ContainerConfiguration	Item	X	n/a	not	No transformation support
FeatureConfiguration	Item	X	n/a	not	No transformation support
InternalBinding	Item	X	n/a	not	No transformation support
PrivateContent	Item	X	n/a	not	No transformation support
ReuseMetaInformation	Item	X	n/a	not	No transformation support
SelectionCriterion	Item	X	n/a	not	No transformation support

Variability	Item	X	n/a	not	No transformation support
VariableElement	Part	X	n/a	not	No transformation support
VariationGroup	Item	X	n/a	not	No transformation support
VehicleLevelConfigurationDecisionModel	Item	X	n/a	not	No transformation support

4.3.9 Requirements

EAST-ADL element	System Weaver concept	Impl Meta	Impl view	Impl XML	Comment
DeriveRequirement	Item	X	n/a	X	Export to EAXML is supported
OperationalSituation	Item	X	n/a	X	Export to EAXML is supported
QualityRequirement	Item	X	n/a	X	Export to EAXML is supported
QualityRequirementKind	Attribute	X	n/a	X	Export to EAXML is supported See section 4.2 attributes for more details.
Refine	Item	X	n/a	X	Export to EAXML is supported
Requirement	Item	X	n/a	X	Export to EAXML is supported
RequirementsContainer	Item	X	n/a	X	Export to EAXML is supported
RequirementsLink	Item	X	n/a	X	Export to EAXML is supported
RequirementsModel	Item	X	n/a	X	Export to EAXML is supported
RequirementSpecificationObject {abstract}	Item {abstract}	X	n/a	not	No transformation support
RequirementsRelatedInformation	Item	X	n/a	not	No transformation support
RequirementsRelationGroup	Item	X	n/a	not	No transformation support
RequirementsRelationship {abstract}	Item {abstract}	X	n/a	not	No transformation support
Satisfy	Item	X	n/a	X	Export to EAXML is supported

4.3.10 Dependability

EAST-ADL element	SystemWeaver concept	mpl Meta	Impl view	Impl XML	Comment
ControllabilityClassKind	Attribute	X	n/a	X	Export to EAXML is supported
Dependability	Item	X	n/a	X	Export to EAXML is supported
DevelopmentCategoryKind	Attribute	X	n/a	X	Export to EAXML is supported
ExposureClassKind	Attribute	X	n/a	X	Export to EAXML is supported
FeatureFlaw	Item	X	X	X	Export to EAXML is supported

Hazard	Item	X	n/a	X	Export to EAXML is supported
HazardousEvent	Item	X	n/a	X	Export to EAXML is supported
Item	Item	X	n/a	X	Export to EAXML is supported
SeverityClassKind	Attribute	X	n/a	X	Export to EAXML is supported

4.3.11 ErrorModel

EAST-ADL element	SystemWeaver concept	Impl Meta	Impl view	Impl XML	Comment
Anomaly	Item	X	X	X	Export to EAXML is supported
ErrorBehavior	Item	X	n/a	X	Export to EAXML is supported
ErrorBehaviorKind	Attribute	X	n/a	X	
ErrorModelPrototype	Part	X	X	X	Export to EAXML is supported
ErrorModelType	Item	X	X	X	Export to EAXML is supported
FailureOutPort	part	X	X	X	Export to EAXML is supported
FaultFailurePort	part	X	X	X	Export to EAXML is supported
FaultFailurePropagationLink	Node	X	X		
FaultInPort	part	X	X	X	Export to EAXML is supported
InternalFaultPrototype	part	X	X	X	Export to EAXML is supported
ProcessFaultPrototype	part	X	X	X	Export to EAXML is supported

4.3.12 UseCases

EAST-ADL element	SystemWeaver concept	Impl Meta	Impl view	Impl XML	Comment
Actor	Item	X	n/a	X	Export to EAXML is supported
Extend	Item	X	n/a	X	Export to EAXML is supported
QuantitativeSafetyConstraint	Part	X	n/a	X	Export to EAXML is supported
shortName	Attribute	X	n/a	X	Export to EAXML is supported

5 View Examples

5.1 Overview

The view examples are prepared as screen shots, using the swExplorer client version 1.45.0.8918 and EAST-ADL specific Extension views SWExtension.GraphsForMAENAD.dll version 1.0.5000.17884 and SWExtension.GraphsForFeatureModels.dll version 1.0.5044.28907

The views are further from examples prepared in the server set up for the MEANAD project at address maenad.systemite.net, port number 443, either imported from examples prepared by other tools of the MAENAD project, or created manually.

Additional view examples can be found in reference [3].

5.2 Generic views

The figure below illustrates the use of standard generic views for browsing and editing of EAST-ADL models, using a tree view for browsing and building model structures. Depending on the type of the selected model element optional detailed views become available, either as seen in the “ribbon” part of the tool or the drop-down menu, where the Overview view is selected below.

(There are around 100 generic views in SystemWeaver, many of which support aspects like versioning, access control etc.)

Generic views in SystemWeaver are those views that need not be configured specifically for a specific meta model.

Note: As can be seen in the figure the ‘name’ attribute is left empty for the selected FeatureModel (of the BBW_4Wheel example), since the standard SystemWeaver Item name concept is used for the same purpose. (The ‘name’ attribute was retained in the meta model for sake of completeness).

