CHALMERS

Supporting Embedded Systems Development

Tool Support for EAST-ADL import of Modelisar FMU

Master of Science Thesis

SELÇUK ÇAVDAR

Department of Applied Information Technology
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden, 2011

Report No. 2011:049
ISSN: 1651-476

CHALMERS

Supporting Embedded Systems Development

Tool Support for EAST-ADL import of Modelisar FMU

SELÇUK ÇAVDAR

Technical Report No. 2011:049
Department of Applied Information Technology
CHALMERS UNIVERSITY OF TECHNOLOGY
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

1. Summary

Embedded Systems play an important role in nearly every type of product now and it is increasing its popularity even more. Thus, designing more reliable embedded systems in a shorter time of period has been a very important topic and research area for the past decades. Model Based Development is one of the most effective tools that researchers have come up with for this topic and is the new era of designing Embedded Systems more efficiently. The importance and application area of Embedded Systems has also been increasing rapidly in the Automotive Industry. Hence, it is in great interest to develop powerful Model Based Development tools for the design of Automotive Embedded Systems to decrease the development time and increase the safety and reliability of them. Moreover, Model Based Development provides reusability and makes the collaboration between the developers much easier, more reliable and more time efficient.

ATESST2 and ITEA2 are two projects which have taken place for research in "Model Based Development of Automotive Embedded Systems" and which resulted in EAST-ADL and Modelisar. The main aim of the Thesis was to analyse the possibility of a mapping between a Modelisar and an EAST-ADL model and then to implement a Modelisar→EAST-ADL model conversion tool.

The Thesis work resulted in obtaining an XML based tree editor in Eclipse for EAST-ADL models and a Modelisar→EAST-ADL model convertor application. Obtaining this conversion tool included the steps of defining Modelisar→ EAST-ADL Mapping rules, implementing these rules in ATL and forming an executable application.

The outcomes of the Thesis work has been used in Maenad, a European Commision FP7 project.

Keywords: Tool Integration, Model Transformation, EAST-ADL, Modelisar, Eclipse, EMF, ATL

2. Contents

3. Acronyms

Table 1. Acronyms.

VTEC	Volvo Technology Corporation
EAST-ADL	Electronic Architecture and Software Tools- Architecture Description Language

4. Introduction and Overview

Connecting Model-Based-Development tools with other Model-Based-Development and Simulation tools have always been an interest of the ongoing research about Embedded Systems. This interest arises of the fact that every Model Based Development Tool and Simulation Tool has different features and characteristics. Connecting a Model Based Development Tool to another one or to a Simulation Tool can give different advantages and can be done in different ways. In other words, each tool needs to work in cooperation with the artifacts created by other tools to increase the efficiency of development.

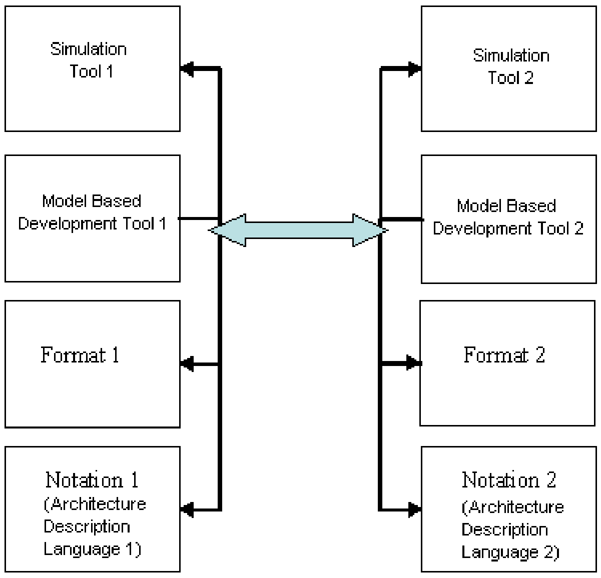

Simulation Tool 1
Model Based Development Tool 1
Format 1
Notation 1 (Architecture Description Language 1)
Simulation Tool 2
Model Based Development Tool 2
Format 2
Notation 2 (Architecture Description Language 2)

Figure 4.1/ 2 Model Based Development Tools with different characteristics

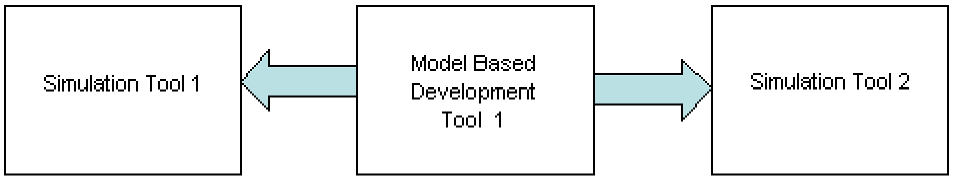


Figure 4.2/ A Model Based Development Tool that has integration with 2 Simulation Tools

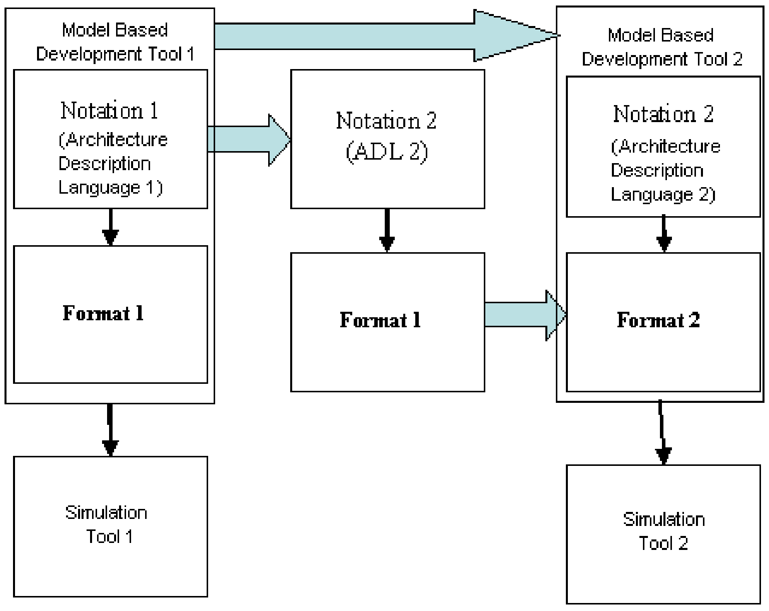

Model Based Development Tool 1
Notation 1 (Architecture Description Language 1)
Format 1
Notation 2 (ADL 2)
Format 1
Model Based Development Tool 2
Notation 2 (Architecture Description Language 2)
Format 2
Simulation Tool 1
Simulation Tool 2

Figure 4.3/ Integration Process of a Model Based Development Tool with another

As seen in Figure 4.1 Model Based Development Tool 1 and 2 have different characteristics as they have different Notations, Simulation Tools and Formats. One important note to take here is that Notations in our context more specifically mean Architecture Description Languages. So connecting these 2 tools extends their capability and expressiveness. A model designed in Model Based Development Tool 1 can be simulated by Simulation Tool 2 or can be expressed by Architecture Description Language 2 after Model Based Development Tool 1 and 2 are connected to each other. Connecting different architecture description languages is valuable as these languages can gain new capabilities by it. As in Architecture Description Languages, also different Simulation Tools have different strong and weak points. Furthermore, there are lots of Model Based Development tools already developed for automotive industry too and it is always good to connect different tools to each other for maximizing the reusability of already designed models. So, connection of different tools can put in extra freedom in the design process of a model. Since different tools have different strengths in modelling a system, a user can design some aspects of the system model in one tool and send it to another tool for designing/simulating some other aspects of the model which gives flexibility. Another possible approach is connecting a Model Based Development Tool with one or several Simulation Tools like shown in Figure 4.2. This would support the user with different methods of analysis and verification of the various parts of the models. As different Simulation Tools could be more specialized in one aspect of the models, having compatibility with several of them is definitely an advantage. If 2 Model Based Development Tools are connected to each other then this means one can transfer one model designed in 1 of them into the other with using a common Simulation Tool as a link. Another possible linking way could be as shown in Figure 4.3. Here Model Based Development Tool 1 uses Architecture Description Language 1 and Format1 and is compatible with Simulation Tool1. Similarly Model Based Development Tool 2 uses Architecture Description Language 2 and Format2 as its modelling base and is connected with Simulation Tool 2. These 2 Model Based Development Tools can be linked by having a mapping between Architecture Description Language 1 and 2 and afterwards changing Format 1 with Format 2. So Model1 designed in Model Based Development Tool 1 would go in a conversion from Architecture Description Language 1 to 2 and then change its format from Format1 to Format2. This transformation of the model could have been done in different ways like changing only the Architecture Description Languages, Formats, etc. of the model depending on the differences between Model Based Development Tools. When these additional efforts are given for the conversion of models, as a result, all the models designed in Model Based Development Tool 1 will have the opportunity to use all the supporting tools of Model Based Development Tool 2, to be analyzed from a different angle and be reused in Model Based Development Tool 2 as a new model or an addition to a previously designed model. In short, the features obtained by integrating Model Based Development and Simulation tools gives a lot of flexibility to the user and supports reusability in the design process.

During the development process of such integrations, It is beneficial to make a research on similar efforts in this area for saving time and for increasing the capability of integration. For this intent, 2 different projects that have been going on in cooperation with VTEC have been selected. These 2 projects' main scope has been integrating Simulink with Papyrus and Simulink with SystemWeaver respectively. In short, Papyrus is a platform that aims to offer an environment for editing all kinds of EMF based models and Systemweaver is a product life cycle management platform.

Linking Simulink with Papyrus is an approach leaded by KTH. Papyrus, in the scope of this project is used to develop EAST-ADL models with UML profiling. So Papyrus is the Model Based Development Tool, the Architecture Description Language is EAST-ADL and the Format is UML. The project applies different processes to a model (an EAST-ADL model created with Papyrus and profiled with UML) and the building blocks of these processes are EMF, Simulink and EAST-ADL metamodels, UML, Java, ATL, Eclipse and Matlab.

Connecting SystemWeaver with Simulink has more or less the same goal as KTH's implementation; connecting a Model Based Development Tool to a Simulation Tool. This project also has a similar mindset and building blocks behind the conversion. The building blocks of the project consist of Simulink and EAST-ADL MetaModels, SystemWeaver MetaMetaModel, XML, SystemWeaver and Matlab.

These 2 projects had similar development steps as in our project. The difference was that, their aim is to connect a Model Based Development Tool to a Simulation Tool where our aim is to connect a Model Based Development Tool to another one. But our project and these 2 others correspond in their similar implementation phases. Furthermore one of the future plans of our project has been put as as integrating Simulink with our conversion tool which is the main topic of the other 2 projects. The current version of our project uses EMF, FMI and EAST-ADL metamodel, XML, Java, ATL, VSA, Modelisar and Eclipse.

5. Background

This part of the report will give you an insight about some of the topics that are needed to be familiar with to understand the work done in the thesis.

5.1 Related Projects

Modelisar, Atesst2 and Maenad projects are the closely related projects with the thesis work. Modelisar and Atesst2 projects' results make up the core of the thesis work and Maenad is the project which is contributed to.

5.1.1 MODELISAR(MODELIca-autoSAR)

Modelisar is an ITEA2 project which aims in significant improvement in the design of systems and embedded software in the vehicles. It supports Vehicle Functional Mock-up, a next generation of methods, standards and tools to support collaborative design, simulation and test of systems and embedded software. Functional Mock-Up Interface is one of the important results of the project which also has a very significant role in the thesis work.