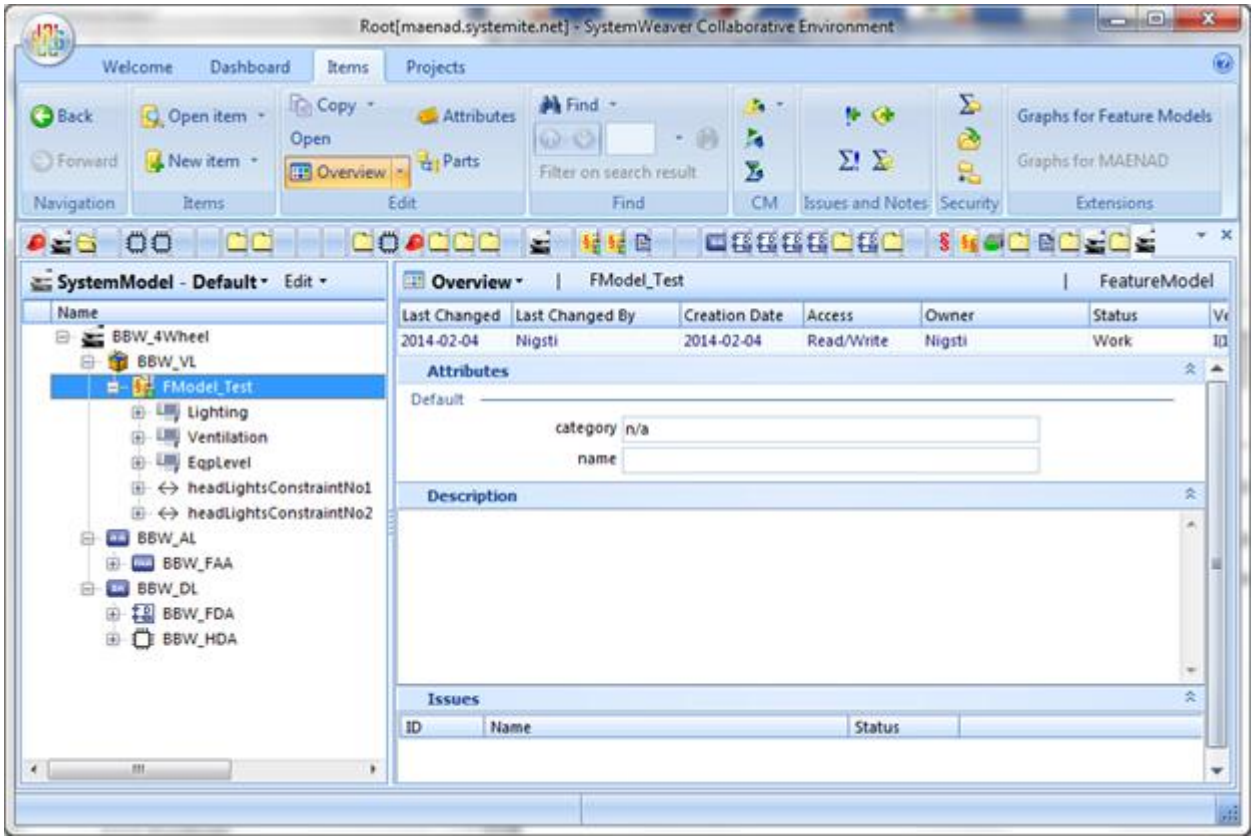

Figure 4: Generic view

5.3 Configurable standard views

The example below shows the AnalysisLevel AnalysisFunctionType FAA_T of the (imported) BBW_SystemModel example.

The graphical view used for the AnalysisFunctionType is a standard, configurable, view in SystemWeaver, in this case configured specifically to display the EAST-ADL AnalysisFunctionType elements.

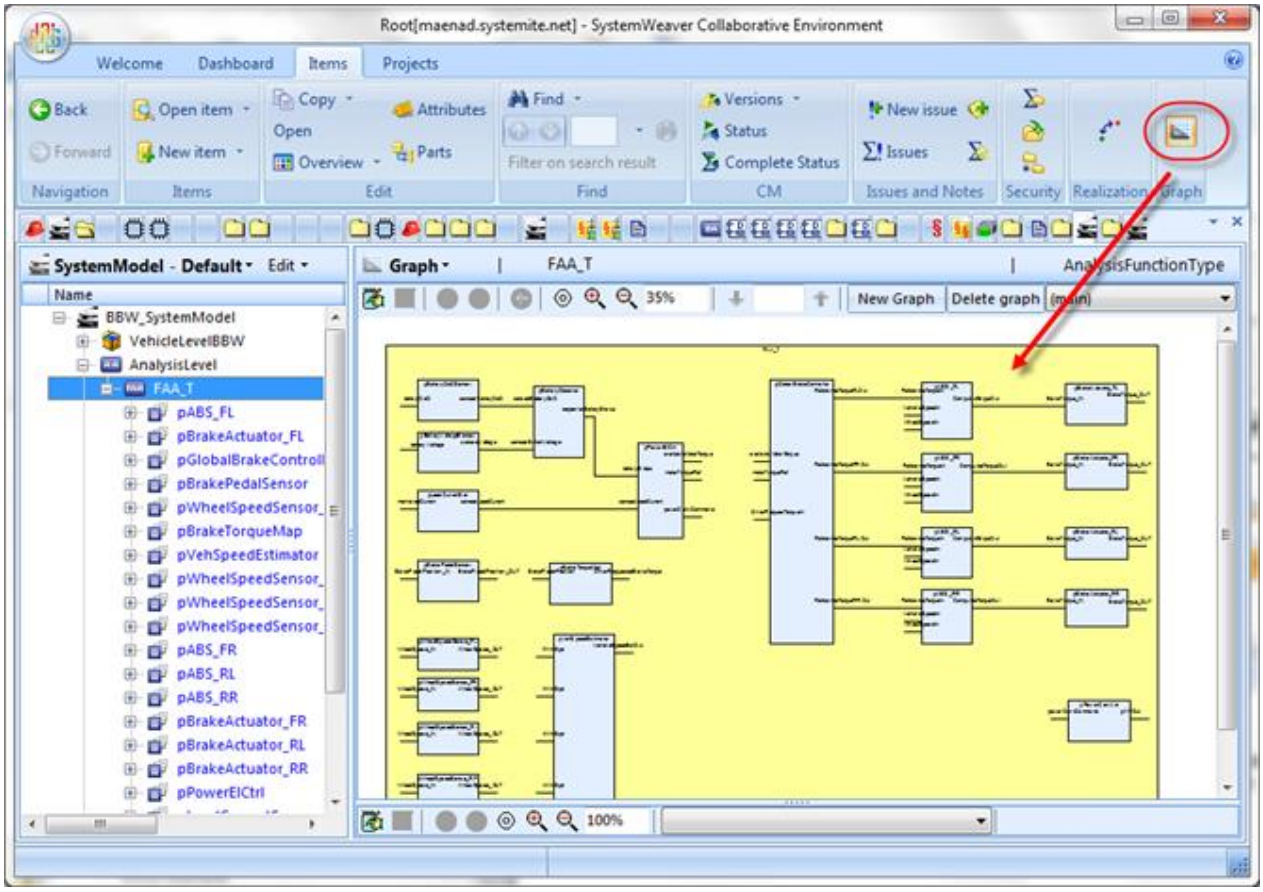

Figure 5: Configurable graphical view

The figure below shows a special configurable view, in that the view was implemented to support the allocation model used in EAST-ADL, although the view was made configurable so that it could be used for most mapping and allocation purposes in EAST-ADL using the instanceRef construct.

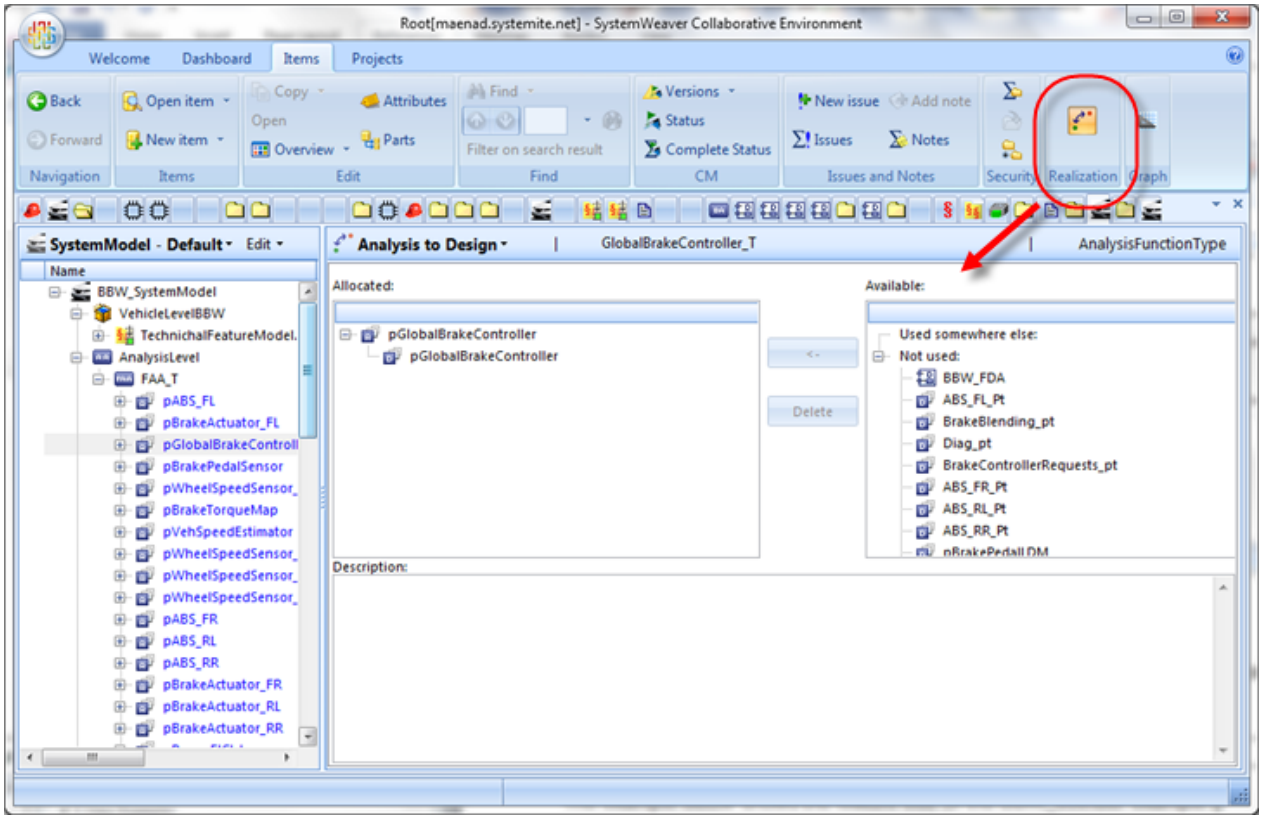

Figure 6: Configurable allocation/traceability view

The view supports traceability in three different ways, by displaying the established traceability from as seen above from the selected AnalysisFunctionPrototype where the traceability is created, secondly by displaying the same traceability from the DesignFunctionPrototype, and thirdly by displaying the overall traceability from a context perspective (see figures below).

Note that each of these views can scale to large size models.