5.1.2 ATESST2(Advancing traffic efficiency and safety through software technology phase 2)

Atesst2 is a completed European Commision FP7 Project. The objective of this project was to refine and improve EAST-ADL. The imporvement part is concentrated on extending the language in modelling cooperative active safety systems. As a result of the project:

→ EAST-ADL has been refined for capturing the requirements, characteristics and configurations of cooperative systems and the related analysis, validation and verification of them.
→ A guideline has been defined to use EAST-ADL for construction, validation and reuse of automotive embedded software.
→ EAST-ADL domain model has been modified for compatibility with AUTOSAR.

The current version of EAST-ADL which is used in the Thesis too is a result of this project. Also Papyrus and KTH Simulink exchange plug-in are 2 of the several outputs of the project.

5.1.3 MAENAD(Model-based Analysis & Engineering of Novel Architectures for Dependable Electric Vehicles)

Maenad is an ongoing European Commision FP7 Project. Maenad is also based on EAST-ADL. This time the main scope is to refine EAST-ADL to meet the challenges in the design of Fully Electricle Vehicles. The official main objectives of the project are as follows:

→ Provision of support for the automotive safety standard ISO 26262
→ Provision of capabilities for prediction of dependability & performance
→ Provision of capabilities for design optimization
→ Demonstration of project results in a practical electrical vehicle design

The thesis work took place for contribtuion to Maenad project.

5.2 Modelling Technology

Model Driven Engineering is the main concentration and contribution area of the thesis work. MetaModelling, MetaMetaModelling and Model Transformation topics are the main notions that have been used throughout our studies.

5.2.1 Model Based Development/ Model Driven Engineering

Model Based Development and Model Driven Engineering aims mainly on increasing the productivity. They help the development teams to co-operate with each other easier and to increase the compatibility and reusability of systems. Simply they help to increase the efficiency of the design process by means of offering a domain model and tools such as development environments to the developers. Figure 5.2.1 shows a typical software/hardware development process. One can apply Model Based Development and Model Driven Engineering approaches through all the steps of this process, from project definition to project test and integration. For instance EAST-ADL aims to cover earlier phases in the development process where AUTOSAR more tries to cover the implementation phase.

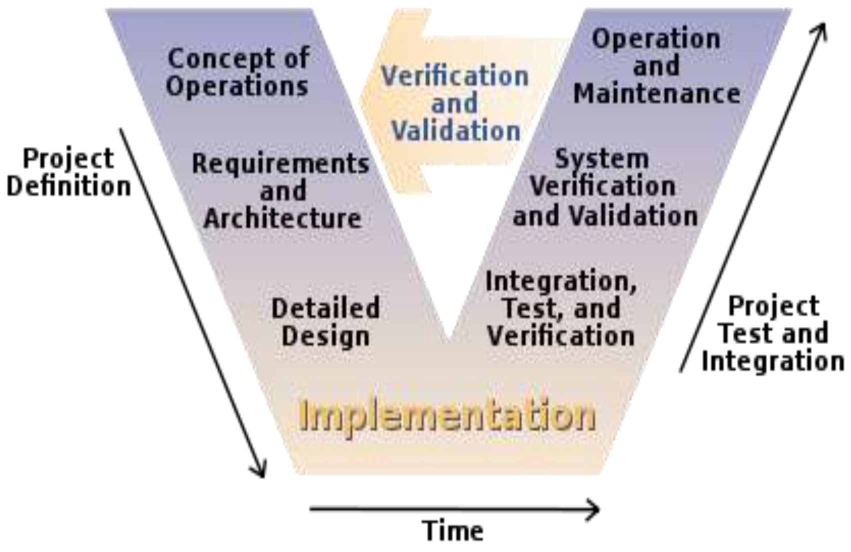

Concept of Operations
Verification and Validation
Operation and Maintenance
Project Definition
Requirements and Architecture
System Verification and Validation
Detailed Design
Integration, Test, and Verification
Project Test and Integration
Implementation
Time

Figure 5.2.1/ V Model(Software Development Process)

5.2.2 MetaModel/MetaMetaModel

Metamodeling, is the development of the structures, rules and constraints for modeling a class of problems. A MetaModel determines the possible structures of the model. The MetaModel concept has a very parallel charactersitic with programming languages. For instance, a programming language is the grammar of a computer program where a MetaModel is the grammar of constructing a model. A model that respects the semantics defined by a metamodel is said to conform to this metamodel. So as a computer program should conform to its language, a model should conform to its MetaModel as well. MetaMetaModel is one level more abstract than MetaModel. MetaMetaModel determines the semantics of the MetaModel just as MetaModel does for its Models. The relation between these concepts can be seen more clearly from the Figure 5.2.2. In M0 level, there is a video model which is an instance of Video element in M1. The video element is an instance of attribute in M2 and attribute is an instance of Class in M3. So, M0 is the model level, M1 is the Meta-Model level, M2 is the MetaMetaModel level. In this example, M3 is the MetaObjectFacility 2.0 which is formed by OMG group and determines the structure of M2 which is in this case UML 2.0. M2 could have been another language other than UML which conforms to MOF. So MOF puts some limits on forming models conforming to it but one can create different models (languages) using MOF. One can also create different M1 and M0 models as well, only limitation is to make sure that they conform to their corresponding M2 and M1 models. A typical way of representing the information in MetaModels and MetaMetaModels is using XSD files which is also used to reprsent the MetaModels that are used in the thesis work.

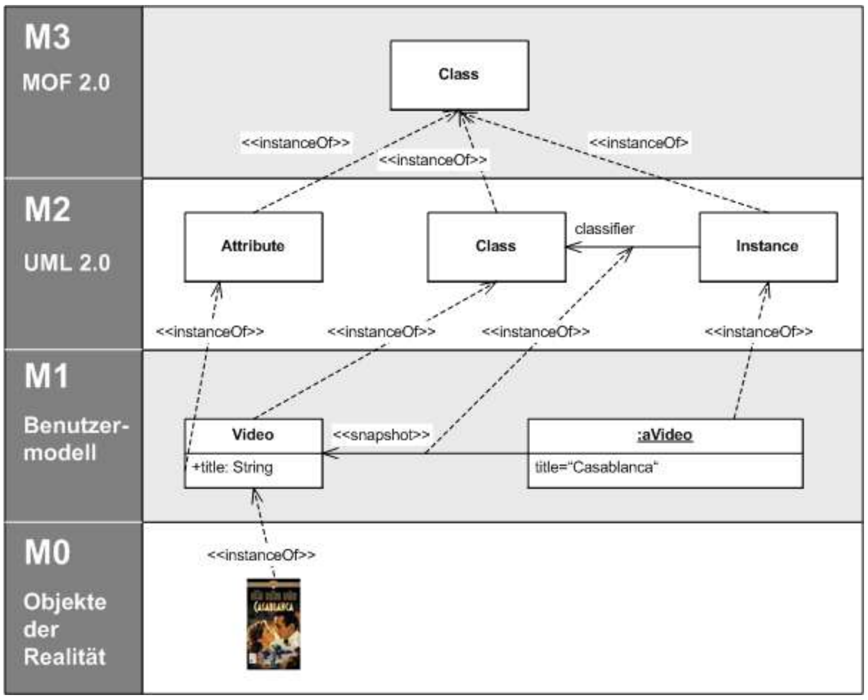


Figure 5.2.2/ MetaMetaModelling Concept Example

5.2.3 Model Transformation

Model transformation is the way for specifying how a target model will be formed from a source model. For this purpose, the developers must match source model elements to initialize the target model ones. Formally, a simple model transformation has to define the way for generating a model Mb, conforming to a metamodel MMb, from a model Ma conforming to a metamodel MMa. The transformation rules can also be seen as a model since it has to conform to a transformation metamodel that defines the model transformation semantics.

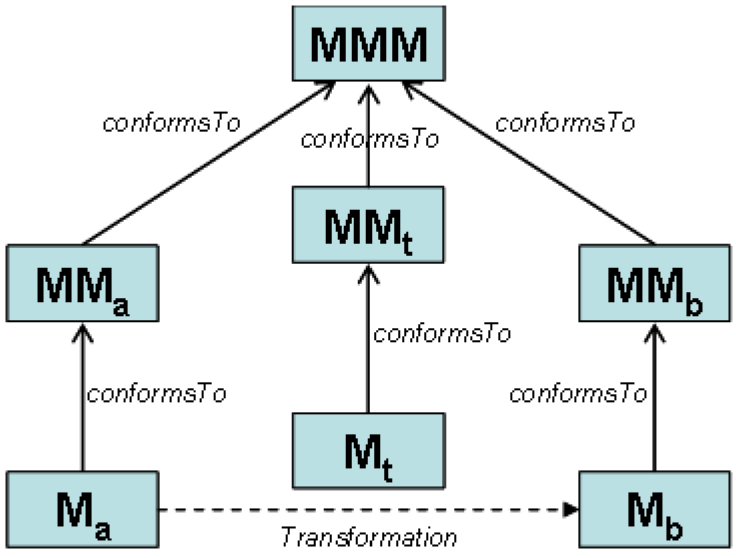


Figure 5.2.3/ General structure of Model Transformation

Figure 5.2.3 shows the general structure of a model transformation. Ma is a source model which conforms to MMa which is the MetaModel of the soruce model. Mt is the transformation rules which will match the source elements to the target ones. Mb is the obtained target element by using Mt with Ma. Mb should also conform to MMb (target model's MetaModel), hence Mt is responsible of forming a valid target model. MMa, MMt and MMb have a common MetaMetaModel which is MMM that they conform to. MMM is the Ecore MetaMetaModel, MMa is the FMI MetaModel, MMt is the ATL MetaModel, MMb is the EAST-ADL MetaModel, Ma is an FMI model, Mt is the ATL transformation rules and Mb is the EAST-ADL model obtained after the converesion in our case.

5.3 FMI(Functional Mockup Interface)/FMU(Functional Mock-up Unit)

FMI defines an interface to be implemented by an executable called FMU. The FMI functions are called by a simulator to create one or more instances of the FMU, called models, and to run these models, typically together with other models. Models are described by differential, algebraic and discrete equations and they are independent of the target simulator as they do not use a simulator specific header file. An overview of the data flow of an FMU has been shown in Figure 5.3.

The purpose is to describe dynamic systems with these models and provide an interface to run them as needed in different simulation environments. FMI is distributed in one zip-file. The zip-file contains

- The Model Description File (XML format): This xml-file contains the definition of all variables in the model in a standardized way.
- The C sources of the Model Interface (including the needed run-time libraries used in the model) and/or Dynamic link libraries (DLL) for one or several target machines. This solution is especially used if the model provider wants to hide the model source code to secure the contained know-how. A model may contain physical parameters or geometrical dimensions, which should not be open. On the other hand, some functionality requires source code.
- Additional model data (like tables, maps) in model specific file formats.