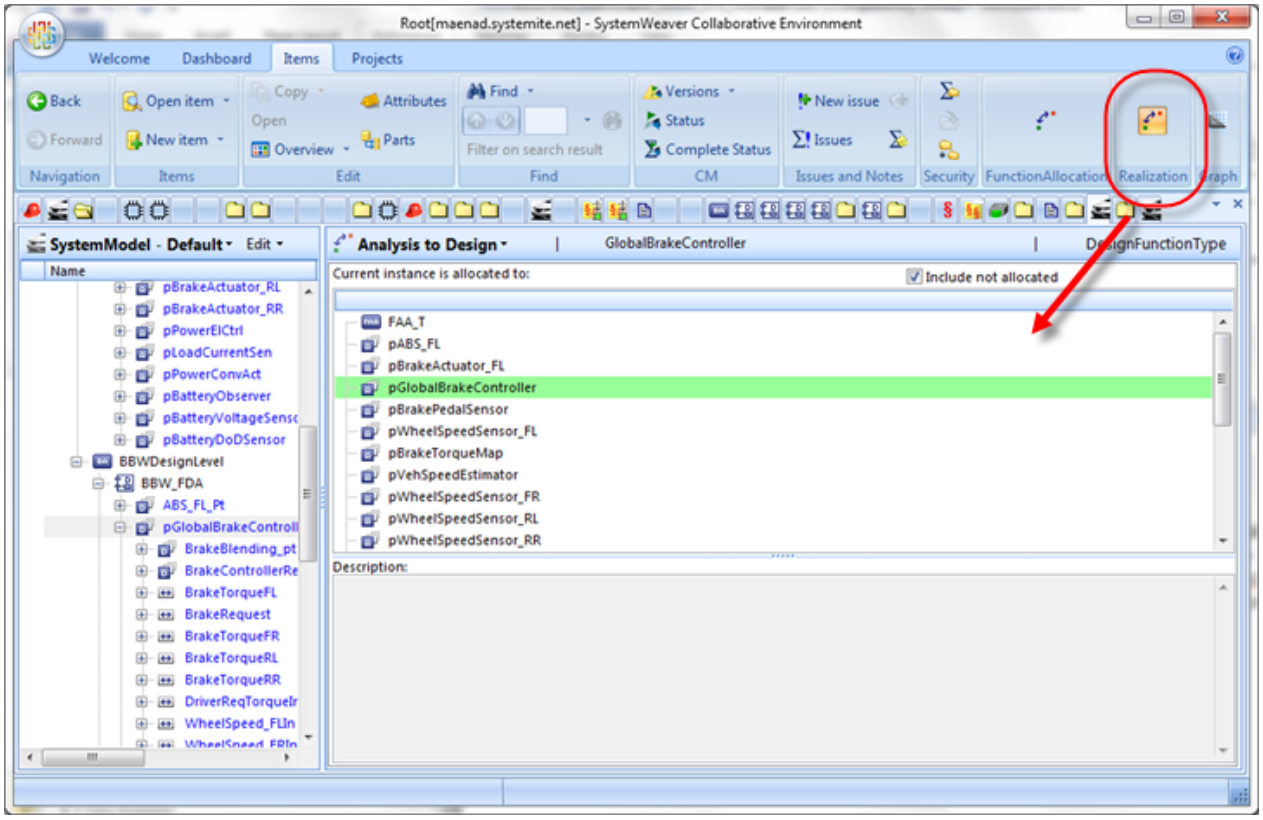

Figure 7: Traceability from Design level

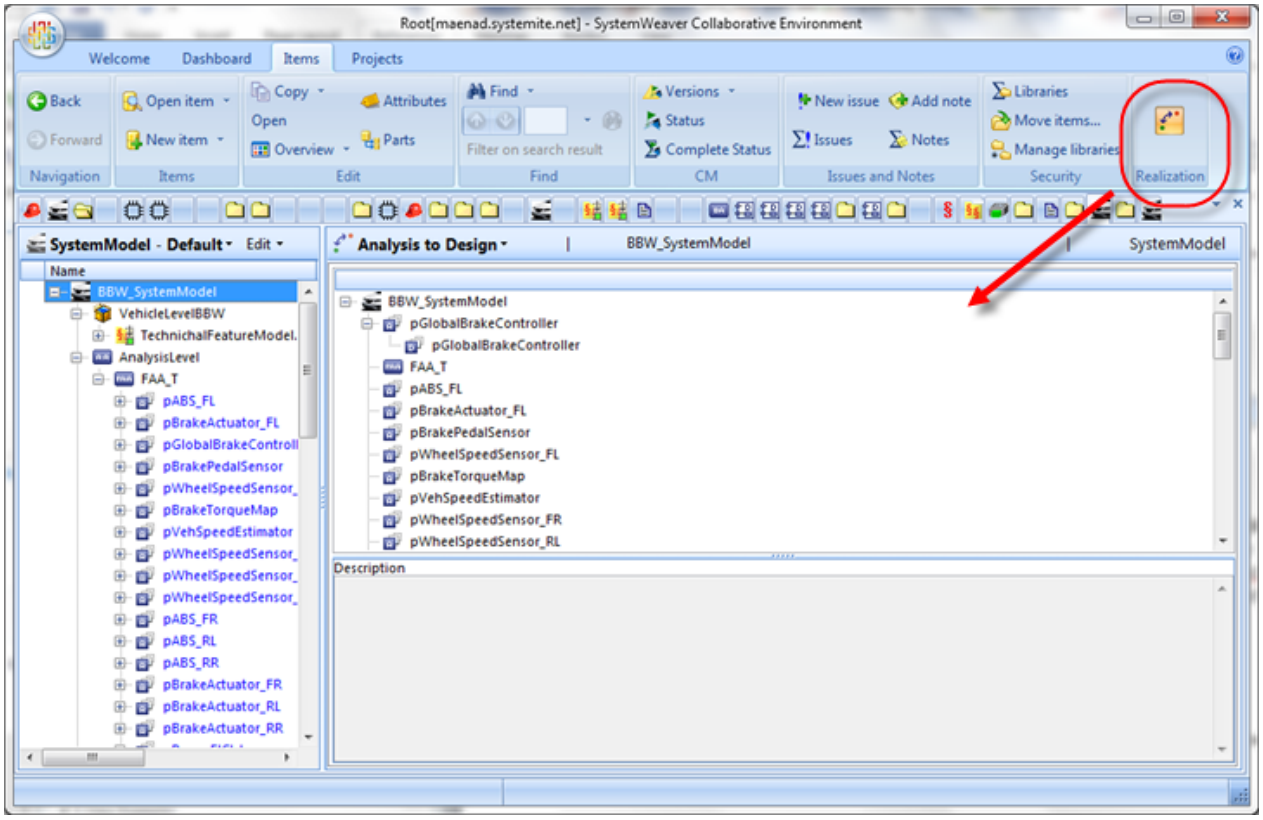

Figure 8: Traceability from SystemModel (context) perspective

5.4 Custom views: Graphical Feature View

The example below shows the feature tree of the BBW_4Wheel example.

The feature view can be activated by selecting the **Graphs for Feature Models** view whenever a FeatureModel is selected in the tree.

This view has been implemented as a custom SystemWeaver Extension view, distributed as a separate DLL file, but integrated into the GUI of the swExplorer application.

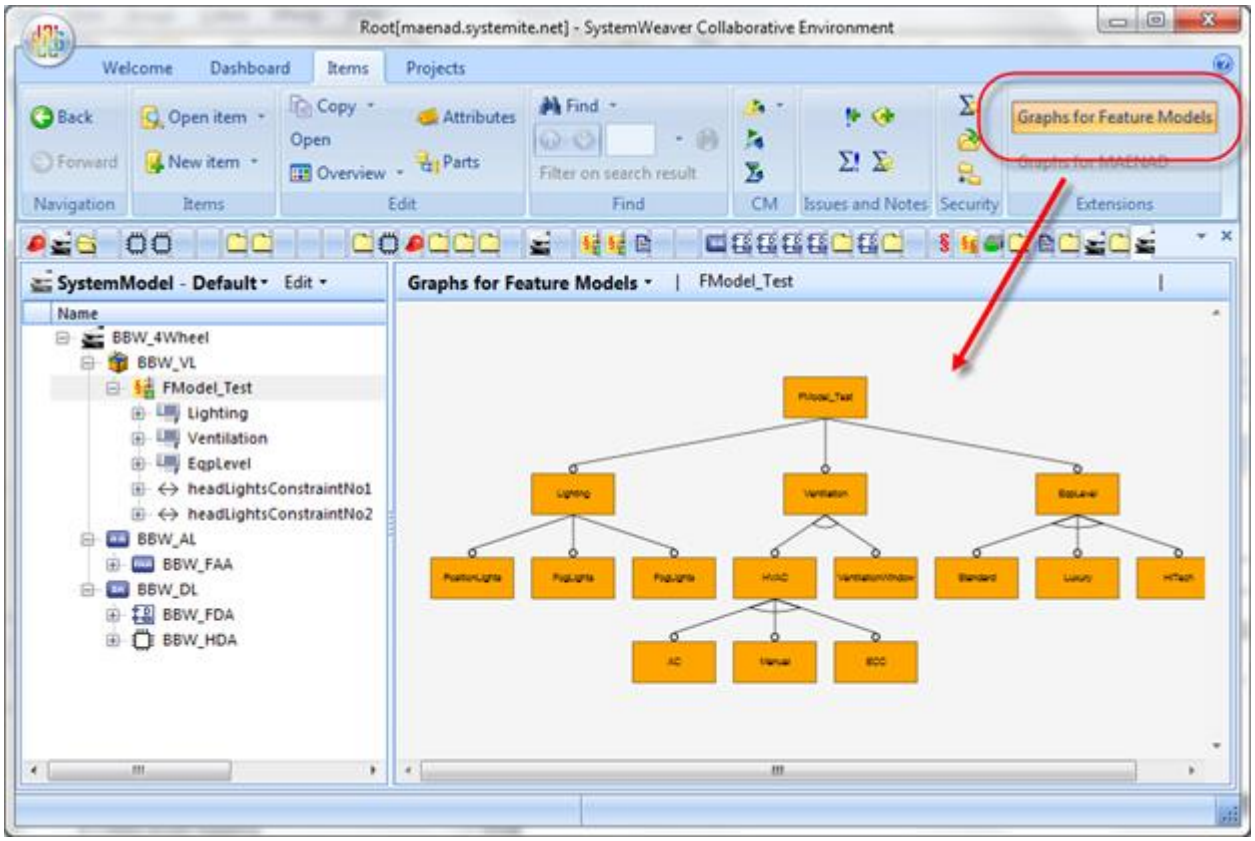

Figure 9: Graphical feature view

This view uses an automatic layout and rendering algorithm, meaning that no manual drawing is required, e.g. after an import of the model into the SystemWeaver database.

A related custom view type is the support for automatic calculation of attribute values, as seen in the figure below, where an ASIL value is calculated out of other attributes.

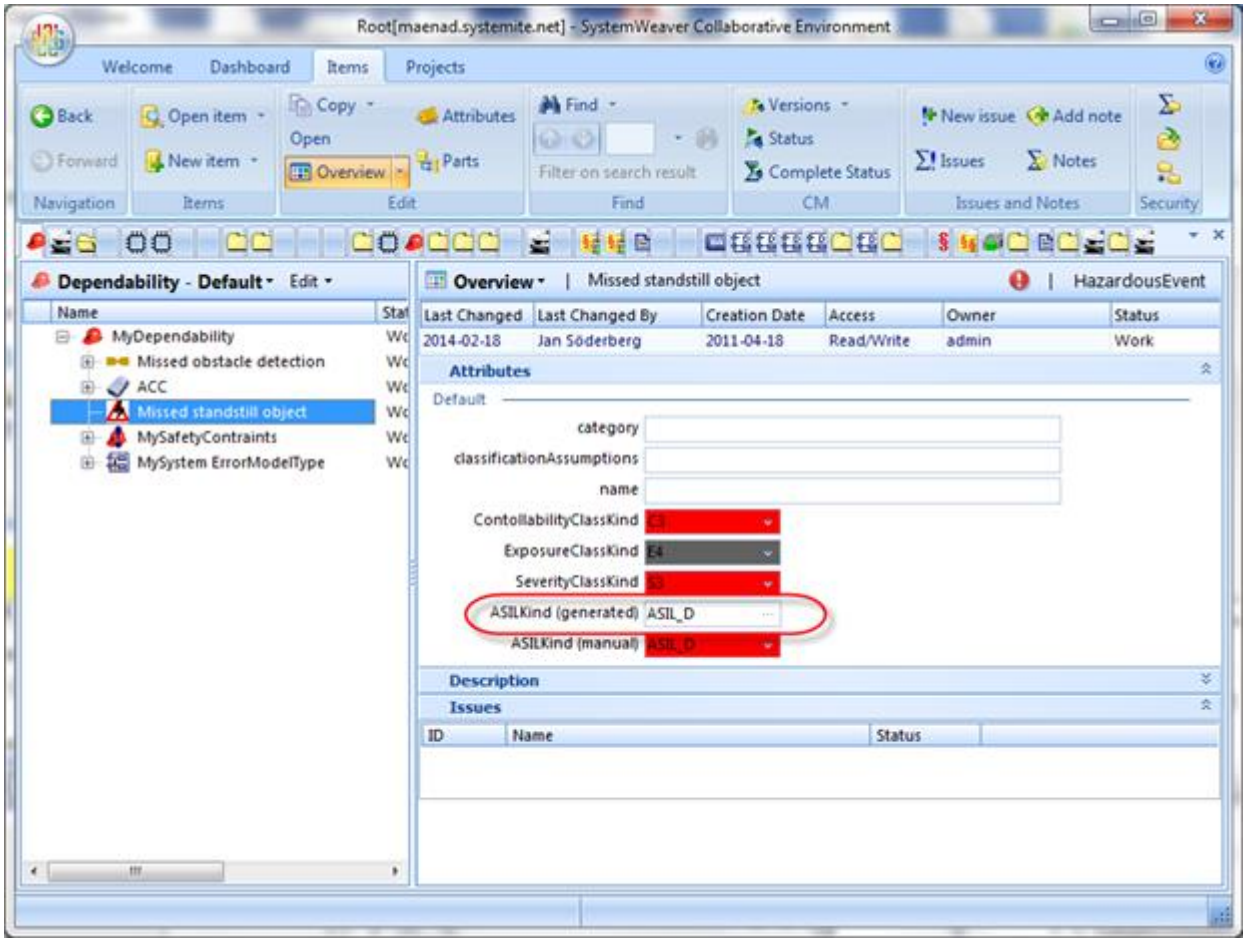

Figure 10: Calculated ASIL value

6 Model Transformation

ATL is used to implement the transformation between SystemWeaver and EAXML models.

ATL is a popular M2M transformation technology. According to [4], ATL model transformation adopts partially declarative transformation approach which merges both declarative and imperative transformations. The hybrid feature enables us to deal with conditions of domain specific scenarios, simpler formal syntax and mathematical foundations. The fact that this tool has declarative behavior helps to have error free code due to the no side effect feature [5].

ATL complements declarative behavior over imperative and vice versa in case of impossibilities. On situations such as impossibility of transformation using declarative language, ATL allows users to use imperative language in order to enable complex transformations. ATL is one among the different model transformation tools which has the capability to automatically create target model using the information represented in source model, source meta-model and target meta-model.

The ATL model, which is basically the main transformer, uses source meta-model, target meta-model and source model as inputs during transformation. The ATL model consists of import, rules and helpers. The helpers mostly describe global variables derived from the input meta-model. These helpers are used later in the rules to implement the real transformation. ATL model supports unidirectional model transformation. However, it is possible to implement bidirectional transformation through explicit implementation of both side transformations.

Detailed analysis of both the source and the target meta-models is very important prior to the transformation. These derived rules to transform SystemWeaver to EAXML and vice versa are derived from the Analysis result and are discussed in detail in the forthcoming topics.

Methodology used to implement transformation:

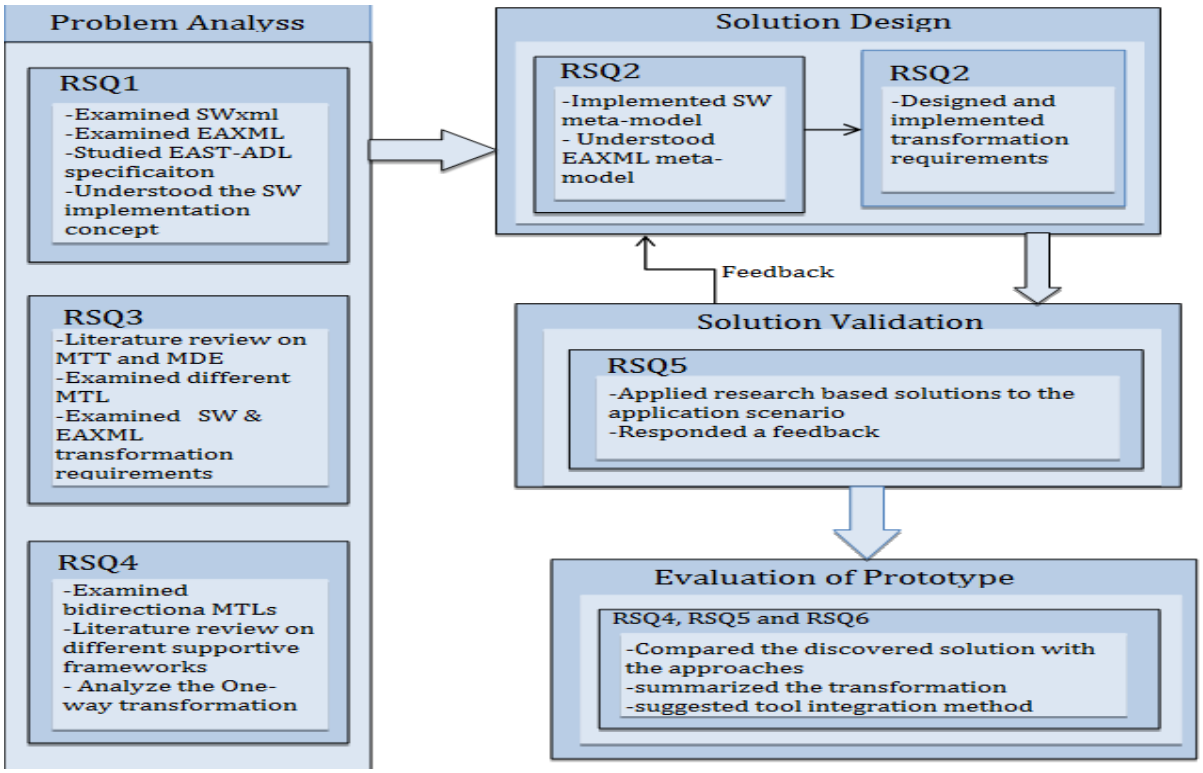

Figure 11: Methodology

Generally more than one transformation is used to handle the EAXML and SWxml models. ATL can read and write .ecore or .xml models.