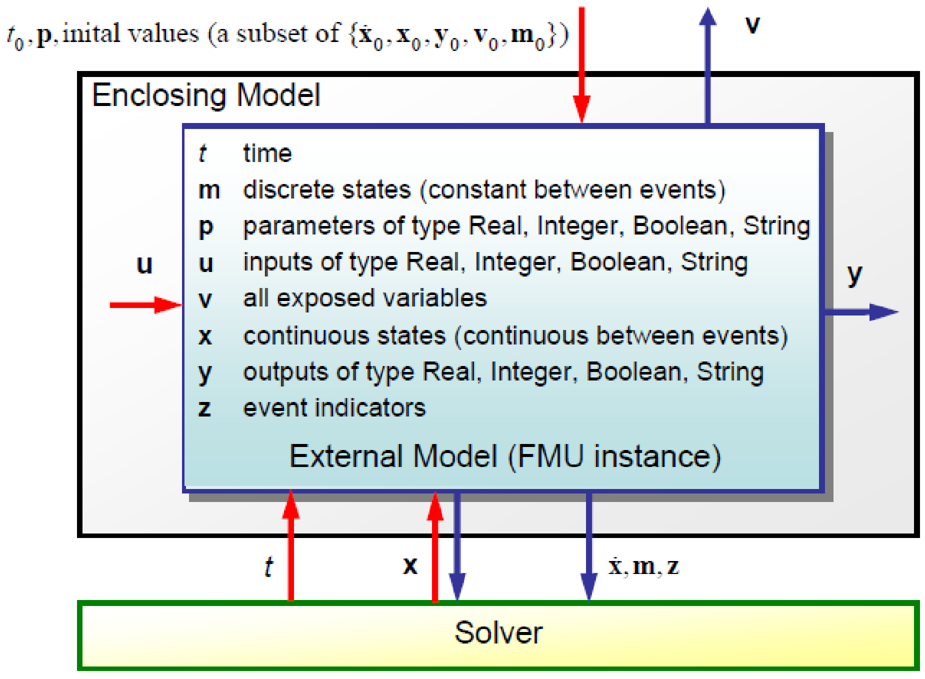

$t_0, \mathbf{p}$, inital values (a subset of $\{\dot{\mathbf{x}}_0, \mathbf{x}_0, \mathbf{y}_0, \mathbf{v}_0, \mathbf{m}_0\}$)
v
Enclosing Model
t time
m discrete states (constant between events)
p parameters of type Real, Integer, Boolean, String
u inputs of type Real, Integer, Boolean, String
v all exposed variables
x continuous states (continuous between events)
y outputs of type Real, Integer, Boolean, String
z event indicators
External Model (FMU instance)
u
y
t
x
$\dot{\mathbf{x}}, \mathbf{m}, \mathbf{z}$
Solver

Figure 5.3/ A schematic view of an FMU and its interface

5.4 AUTOSAR(AUTomotive Open System ARchitecture)

Autosar is an industrial software architecture standard. The motivation behind starting the Autosar project was to lower down the cost of the design phase of E/E systems for automotive industry by having a standardized infrastructure for the management of functions. The cost of the design phase became an important issue since more and more E/E systems has been introduced to the vehicles in the last 2 decades which made the engineering of them more complex and a longer process to go in. In this process, automotive manufacturers work with different work groups inside the company as well as with the others such as suppliers which makes collaboration and integration important issues. Refinements to the system take place during the development cycle. Also there are increasingly more challenging regulations for safety and environmental care which creates the need of improvement both in the SW and HW aspects of the E/E system continually. Autosar's aim is to provide a solution to these modularity, scalability, transferability and re-usability issues with a standardized software infrastructure.

5.5 EAST-ADL(Embedded Architectures and Software Technologies - Architecture Description Language)

EAST-ADL is an architecture description language which aims to introduce more abstract levels than AUTOSAR. Also it aims to complement AUTOSAR in the development of automotive embedded systems. In short, the premise behind EAST-ADL is the need of managing the overall engineering information to control system definition.

The model organization of EAST-ADL can be seen in the Figure 5.5. There are four abstraction levels for describing software and electronics based functionality of a vehicle. The embedded system is complete on each abstraction level and the different parts of it can be connected with using various traceability relations.

VehicleLevel lets the user to represent the features of the vehicle from top level perspective. AnalysisLevel models the electronic functionality in an abstract form. DesignLevel includes the implementation oriented aspects and ImplementationLevel is the link of EAST-ADL with Autosar. The extensions are for giving further modelling support in different aspects of the embedded systems.

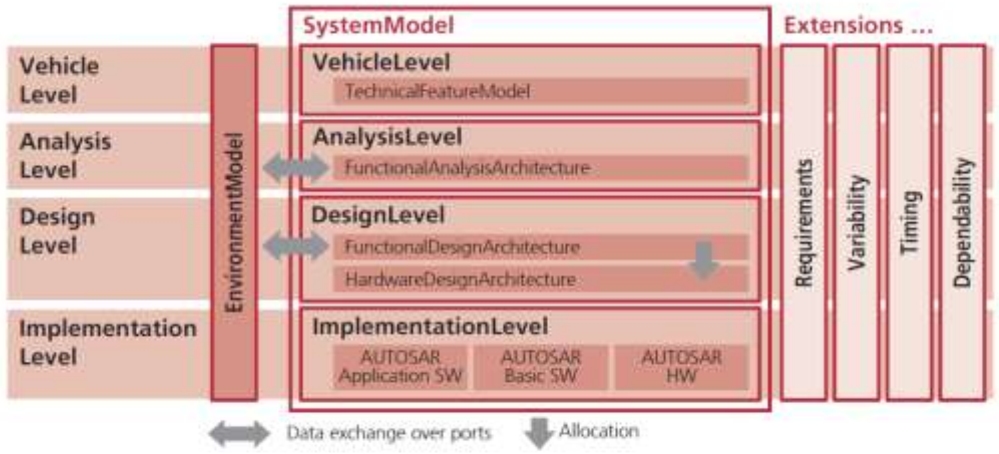


Figure 5.5/ EAST-ADL abstraction levels

5.6 EAST-ADL Behavioural Representation candidates

UML (Unified Modelling Language), XML (Extensible Markup Language) and XMI (XML MetaData Interchange) are used as behavioural representation candidates of EAST-ADL. UML is extensible with 2 mechanisms which are profiles and stereotypes. UML2 profile is applied to EAST-ADL domain with using these mechanisms. XML is used in EAST-ADL for definining EAXML, a model exchange format. Also, Papyrus is a development environment of UML profiled EAST-ADL models where as the generated Editor in the thesis work is an XML based one. The FMU2EA tool uses both XMI and XML for the representation of EAST-ADL models.

5.7 EAST-ADL Tooling

VSA (Volcano Vehicle System Architect), Papyrus and SystemWeaver are some of the EAST-ADL related tools.

5.7.1 VSA

VSA is mainly developed for AUTOSAR based systems. Its aim is to give support for designing SW and HW architectures and the interaction between them for automotive systems. It also has a feature of forming XML based EAST-ADL models. The formed EAST-ADL models with VSA are EAXML extensioned files and it is known that they were in the desired format. So VSA is used to verify the models that are obtained from the generated Editor and the FMU2EA tool in terms of their structure and format as they are XML based too.

5.7.2 PAPYRUS

Papyrus is a tool for modelling in UML2. It also has an EAST-ADL modelling add in. So it is a UML based EAST-ADL model development environment. The reason Papyrus has been in attention in the thesis work is that KTH's Simulink Plug-in application, a project that is considered as an important related one, is designed for Papyrus.

5.7.3 SYSTEMWEAVER

SystemWeaver is an engineering information platform designed for easier product life-cycle management (PLM). SystemWeaver offers the possibility of establishing one single source for all E/E systems information for all developed products. The users of the system are found in different sites within many diverse organizational areas, and in different project phases. The "Concurrent Engineering" approach is implemented in SystemWeaver to support these developers with up-to-date information and with data integrity. SystemWeaver offers a Simulink exchange feature too which gives the SystemWeaver users the capability of transferring their SystemWeaver models to Simulink. SystemWeaver has a MetaMetaModel which enables the developers to develop a MetaModel on it and afterwards to use this MetaModel for managing corresponding models in SystemWeaver. A part of the EAST-ADL MetaModel has been applied on SystemWeaver as well. For its connection with Simulink and EAST-ADL, SystemWeaver is considered as an important related approach for the thesis work and therefore an analysis of it has been done to collect more ideas especially on its Simulink integration feature and on its way of supporting the development of EAST-ADL models.

5.8 Platform Technology

5.8.1 ECLIPSE

Eclipse is a multi language software development environment. It has an extensible plug-in system. Eclipse itself provides most of its features with providing plug-ins that run on top of its runtime system. So users of Eclipse can develop their own applications and introduce them as plug-ins to Eclipse which makes these applications to become a part of the environment. Eclipse supports a number of powerful tools in modelling context hence it has an important role both in the thesis work, ATESTT2 and MAENAD projects.

5.8.2 EMF(Eclipse Modelling Framework)

EMF is a modelling framework. It has code generation capabilities for enabling viewing and editing of models. It is a very strong tool and played an important role through out the thesis work. EMF was the key element in obtaining the FMI and EAST-ADL MetaModels and the EAST-ADL Editor.

5.8.3 ATL

ATL is a model transformation language that enables to specify how target models can be produced from source models. ATL is composed of helpers, attributes and rules.
ATL helpers can be seen as equivalent to Java Methods. They are suitable for defining operations and can be called from different points of ATL code.

ATL supports both declarative and imperative programming by ATL rules. There are 3 kinds of ATL rules which are matched, called and lazy rules. Matched rules and lazy rules enable the programmer to do declarative programming whereas called rules are used for imperative programming.

Matched rules find the specified source model elements and forms target model elements with these. These rules specify how the target model elements should be created from source model elements. Lazy rules are similar to matched rules but they are only executed when they are called from an ATL imperative block. Called rules should be called from an ATL imperative block as well and they let to create target models by imperative code.

6. Problem approach

This section gives an overview of the work flow taken in the thesis. It also includes a detailed explanation of the implementation steps taken for forming the FMU2EA tool and the EAST-ADL Editor.

6.1. Work Flow

The general approach taken for the whole project can be divided into 2 big modules as seen in Figure 6.1.1. The steps that are planned to be taken can be summarized under the headings that can be found inside the modules. The applied approach evolved with time and so has some differences with this overview of the work plan. For instance, in the beginning finding new solution paths/changing the selected solution was a part of the general plan but in the end there remained no need to change the selected solution.

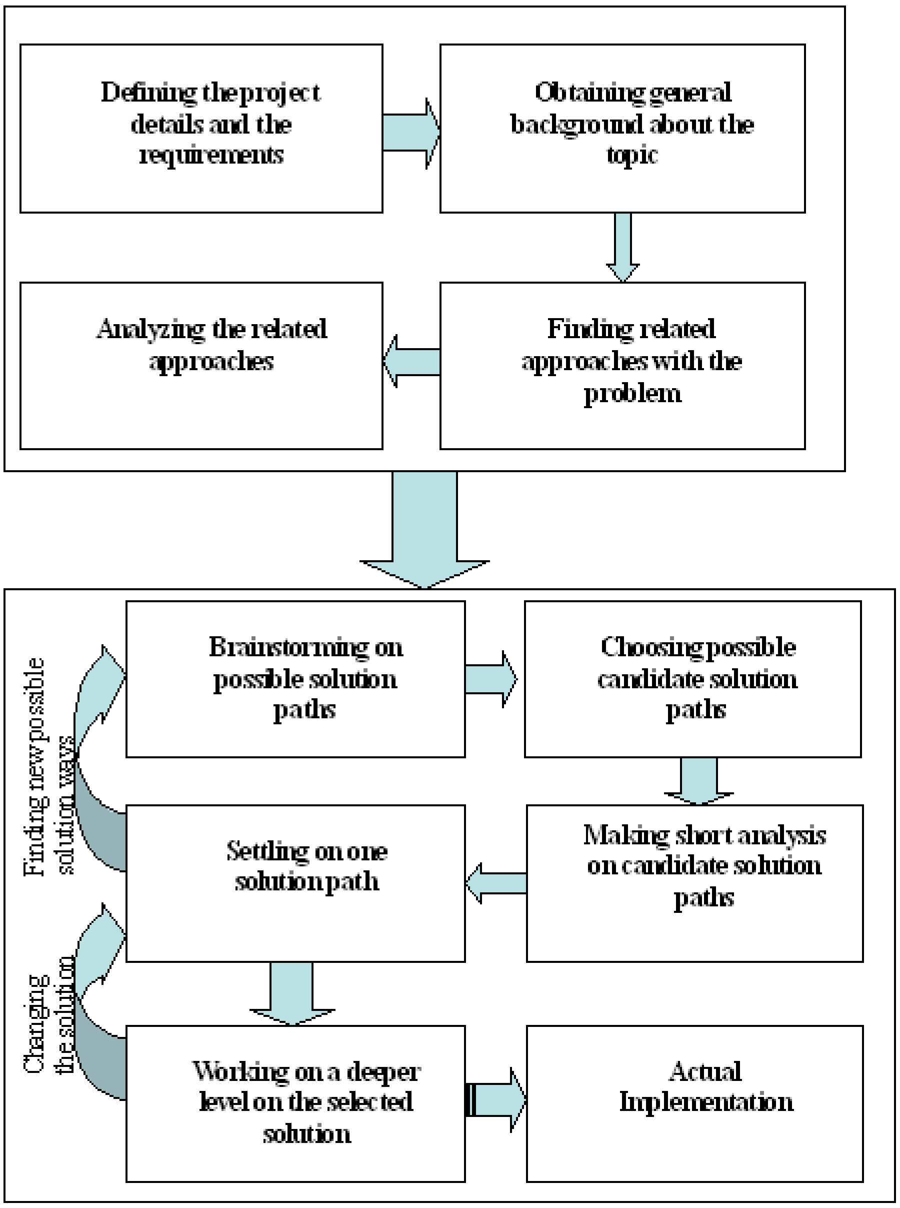

Defining the project details and the requirements
Obtaining general background about the topic
Analyzing the related approaches
Finding related approaches with the problem
Brainstorming on possible solution paths
Choosing possible candidate solution paths
Finding new possible solution ways
Settling on one solution path
Making short analysis on candidate solution paths
Changing the solution
Working on a deeper level on the selected solution
Actual Implementation

Figure 6.1.1/ An Overview of the Work Plan

In Figure 6.1.2, the process followed in “Working on a deeper level on the selected solution” and “Actual Implementation” parts of the work flow that are also the last two steps seen in Figure 6.1.1 are shown in detail. The first phase (first big box) in this figure shows the work done to test the selected solution path in order to see if it will be a preferable choice. The other 2 phases shows the steps taken for the actual implementation. Further discussion about the work flow takes place in the rest of this part.

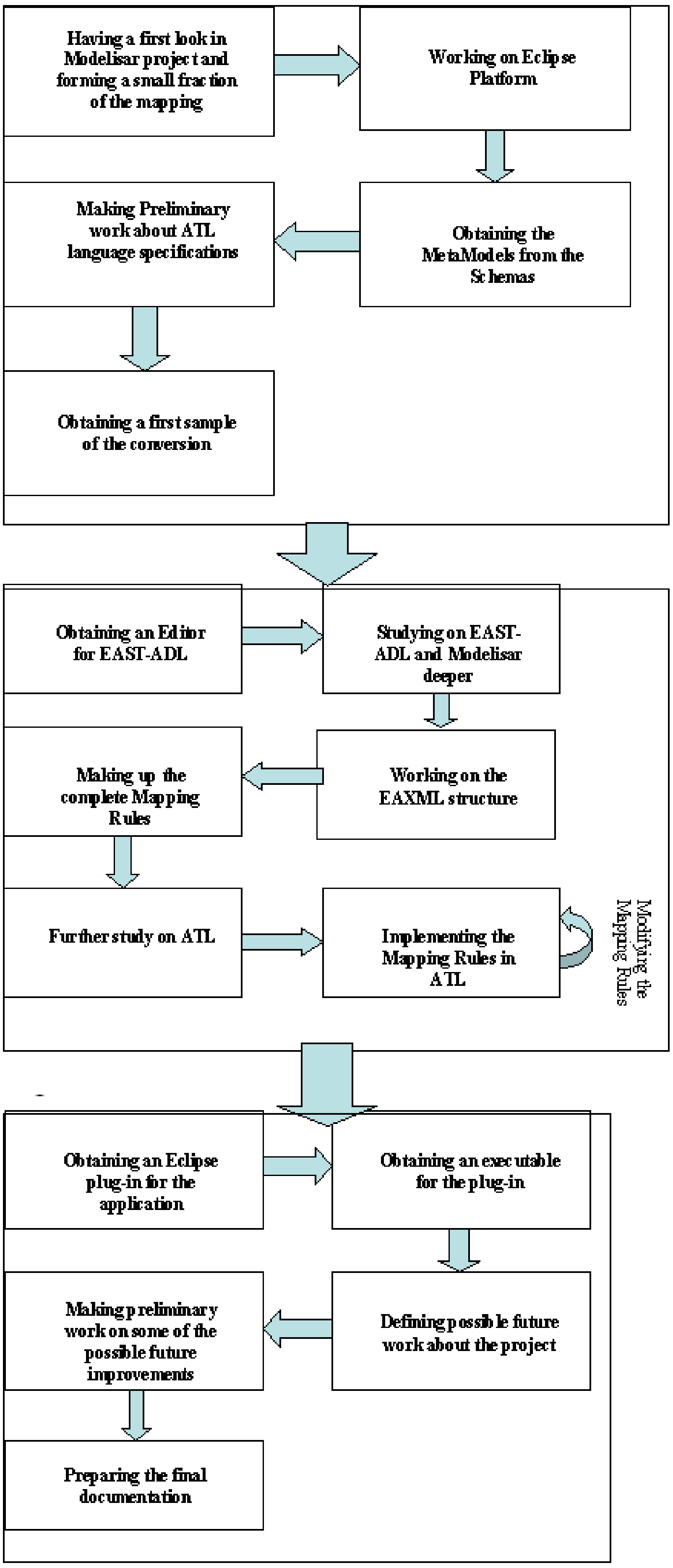

Having a first look in Modelisar project and forming a small fraction of the mapping
Working on Eclipse Platform
Obtaining the MetaModels from the Schemas
Making Preliminary work about ATL language specifications
Obtaining a first sample of the conversion
Obtaining an Editor for EAST-ADL
Studying on EAST-ADL and Modelisar deeper
Working on the EAXML structure
Making up the complete Mapping Rules
Further study on ATL
Implementing the Mapping Rules in ATL
Modifying the Mapping Rules
Obtaining an Eclipse plug-in for the application
Obtaining an executable for the plug-in
Defining possible future work about the project
Making preliminary work on some of the possible future improvements
Preparing the final documentation

Figure 6.1.2/ A more detailed look into the work done for the selected solution path

6.1.1 First Phase

The thesis work started with defining the characteristics and limitations of it. The first plan was to obtain some kind of tool that makes a conversion from a Modelisar model to an EAST-ADL model. It is planned to implement an interface for the conversion tool to make it more user friendly. Eclipse is decided to be used as the development platform since this project is a part of Maenad project which uses Eclipse as well. Obtaining XML based final models which is in the structure of EAXML format was aimed. Additionally, supporting model exchange between the built tool and Simulink is put as a considerable future extension of the tool which was a fact that should be caught in mind during the development. The thesis work is named as FMU2EA project.

After deciding on the requirements, the project flow went on with obtaining a general background about the possible building blocks of the project. Since the main interest was in EAST-ADL, gaining background step has started with learning this language. Model-Based-Development and MetaModeling concepts were studied. Also a quick look on XML, Modelisar and Eclipse platform have been taken.

Gaining the fundamental information moved the project to finding the related approaches. After some short search, 2 projects have been picked up for deeper analysis. They were selected because of having EAST-ADL as their core. Moreover, they had Simulink integration which is a possible future focus of the project. So these approaches were thought as having suitable characteristics to obtain an overview of FMU2EA project. As told before, “KTH Simulink Exchange” application, which has been developed in Eclipse environment as well, was one of the selected approaches. The other selected project was Systemite AB’s SystemWeaver platform with the Simulink exchange plug-in.

After finding these 2 related projects, detailed analysis of them have been made. First, their building blocks have been given attention. Papyrus, ATL and UML were studied in the light of KTH’s project in addition to the previously studied topics. In connection with Systemite’s efforts, SystemWeaver and MetaMetaModeling concept were studied. Secondly, their structures have been looked through. Close look have been given to their Simulink integration and their way of handling EAST-ADL models.

6.1.2 Finding Possible Paths

Obtaining an understanding of the related projects drew a clearer picture of the possible structure of the FMU2EA. A number of brainstorming sessions have been gone through which ended up with several ideas. Among these ideas, 1 of them was proposed as a highly potential candidate solution for implementing the structure. This solution included some decision points such as choosing the language to use for the application of the mapping rules between FMU and EAST-ADL. The general overview of the plan looks like as in the Figure 6.1.2.1.

It has been planned to support a user-interface that will let the user to choose a Modelisar model from the file system. This interface could be embedded to Eclipse platform or can take another formation depending on the preference later. The interface would also let the user to specify the location of the output model generated by tool. After the user selections, the corresponding EAST-ADL model should be created in the previously specified location. It was known that this conversion would extract information from an XML file and would create another XML file depending on the extracted information. This process means that there was a need of mapping rules that shows which element of Modelisar corresponds to which element in EAST-ADL. These mapping rules should have been implemented in Eclipse but how it will be done has been a discussion topic. One of the proposals was using Java to parse the Modelisar Model’s XML and create another XML that will correspond to and EAST-ADL model. Another significant proposal was to use ATL to make the conversion. Java had the advantage of being a known programming language which makes it a more suitable selection for making modifications by other people on the previous studies in the future. Also it had strong features which could have been used in case of need during the project. In the other hand ATL was a dedicated language to Model-to-Model transformation. Also, KTH’s simulink exchange project used ATL for making a model conversion. But, there were doubts about the capabilities of ATL since it was still a research project. Moreover, learning and implementing ATL language would take extra time as it had a specific structure unlike to Java, C, etc. which would need extra effort for getting used.

After some evaluation, it has been decided to use ATL, since KTH used ATL too in their “Simulink-Exchange” project as KTH already evaluated this language and also it was favourable to have compatibility with this tool as there can be a KTH-VTEC collaboration about this project in the future as KTH is a partner in Manead. But after this evaluation, it has been noticed that ATL has no good source of documentation and have still bugs. Additionally as it is an evolving language, an implementation running in an older version has the risk of not being able to run in a newer one. Still, ATL seemed like a good choice as compatibility was a more prioritized issue. Also, this project was a research project itself too so there was no problem to put time on examining the capabilities of ATL as it would give feedback for the future projects for showing whether if it is preferable or not to use ATL.