To hide the configuration complexity of ATL, a graphical user interface has been implemented. The transformation supports both for EAST-ADL M2.1.9 and M2.1.10.

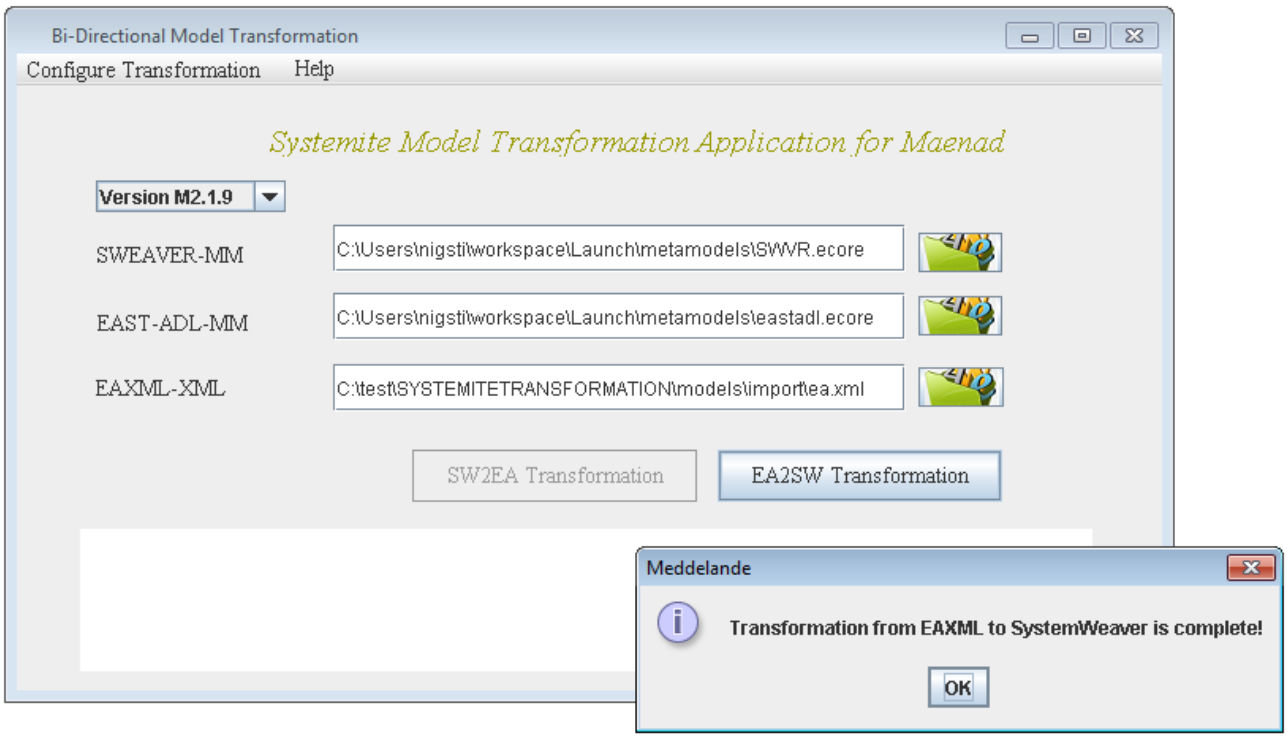

Figure 12 : Systemite bidirectional model transformation application

6.1 Transformation Architecture

The ATL transformation is a model based model to model transformation. The bidirectional transformation between SystemWeaver model and EAXML is realized through two ways of single transformation. The transformation model always checks at the input and output meta-models to check the semantic correctness of the input model and to create the output meta-model based model. The figure below shows how the ATL model checks on the meta-models during transformation.

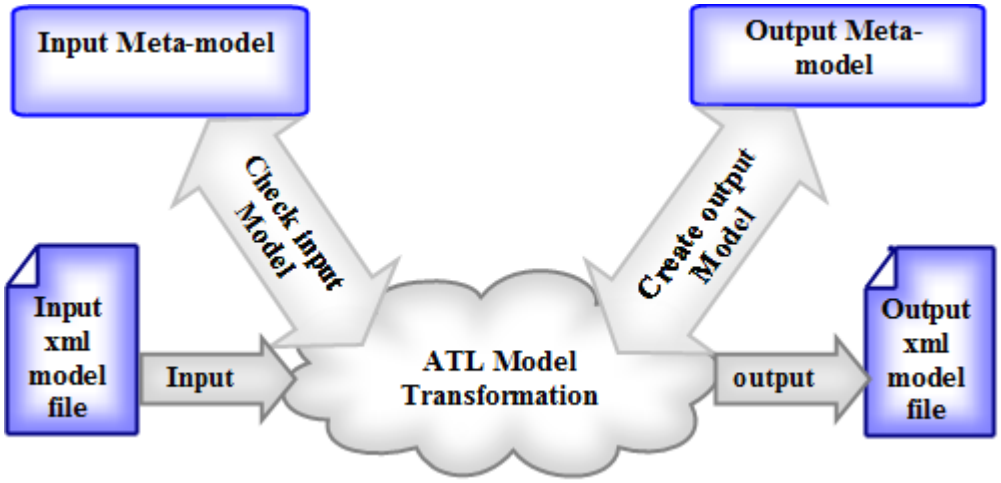

Figure 13: Over all transformation architecture

6.2 Meta-model mapping

According to the investigation made between SystemWeaver and EAXML meta-models, it is discovered that the EAXML meta-model is implicit. This means there are no inheritances applied. However, the SystemWeaver meta-model is an implicit meta-model where represents elements as Item and relationships as part or node. The figure below shows how the meta-model implementations are different from one another. Both meta-models are represented using ECORE format.