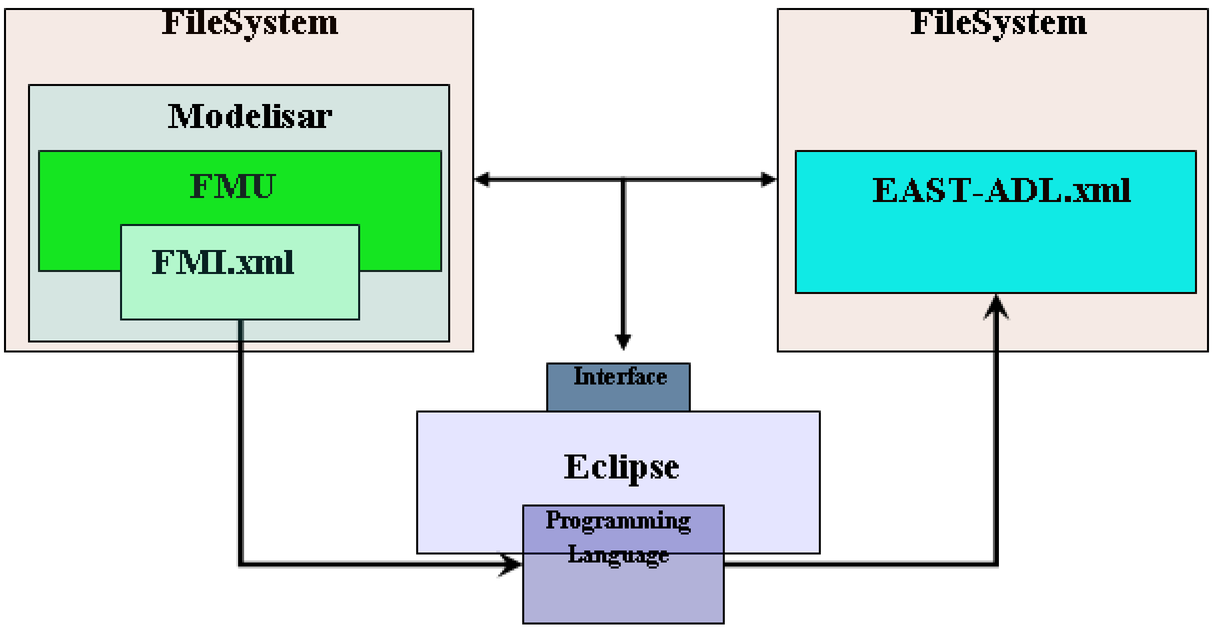


Figure 6.1.2.1/First state of the interaction planned between the components of the Tool

6.1.3 Working on the Selected Path/Software Development Process

Before starting to work on the planned solution, a general background refinement has been done. For this purpose, a general study has been done on the Modelisar project. Secondly, a closer look has been given to ATL structure and its working principle. Also, some previously completed ATL projects have been looked into for getting a better idea. Additionally, Eclipse platform and EMF (Eclipse Modeling Framework) have been studied in a deeper level. Especially, EMF is a very complex framework and a powerful tool so special attention has been given to it. After this second background study, a more detailed overview of the process that will take place in Eclipse has been obtained as can be seen in Figure 6.1.3.

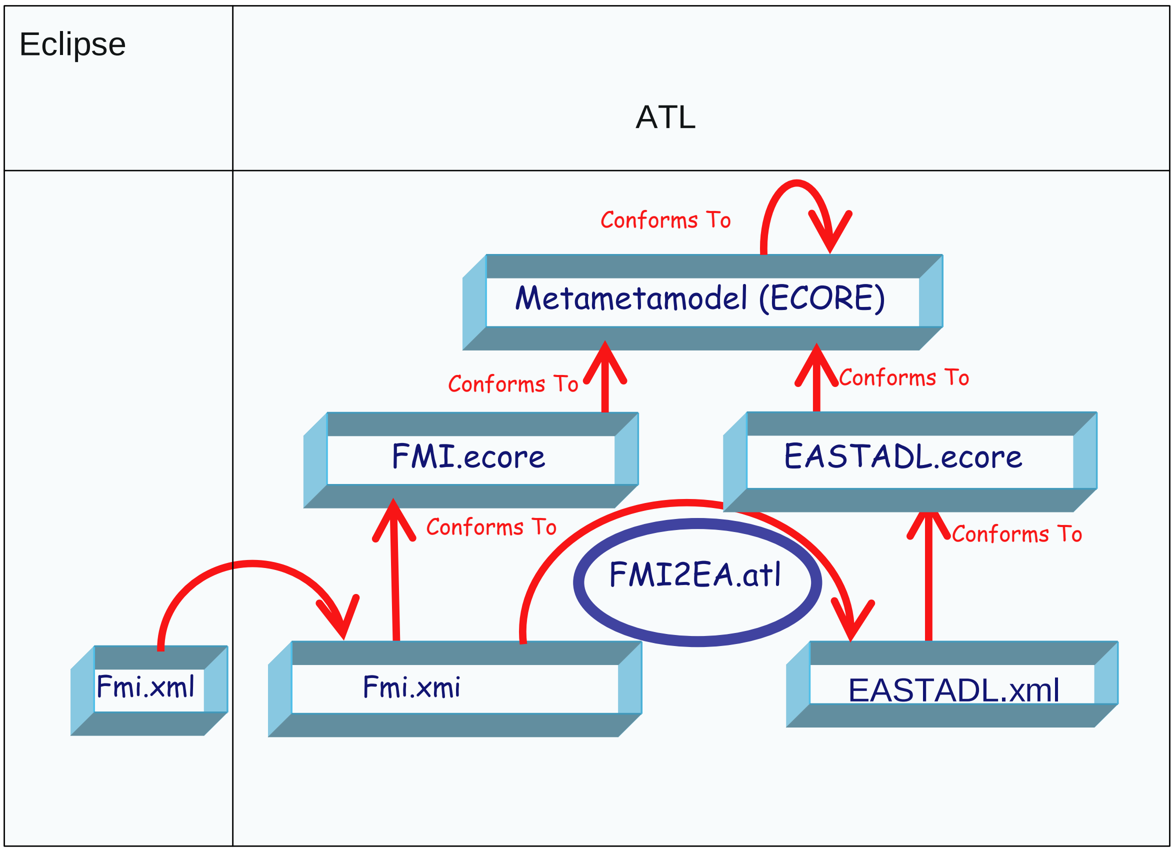


Figure 6.1.3/ First Planned System Data Flow of the Tool

After somehow taking the Fmi.Xml file with the help of the interface that will be designed, this file needs to be converted to another file format, XMI. This is because ATL works on XMI formatted files. This process should take place every time a conversion is done since in a Modelisar model, there is Fmi.xml file by default, so after every import of Fmi.xml there will be a conversion to Fmi.xmi. A study has been done for converting an XML file to an XMI in the context of Fmi.xml files and the steps for the conversion have been concluded. It was planned to do this simple transformation by Java. Moreover, ATL needs the ECORE MetaModels of the source and target models which are FMI.ecore and EASTADL.ecore respectively. These Metamodels can be obtained in two ways. One can manually create the Metamodel by writing the correspding Java Classes. Or, if there are XSD files available then EMF can be used to generate the Ecore MetaModels. In our case, XSD files for both FMI and EASTADL were available, so EMF was used to generate the FMI.ecore and EASTADL.ecore files. With these processes, nearly everything was ready for the first trial of the FMI.xmi to EASTADL.xml conversion. The only missing component was the ATL file that will supply the rules for the conversion. For this, 2 things had to be done. First step was to define some principle and basic mapping rules. Second was to implement these mapping rules in ATL language. There was already an idea on the correspondance between Modelisar and EAST-ADL from the start which was one of the important motivation points of the Thesis Topic. So this idea has been shaped to obtain the first version of the mapping that formed the main structure of the mapping. To implement the mapping in ATL, ATL language specifications have been learned in a basic level and then mapping rules have been implemented. Finally, a number of simple conversions have been run which gave satisfying results so ATL was kept as the transformation language.

6.1.4 Actual Implementation

With the first successful attempt of having a principle conversion, the time came for the actual, fully detailed version. For this, a first working plan that is similar on the figures 7 and 8 have been determined. The Figure 7 and 8 shows the actual sequence of the process which evolved over time and slightly differentiated from the first plan.

It was decided to use majorly an Iterative and Incremental Software Development Process and partly Waterfall Model where it is applicable. This approach was preferred as the project is experimental. How much the mapping will be possible, how suitable is ATL to implement all the mapping rules, what is Eclipse capable of and what structure and format should the model have, were all uncertain issues in the start. These topics get clearer all along the way which made an iterative process preferable. But in certain phases of the project Waterfall model has also been applied.

6.1.4.1 Main Phase

There was a need of a tool that can verify the generated EAST-ADL file by the conversion and edit it. There was no open source environment dedicated to edit EAST-ADL models in XML format. Papyrus is an EAST-ADL editor, but it is based on UML format which didn’t work in our case. So an editor that can edit EAST-ADL models in XML format became an interest. With this editor, the user would be able to generate models from scratch and then import the generated EAST-ADL models by the tool as a part of this model. Or the user can build larger models on top of the generated EAST-ADL models by opening them in the editor. While studying EMF before, an editor generation tool has been noticed. A more in-depth evaluation has been done on this specific tool. After spending some time on the tool, the editor in interest has been obtained. Thereafter, obtaining an editor that has the capability of editing AUTOSAR+EAST-ADL elements through the same model became an additional interest. After putting some extra effort, this editor has been obtained too.

Then the attention has been moved to one of the core topics of the project, obtaining the fully detailed mapping rules. This part of the work was more of an iterative process going back and forth between studying on EAST-ADL and MODELISAR in a very detailed manner and adding the new information gathered to the mapping rules. Also, working on the Papyrus and Eclipse Editor generated models supported the process for deciding on the final structure of the models obtained by the FMU2EA tool.

After some while, the near complete mapping rules have been obtained which carried the project to another core topic, implementing the mapping rules in ATL. For this, further knowledge about ATL was needed to be obtained. Although the documentation about this model transformation language was limited, there were supporting resources like Examples and forums. With using all of the information from these sources, an Apply&See approach has been followed. In this approach the ATL debug tool has helped very much to see the actual characteristics of the program.

With the deeper background obtained about ATL, the near complete mapping rules has been started to be implemented. The characteristics of ATL sometimes prevented the implementation to be straight forward and fast. During the implementation, the mapping rules have been refined too and found its final shape. These final rules have been completely applied in ATL but the models obtained by these rules had minor differences with the desired model structure and these were caused by the limitations of ATL. Fixing these differences is noted as future work.

For the final Analysis&Verification on the generated EAST-ADL.XML file’s structure, different paths have been taken. One of the tests has ended up with the conclusion that there is a need of further modification in the structure of the generated file. This test has been done with the VSA Volcano tool from Mentor Graphics AB. The tool has a feature of creating and editing EAST-ADL models. Furthermore it serializes the generated model in XML format unlike the previously used tools for verification. The reason this tool has not been used before in defining the structure of the EAST-ADL.XML file process is that it is not an open source tool and it became available at this final testing time to us. The generated files from this tool could be taken as reference since it was known that the tool completely conforms to the EAST-ADL Schema (XSD) during serialization of the created models which means that the structure of the obtained EAST-ADL.XML file will be in the desired format. After doing a number of tests, the differences between the generated EAST-ADL model's XML files from ATL and VSA have been found and the needed modifications to be done to the ATL generated XMLs in order to obtain VSA like files have been listed.

The modified conversion process can be seen in Figure 6.1.4.1 more in depth. This figure is very similar to Figure 6.1.3 with 2 differences. An additional step has been added to the end which can be seen in the figure as a dashed arrow starting from EASTADL.XML and ending at EASTADL.EAXML. The EASTADL.EAXML file is the desired final format as discussed above. The other difference is that in Figure 6.1.4.1 the arrows from FMI.XML to FMI.XMI and from EASTADL.XML to EASTADL.EAXML are dashed. This means that the process taken for these conversions will be handled manually. In the start the plan was to make these extra processes with Java but it has been decided to make them manually to leave more time on the main frame since it was known that the implementation of these extra processes is no big deal and can be turned back later on. So this modification has been noted as a Future Work.