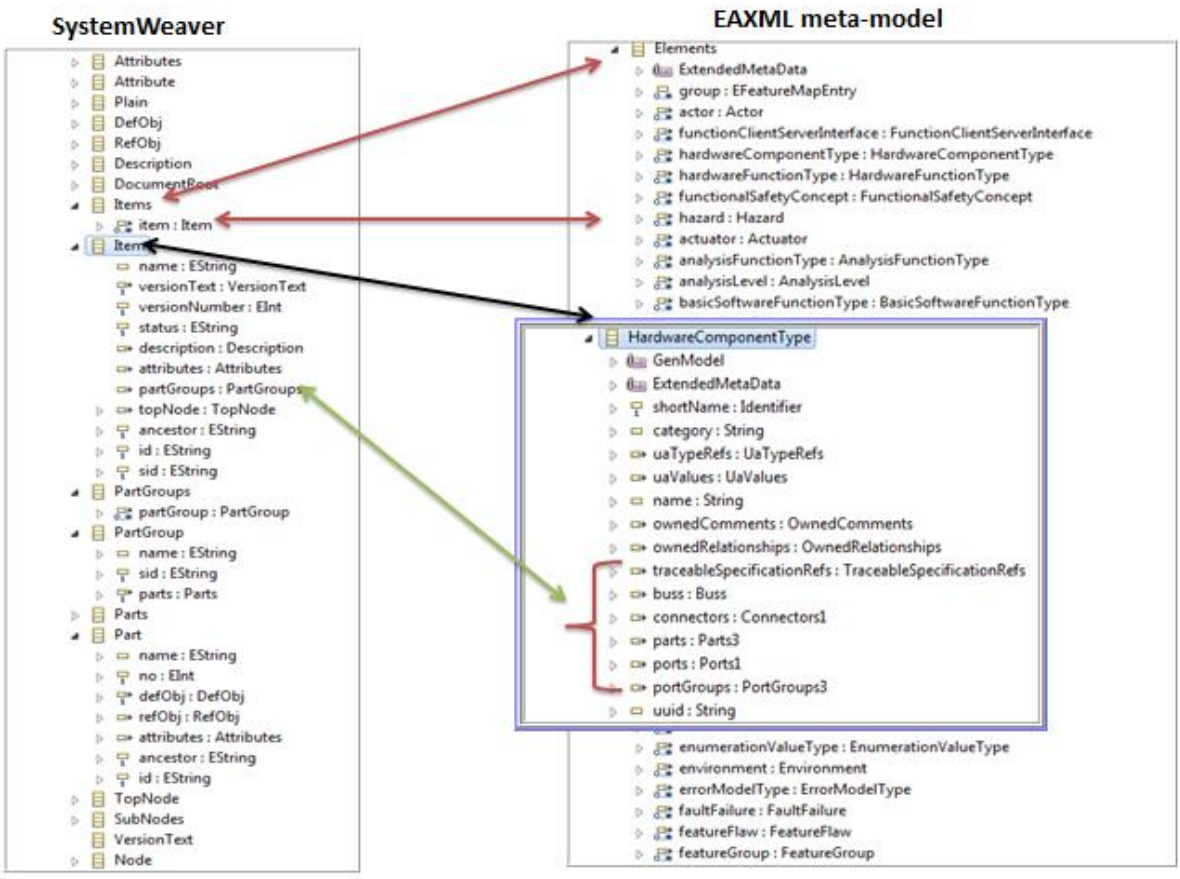

Figure 14: Meta-model comparison

This makes the transformation time taking to implement. The fact that EAXML explicitly describes its elements, leads to equivalent rules in the ATL transformation. Due to this reason a total of 997 ATL rules and 135 ATL helpers are implemented to support the covered part of the meta-model.

6.3 EAXML2SystemWeaver Transformation

The transformation from EAXML to SystemWeaver can be represented in the figure shown below.

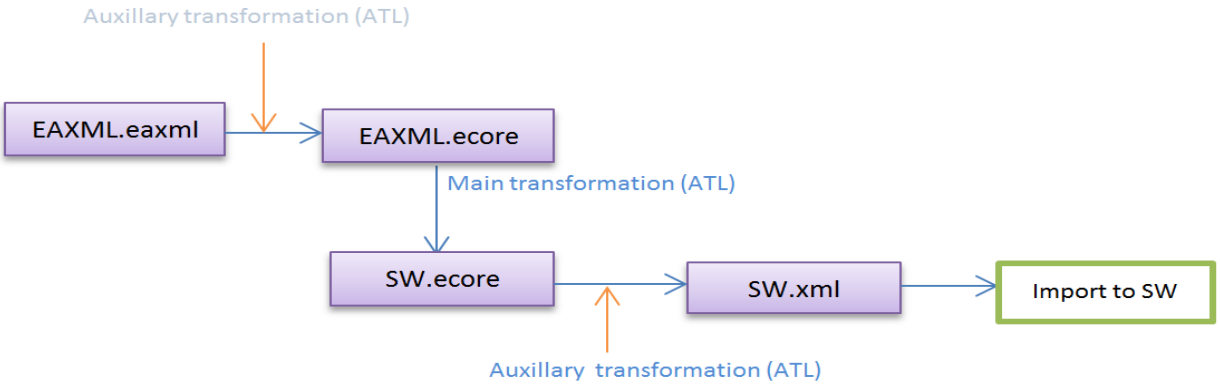

Figure 15: EAXML2SW Transformation Logic

The EAXML.xml is transformed to EAXML.ecore using an auxiliary transformation then the EAXML.ecore is transformed to SystemWeaver model, which results to an ecore model. Since ecore models cannot be imported to SystemWeaver, the auxiliary transformation is required. Generally, in total of three transformations are required to get the transformed model in the desired format which is xml. Mapping for the main transformation is shown on the table.

Name	Mapped To
TOP-LEVEL-PACKAGES	Items
SUB-PACKAGES	Item
EA-PACKAGES	Item
ELEMENTS	Item
HARDWARE-COMPONENT-TYPE	Item
NODE	Item
SENSOR	Item
POWER-SUPPLY	Item
ACTUATOR	Item
BUSS	Item
HARDWARE-CONNECTOR	Item
instanceRef	Node
UUID	Id
SHORT-NAME	name
isOfType	Type which is an Item
HARDWARE-COMPONENT-PROTOTYPE	part

6.4 SystemWeaver2EAXML Transformation

The figure shown below is a demonstration for the transformation between SystemWeaver and EAXML.

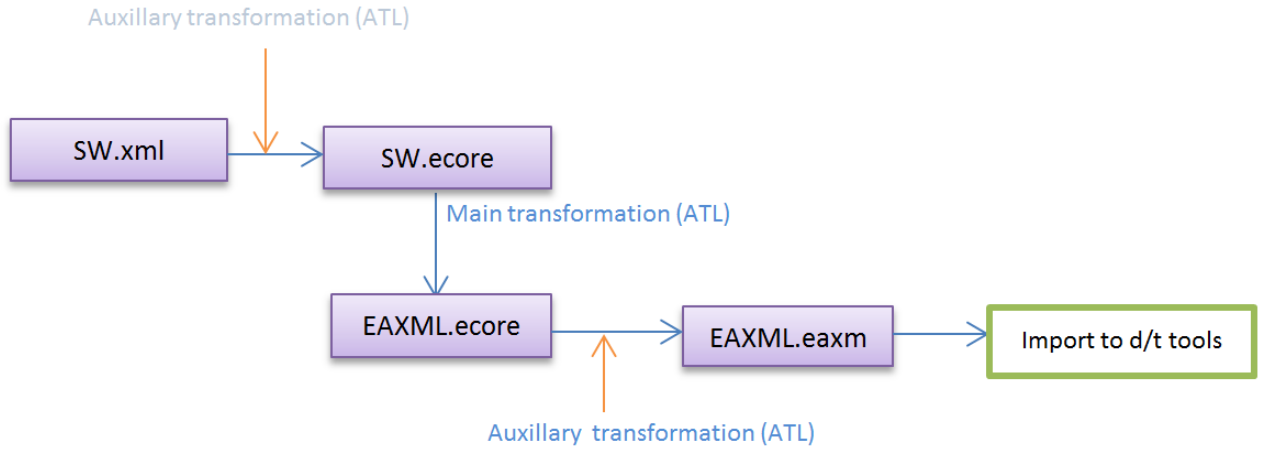

Figure 16: SW2EAXML Transformation Logic

The above transformation is divided into Auxiliary and main transformations. The main transformation is the transformation where different analysis is required. As the references and the size of the meta-model are big, the rules and mappings are more in number and in complexity than it is shown at the table.

Mapping for the main transformation of the export is shown below.

Name	Mapped to
Items	TOP-LEVEL-PACKAGES
Item	Elements, EA-PACKAGES, SUB-PACKAGES , Dependability etc
Part	HARDWARE-COMPONENT-PROTOTYPE, prototypes, ports etc
Id	UUID
Name	Name
shortName	SHORT-NAME
Node	instanceRef
Type	isOfType

6.5 Transformation Coverage of the EAST-ADL meta-model

As it is shown in the mapping principles section, the transformation has covered a big part of the EAST-ADL Specification. The import covers Infrastructure, System Model along with the three abstraction levels, Analysis level, Vehicle Level and Design level. The export covers Requirements, Dependability in addition to the imported elements. The picture below describes the parts the transformation supports to.

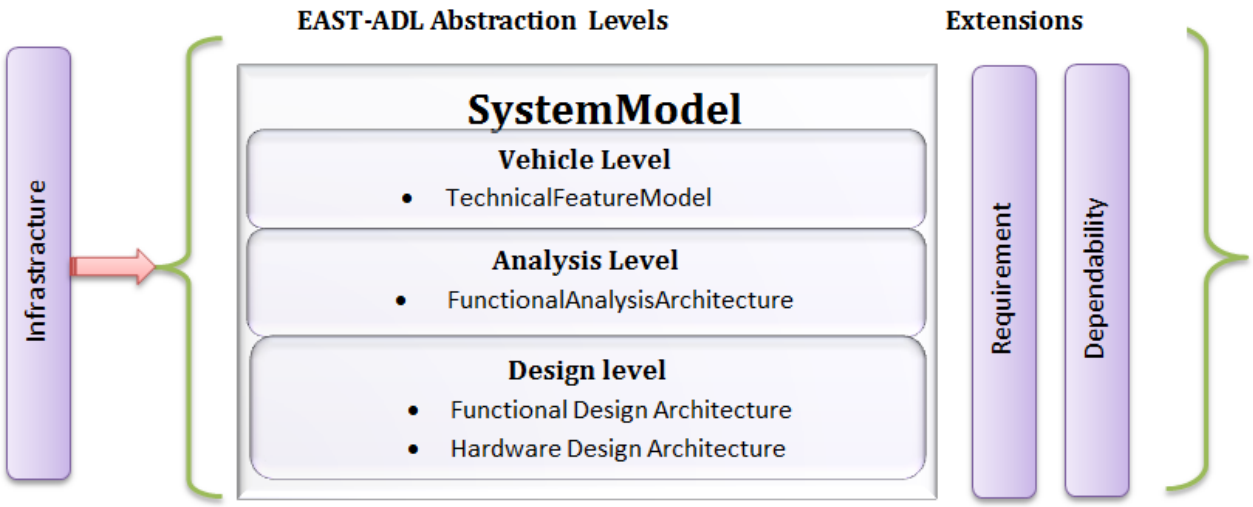

Figure 17: Transformation coverage of EAST-ADL Specification

7 Current status and Future plan

7.1 Current Status

According to the requirements listed on WP2, the following are achieved in this year

1. Meta-model implementation

   The different versions of EAST-ADL are implemented using SystemWeaver. The current version, M2.1.10, M2.1.11 and M2.1.12 are implemented.

2. Tutorial

   The final version of the tutorial for EAST-ADL implementation using SystemWeaver is uploaded and it can be found at SystemWeaver folder situated at WT5.3/SystemWeaver.

3. Transformation

   The model to model transformation between SystemWeaver and EAXML is implemented.

4. GUI for the transformation

   The GUI along with its manual, on how to use the transformation is ready. Application along with its instruction is provided at \WP5\WT5.3\SystemWeaver\SWTransformation V2.

5. Instance ref is handled in Systemweaver. Example: Errormodel on the context of dependability is implemented.
6. FDA to FAA realization and HAD to FDA allocation are implemented in Systemweaver
7. A new better graphical view for the HDA is implemented
8. A tutorial to instruct on how to develop error model and use the allocations in Systemweaver is included.

7.2 Future plan

1. Updating the XML exchange to M2.1.12 will be done in the Swedish research project Synligare.

8 Conclusions & Summary

The full M2.1.12 EAST-ADL implementation is now available in SystemWeaver. A meta-model based implementation, which provides strong typing mechanism, is applied to implement the EAST-ADL in System Weaver. The strong typing prevents users from making mistake while designing EAST-ADL models.

SystemWeaver is a collaborative environment where changes can be reverted or easily abandoned. It is possible for different users to work with different versions of the same element. This easily shows that it is possible to make an impact analysis by looking at the different versions and see what their differences with the help of SWExplorer tool. This makes it one of the best tools. Reusability is a core concept in SystemWeaver.

A bi-directional model transformation is available between SystemWeaver and EAXML models. ATL is used to realize the transformation. This transformation is meta-model based transformation and its output can be of different format models such as xml, xmi and ecore file formats. This can lead to a conclusion that SystemWeaver can now integrate with tools that can import EAXML.xml, EAXML.xmi and EAXML.ecore.

For more detailed information about mapping and other subjects see section "**mapping principles**" and "**Meta model mapping concepts for SystemWeaver**".

9 Reference

1. http://www.systemite.se
2. Nigsti Ayele, Investigating Model Transformation Technologies for Architecture Description Languages. Online available at <http://publications.lib.chalmers.se/records/fulltext/149219.pdf> accessed on 2012-08-21
3. SystemWeaver tutorial for EAST-ADL
4. Tolosa et al, Towards Meta-model Interoperability of Models through Intelligent Transformations S.Omatu et al (EDs.) : IWANN 2009, Part II, LNCS 5518, p. 312-322, SpringerLink – Verglag Berlin Heidelberg 2009
5. Malavolta et al. Providing Architectural Languages and Tools Interoperability through Model Transformation Technologies IEEE TRANSACTIONS ON SOFTWARE ENGINEEERING, VOL 36, NO. 1, JANUARY/FEBRUARY 2010