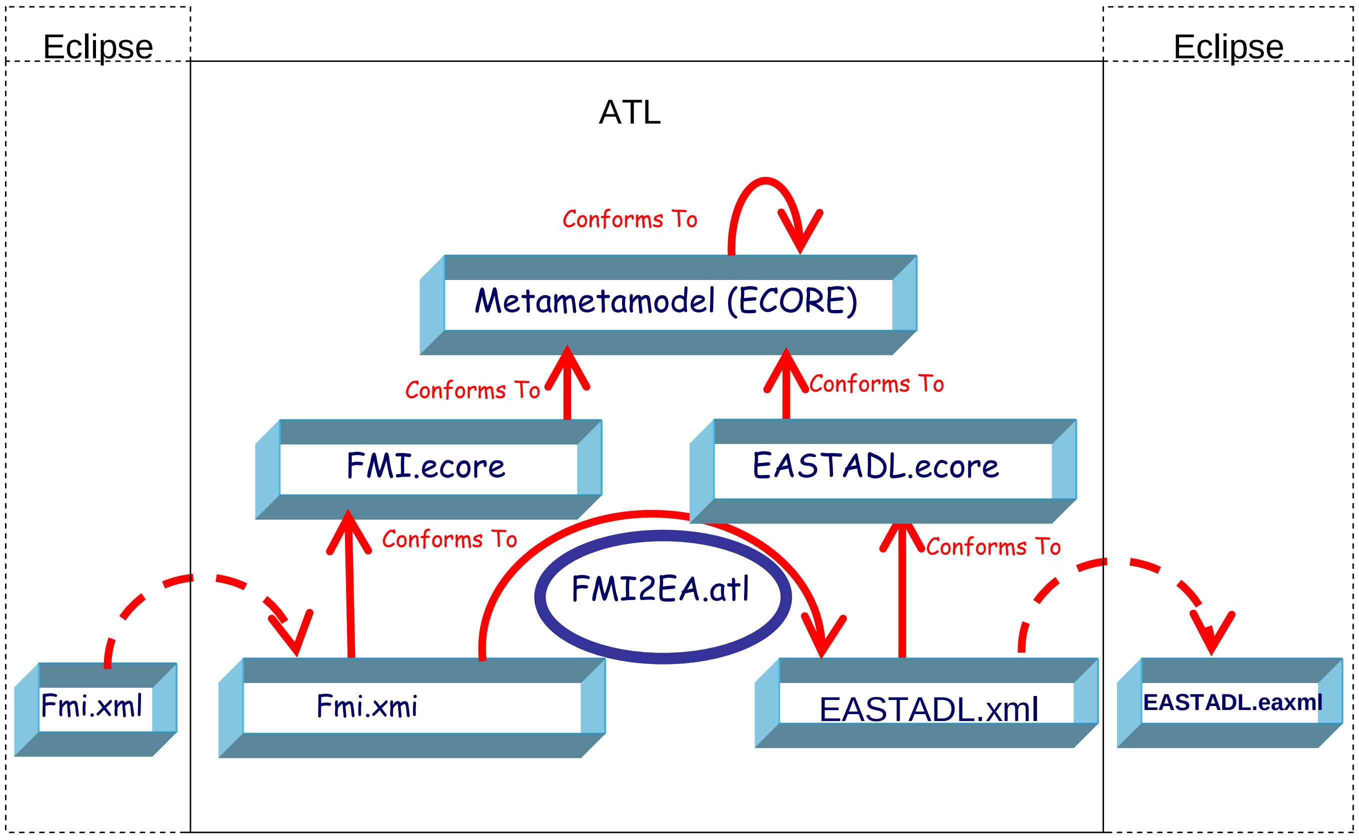


Figure 6.1.4.1/Modified System Data Flow of the Tool

6.1.4.2 Finalization

Now, the time has come to integrate the core with some kind of interface. Although an overview of the interface has been formed in the beginning as a first idea, further discussion went on. An Eclipse plug-in that pop-ups with the right click of the mouse in a Modelisar or an EAST-ADL model has been proposed as the most convenient solution. This plug-in was planned to have the option 'Convert to East Adl' when clicked to a Modelisar object and the option 'import from' when clicked to an EAST-ADL model. If a Modelisar model was selected to be converted with the plug-in, then a new EAST-ADL model would appear in the workspace. If an EAST-ADL model was selected with the plug-in to make a conversion, then a browsing menu would appear to let the user to select the desired Modelisar model to be converted, and after that the EAST-ADL model resulted from that Modelisar model would be outputted to the selected EAST-ADL model in the start.

A general research has been done about the topic and it has been found that Eclipse offers a tool to generate a plug-in for ATL applications. For saving time and having a more generic application, using this tool for the implementation of the planned plug-in has been decided. A deeper work on the tool has been made and afterwards the tool is used to generate the Java source files of the plug-in. Some modifications have been done on these source files in order to make the plug-in work. This plug-in was used by running the the main method of its source files. This was not a desirable result since it would not be neat and straightforward from the point of view of the user. Also it was not good to let the user to access the source code. To avoid the listed circumstances, forming an executable for the obtained plug-in has been found as the convenient solution. Eclipse also supports an easy way of obtaining the so called executable. With using this tool a .jar file is obtained which only needs a running JVM on the machine to operate. This executable runs with the help of command window. The location of the Modelisar model which the conversion will take place on, the desired name of the output model, the desired extension of the output model and the desired location of the output file are needed as inputs. The users enter these inputs from the command prompt. After the executable takes the inputs and the run command, it outputs the result in the specified location with the specified extension and name.

Although the plan of having a plug-in dedicated to Eclipse evolved over time into something else, it resulted in a more satisfying result since the tool in the end became platform independent. But, the original plan is added to the future work list as it could be useful in a possible tool integration process with KTH's tools since the plug-in planned originally is very similar to the KTH's plug-in applications.

After all, several tests have been done on the tool. These tests ended up successfully and made the verification. Then, it has been decided to define and list relevant future improvements that could be useful in the possible future uses and applications of the tool. Also making some preliminary work on some of these has been found convenient to do. Finally, after all of the process that has been gone through, the detailed final documentation of the tool has been completed, which was also the end of the thesis project.

6.2 Final System Overview

6.2.1 FMU2EA Tool

With the evolution of the system in time, the final system overview looks as in Figure 6.2.1:

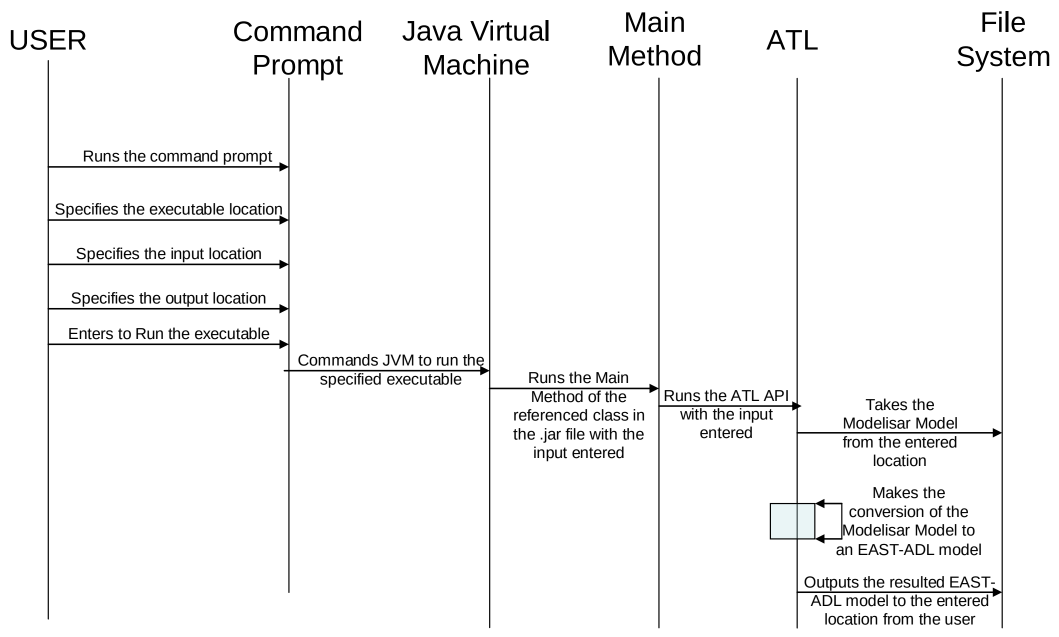

USER
Command Prompt
Java Virtual Machine
Main Method
ATL
File System
Runs the command prompt
Specifies the executable location
Specifies the input location
Specifies the output location
Enters to Run the executable
Commands JVM to run the specified executable
Runs the Main Method of the referenced class in the .jar file with the input entered
Runs the ATL API with the input entered
Takes the Modelisar Model from the entered location
Makes the conversion of the Modelisar Model to an EAST-ADL model
Outputs the resulted EAST-ADL model to the entered location from the user

Figure 6.2.1/ Final System Overview

User starts with opening the command prompt. He specifies the .jar file's location. Then he enters the Modelisar model's location for the input and another desired location for the output and runs the executable. Then JVM is instantiated. JVM runs the main method that is pointed in the .jar file as the starting point. In our case, the main method takes the inputs that is entered by the user in the command window and communicates with ATL API to run the ATL launch configurations. When the ATL launching configurations has been done, the corresponding Modelisar model found from the entered input path is taken from the file system by the ATL VM. After that ATL VM executes the .atl file that includes the mapping rules for creating the corresponding EAST-ADL model of the Modelisar model. Finally, the generated model is put into the previously specified place in the file system.

It should be noted that the input Modelisar model needs a small modification before it is processed by ATL. For the current version of the tool, this modification will be done manually by the user before running the executable. As stated before, the extra process that will be gone through for obtaining the final structure of the EAST-ADL.XML file will be done manually too.

6.2.2 EAST-ADL Editor

User runs the Java source files of the Editor as an Eclipse Application. After that, another instance of Eclipse opens. This new thread of Eclipse recognizes the Editor as a plug-in. If one can see the generated Editor in the plug-in details menu of the new instance of Eclipse then it means that the Editor is generated correctly and is ready to be used. Now the user can create an empty EAST-ADL model and edit it. The Editor directs the user when building up a model. Also it supports a validation feature which checks the model formed and gives errors if there are differences between the model built and the Schema (XSD). When the model is saved, it is serialized into an XML file. An editor is obtained both for FMI and EAST-ADL. The EAST-ADL Editor is capable of building AUTOSAR and EAST-ADL models concurrently.

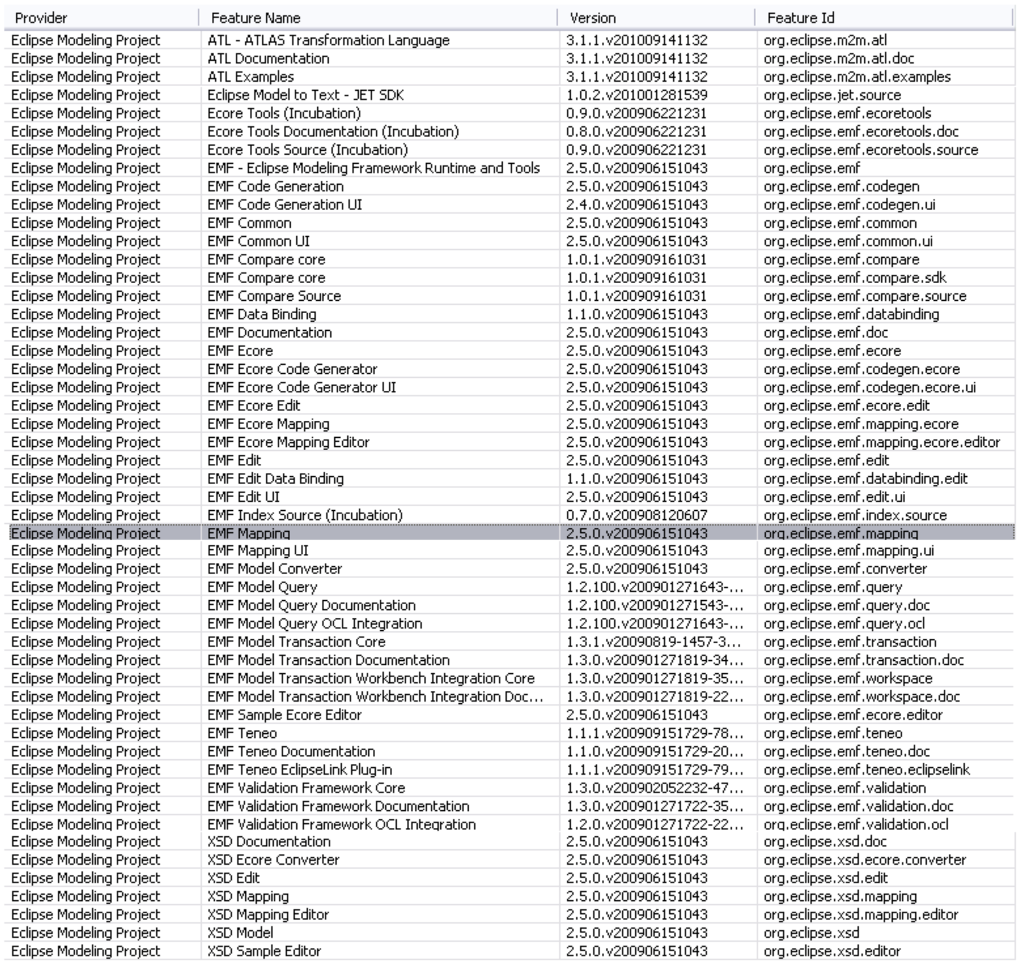
6.3 Implementation Details

In this section an in depth explanation of all the details about the building blocks of the FMU2EA tool and the EAST-ADL editor will take place. A general explanation about the below topics has taken place in the problem approach part and a system overview has been given in the previous section. For understanding more clearly about how these building blocks make up the the tool or the editor, please refer to the previous sections.

Some of the below components are a part of both the Editor and the FMU2EA tool.

6.3.1 Setting up Eclipse

Different tools of Eclipse have been used for the development of the FMU2EA tool and the EAST-ADL Editor. These tools are provided by EMF, MDT, M2M and XSD projects. They also support the infrastructure for the development. All the tools of these projects can be installed to any Eclipse platform. Also there are Eclipse releases which include the listed projects as a pack. As it is better to have a standard platform, it is chosen to use one of these Modelling Tools releases. Galileo, Helios and Ganymede are the latest Eclipse releases that also have the Modelling Tools releases. These three have been published in different times so using the oldest one has the benefit of being a stable release. In the other hand the newer ones has the benefit of supporting the newest editions which sometimes support new tools. For obtaining the best performance, all of the 3 releases have been tested. Finally, Galileo release has been chosen for the tool development and Helios release has been chosen for the Editor generation. For the Galileo release, some updates have been done to some of the features and the final versions of them can be seen in Figure 6.3.1.



Provider	Feature Name	Version	Feature Id
Eclipse Modeling Project	ATL - ATLAS Transformation Language	3.1.1.v201009141132	org.eclipse.m2m.atl
Eclipse Modeling Project	ATL Documentation	3.1.1.v201009141132	org.eclipse.m2m.atl.doc
Eclipse Modeling Project	ATL Examples	3.1.1.v201009141132	org.eclipse.m2m.atl.examples
Eclipse Modeling Project	Eclipse Model to Text - JET SDK	1.0.2.v201001281539	org.eclipse.jet.source
Eclipse Modeling Project	Ecore Tools (Incubation)	0.9.0.v200906221231	org.eclipse.emf.ecoretools
Eclipse Modeling Project	Ecore Tools Documentation (Incubation)	0.8.0.v200906221231	org.eclipse.emf.ecoretools.doc
Eclipse Modeling Project	Ecore Tools Source (Incubation)	0.9.0.v200906221231	org.eclipse.emf.ecoretools.source
Eclipse Modeling Project	EMF - Eclipse Modeling Framework Runtime and Tools	2.5.0.v200906151043	org.eclipse.emf
Eclipse Modeling Project	EMF Code Generation	2.5.0.v200906151043	org.eclipse.emf.codegen
Eclipse Modeling Project	EMF Code Generation UI	2.4.0.v200906151043	org.eclipse.emf.codegen.ui
Eclipse Modeling Project	EMF Common	2.5.0.v200906151043	org.eclipse.emf.common
Eclipse Modeling Project	EMF Common UI	2.5.0.v200906151043	org.eclipse.emf.common.ui
Eclipse Modeling Project	EMF Compare core	1.0.1.v200909161031	org.eclipse.emf.compare
Eclipse Modeling Project	EMF Compare core	1.0.1.v200909161031	org.eclipse.emf.compare.sdk
Eclipse Modeling Project	EMF Compare Source	1.0.1.v200909161031	org.eclipse.emf.compare.source
Eclipse Modeling Project	EMF Data Binding	1.1.0.v200906151043	org.eclipse.emf.databinding
Eclipse Modeling Project	EMF Documentation	2.5.0.v200906151043	org.eclipse.emf.doc
Eclipse Modeling Project	EMF Ecore	2.5.0.v200906151043	org.eclipse.emf.ecore
Eclipse Modeling Project	EMF Ecore Code Generator	2.5.0.v200906151043	org.eclipse.emf.codegen.ecore
Eclipse Modeling Project	EMF Ecore Code Generator UI	2.5.0.v200906151043	org.eclipse.emf.codegen.ecore.ui
Eclipse Modeling Project	EMF Ecore Edit	2.5.0.v200906151043	org.eclipse.emf.ecore.edit
Eclipse Modeling Project	EMF Ecore Mapping	2.5.0.v200906151043	org.eclipse.emf.mapping.ecore
Eclipse Modeling Project	EMF Ecore Mapping Editor	2.5.0.v200906151043	org.eclipse.emf.mapping.ecore.editor
Eclipse Modeling Project	EMF Edit	2.5.0.v200906151043	org.eclipse.emf.edit
Eclipse Modeling Project	EMF Edit Data Binding	1.1.0.v200906151043	org.eclipse.emf.databinding.edit
Eclipse Modeling Project	EMF Edit UI	2.5.0.v200906151043	org.eclipse.emf.edit.ui
Eclipse Modeling Project	EMF Index Source (Incubation)	0.7.0.v200908120607	org.eclipse.emf.index.source
Eclipse Modeling Project	EMF Mapping	2.5.0.v200906151043	org.eclipse.emf.mapping
Eclipse Modeling Project	EMF Mapping UI	2.5.0.v200906151043	org.eclipse.emf.mapping.ui
Eclipse Modeling Project	EMF Model Converter	2.5.0.v200906151043	org.eclipse.emf.converter
Eclipse Modeling Project	EMF Model Query	1.2.100.v200901271643-...	org.eclipse.emf.query
Eclipse Modeling Project	EMF Model Query Documentation	1.2.100.v200901271543-...	org.eclipse.emf.query.doc
Eclipse Modeling Project	EMF Model Query OCL Integration	1.2.100.v200901271643-...	org.eclipse.emf.query.ocl
Eclipse Modeling Project	EMF Model Transaction Core	1.3.1.v20090819-1457-3...	org.eclipse.emf.transaction
Eclipse Modeling Project	EMF Model Transaction Documentation	1.3.0.v200901271819-34...	org.eclipse.emf.transaction.doc
Eclipse Modeling Project	EMF Model Transaction Workbench Integration Core	1.3.0.v200901271819-35...	org.eclipse.emf.workspace
Eclipse Modeling Project	EMF Model Transaction Workbench Integration Doc...	1.3.0.v200901271819-22...	org.eclipse.emf.workspace.doc
Eclipse Modeling Project	EMF Sample Ecore Editor	2.5.0.v200906151043	org.eclipse.emf.ecore.editor
Eclipse Modeling Project	EMF Teneo	1.1.1.v200909151729-78...	org.eclipse.emf.teneo
Eclipse Modeling Project	EMF Teneo Documentation	1.1.0.v200909151729-20...	org.eclipse.emf.teneo.doc
Eclipse Modeling Project	EMF Teneo EclipseLink Plug-in	1.1.1.v200909151729-79...	org.eclipse.emf.teneo.eclipselink
Eclipse Modeling Project	EMF Validation Framework Core	1.3.0.v200902052232-47...	org.eclipse.emf.validation
Eclipse Modeling Project	EMF Validation Framework Documentation	1.3.0.v200901271722-35...	org.eclipse.emf.validation.doc
Eclipse Modeling Project	EMF Validation Framework OCL Integration	1.2.0.v200901271722-22...	org.eclipse.emf.validation.ocl
Eclipse Modeling Project	XSD Documentation	2.5.0.v200906151043	org.eclipse.xsd.doc
Eclipse Modeling Project	XSD Ecore Converter	2.5.0.v200906151043	org.eclipse.xsd.ecore.converter
Eclipse Modeling Project	XSD Edit	2.5.0.v200906151043	org.eclipse.xsd.edit
Eclipse Modeling Project	XSD Mapping	2.5.0.v200906151043	org.eclipse.xsd.mapping
Eclipse Modeling Project	XSD Mapping Editor	2.5.0.v200906151043	org.eclipse.xsd.mapping.editor
Eclipse Modeling Project	XSD Model	2.5.0.v200906151043	org.eclipse.xsd
Eclipse Modeling Project	XSD Sample Editor	2.5.0.v200906151043	org.eclipse.xsd.editor

Figure 6.3.1/ Version information of the plug-ins of the Galileo platform which is used as the development environment of the FMU2EA tool

6.3.2 Obtaining the MetaModels

Metamodel of the FMI and EAST-ADL were needed for developing and using ATL and EMF editor generator. Moreover this metamodel should be an Ecore Metamodel which means that it has to follow the Ecore MetaMetamodel which can be seen from the top-view in Figure 6.3.2.1. ‘Following’ here means, creating a metamodel that is in the structure of Ecore MetaMetaModel.

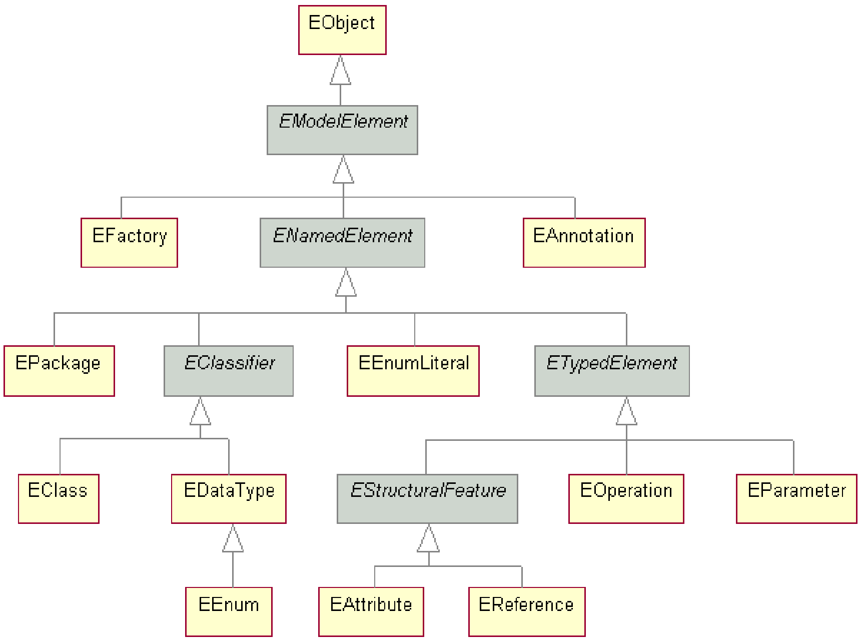
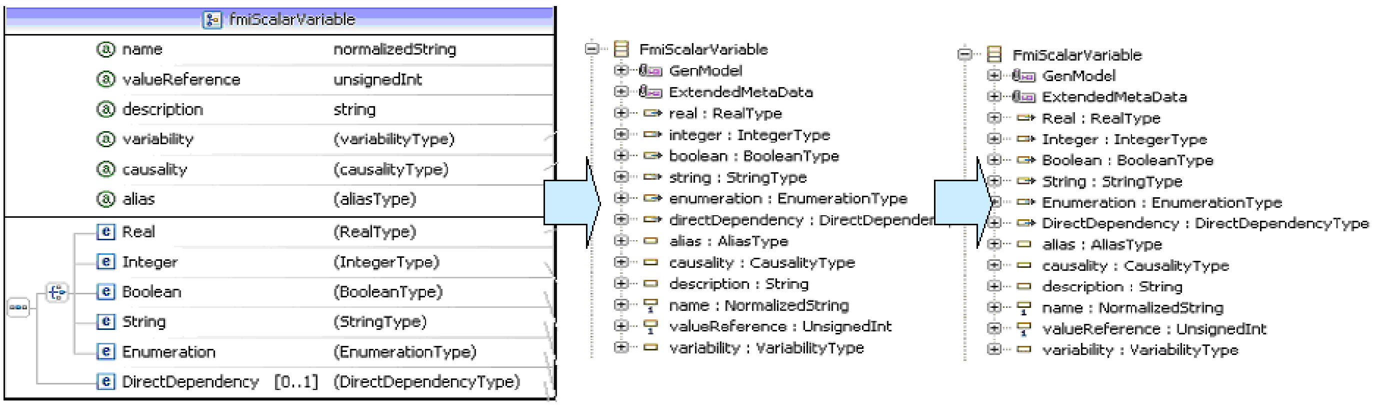


Figure 6.3.2.1 / Ecore MetaMetaModel

There are 2 ways to form a Metamodel that conforms to this ECore MetaMetaModel. First is to create it manually from scratch. Second is to use EMF to generate one. EMF has a built in tool for generating Ecore Metamodels from XSD files. The second one has been chosen since XSD files of FMI and EAST-ADL were available. The first way is a huge process and can be a thesis project itself.

Applying certain steps, the MetaModels were created (These steps can be found in the **Appendix C1**). But the Metamodels resulted from the generation were not in the exact format that is desired. They had small differences. A small part of the XSD and the generated MetaModel is shown in Figure 6.3.2.2. As can be seen in the XSD, all the elements (Real, Integer, etc.) starts with upper-case letters where as the generated MetaModel have them as lower-case. As the attributes starts with lower-case letters in XSD, this has caused no difference in the MM.



fmiScalarVariable

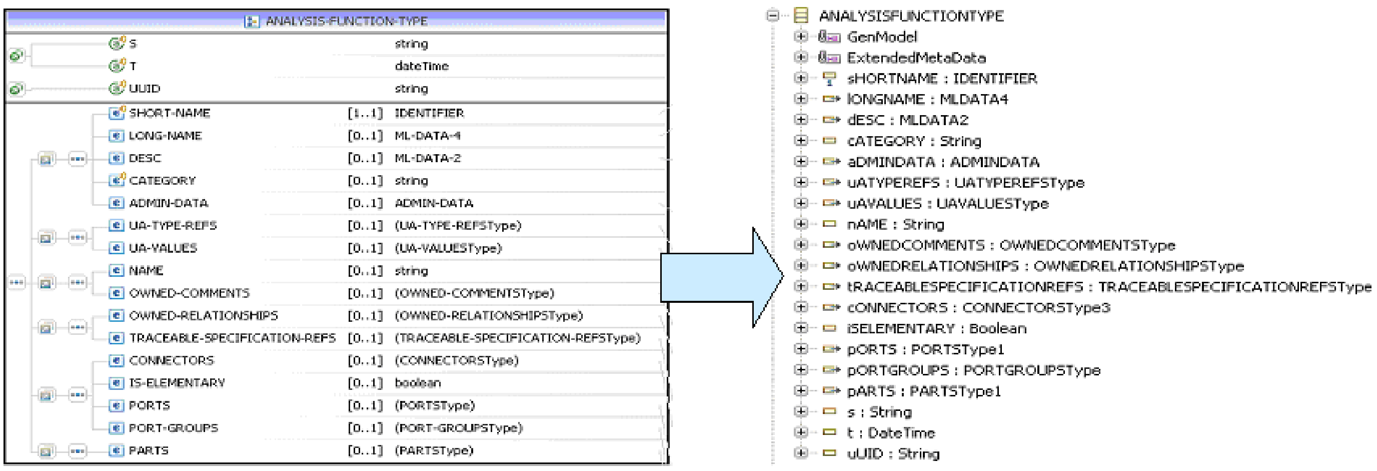
name	normalizedString
valueReference	unsignedInt
description	string
variability	(variabilityType)
causality	(causalityType)
alias	(aliasType)
Real	(RealType)
Integer	(IntegerType)
Boolean	(BooleanType)
String	(StringType)
Enumeration	(EnumerationType)
DirectDependency [0..1]	(DirectDependencyType)

```
FmiScalarVariable
    GenModel
    ExtendedMetaData
    real : RealType
    integer : IntegerType
    boolean : BooleanType
    string : StringType
    enumeration : EnumerationType
    directDependency : DirectDepender
    alias : AliasType
    causality : CausalityType
    description : String
    name : NormalizedString
    valueReference : UnsignedInt
    variability : VariabilityType
```

```
FmiScalarVariable
    GenModel
    ExtendedMetaData
    Real : RealType
    Integer : IntegerType
    Boolean : BooleanType
    String : StringType
    Enumeration : EnumerationType
    DirectDependency : DirectDependencyType
    alias : AliasType
    causality : CausalityType
    description : String
    name : NormalizedString
    valueReference : UnsignedInt
    variability : VariabilityType
```

Figure 6.3.2.2 / FMI XSD and generated FMI MM and the modified FMI MM

This situation is similar in the EAST-ADL case with additional differences. In EAST-ADL XSD, all the elements and attributes start with uppercase letter. Additionally, every different word in the name of an element or attribute is separated by dash characters(-). EMF removes these characters and changes the first uppercase letters with lower ones. This can be seen in Figure 6.3.2.3. In the left figure ANALYSIS-FUNCTION-TYPE elements' attributes and other elements inside it is shown as it is in the EAST-ADL XSD. The right figure shows the same element in the EAST-ADL MetaModel that is generated by EMF. As can be seen, all the element and attribute names has gone through a change. For instance SHORT-NAME became sHORTNAME in the MetaModel.



```
ANALYSIS-FUNCTION-TYPE
S                                          string
T                                          dateTime
UUID                                       string
SHORT-NAME                        [1..1]   IDENTIFIER
LONG-NAME                         [0..1]   ML-DATA-4
DESC                              [0..1]   ML-DATA-2
CATEGORY                          [0..1]   string
ADMIN-DATA                        [0..1]   ADMIN-DATA
UA-TYPE-REFS                      [0..1]   (UA-TYPE-REFSType)
UA-VALUES                         [0..1]   (UA-VALUESType)
NAME                              [0..1]   string
OWNED-COMMENTS                    [0..1]   (OWNED-COMMENTSType)
OWNED-RELATIONSHIPS               [0..1]   (OWNED-RELATIONSHIPSType)
TRACEABLE-SPECIFICATION-REFS      [0..1]   (TRACEABLE-SPECIFICATION-REFSType)
CONNECTORS                        [0..1]   (CONNECTORSType)
IS-ELEMENTARY                     [0..1]   boolean
PORTS                             [0..1]   (PORTSType)
PORT-GROUPS                       [0..1]   (PORT-GROUPSType)
PARTS                             [0..1]   (PARTSType)
```

```
ANALYSISFUNCTIONTYPE
  GenModel
  ExtendedMetaData
  sHORTNAME : IDENTIFIER
  lONGNAME : MLDATA4
  dESC : MLDATA2
  cATEGORY : String
  aDMINDATA : ADMINDATA
  uATYPEREFS : UATYPEREFSType
  uAVALUES : UAVALUESType
  nAME : String
  oWNEDCOMMENTS : OWNEDCOMMENTSType
  oWNEDRELATIONSHIPS : OWNEDRELATIONSHIPSType
  tRACEABLESPECIFICATIONREFS : TRACEABLESPECIFICATIONREFSType
  cONNECTORS : CONNECTORSType3
  iSELEMENTARY : Boolean
  pORTS : PORTSType1
  pORTGROUPS : PORTGROUPSType
  pARTS : PARTSType1
  s : String
  t : DateTime
  uUID : String
```

Figure 6.3.2.3 / AnalysisFunctionType(AFT) XSD and the generated AFT MM

This rather small and partial difference in the obtained FMI MM causes EMF not recognizing the Modelisar models since the MM is used to open the models. This means that ATL will not be able to interpret the model directly as it is. There are 2 solutions to this problem. Either changing the MM manually in the desired way, or adding an additional process of changing the Models in the way that they will conform to the original generated MM. The case in EAST-ADL is similar too. The output that is obtained after the conversion would have slight differences than the desired one if the generated MM for EAST-ADL is used directly. So either the MM for EAST-ADL can be changed manually or an extra step can be added to the conversion before the output takes its final shape.

Changing the MetaModel manually is all right for the FMI MM but it is not feasible for the EASTADL MM since EAST-ADL MM is much bigger than FMI MM which would take much more time to modify. Also the chances of modifying the MetaModel for the EAST-ADL one in an incomplete manner is much higher. Modifying both of the MetaModels is nicer from the tool perspective as having a compatible MM with the model avoids an additional process on the model to make it look like in the MM. But still, this is not a good solution in the sense that the tool loses a part of its generic character since the MetaModel used becomes a specific one when it is modified. Also in the case of keeping the original MMs and adding an extra process on the models, if the XSD goes through a change, then it would be possible to generate the modified MM from EMF directly but otherwise either the changes should be applied to the MM itself or if the XSD is used to generate a new MM, all of these manual changes should take place again on the MM. But some changes on the XSD even can affect the ATL conversion and a refinement can be needed in the rules depending on the case. So modifying the MM maybe can not be seen as a big deal as other modifications can as well be needed in the case of a change of XSD.

As a result, FMI MM is changed manually for this version of the application but EAST-ADL MM is kept as it is generated from EMF. So as a result the extra process needed for the Modelisar models are avoided. As the original generated EAST-ADL MM is used in the system, an additional process is needed after the output is formed. This process is explained in the "Converting the output to EAXML format" section.

6.3.3 Obtaining the Editor

During the generation of the MetaModels, an EMF model is created too. Eclipse has the feature to generate an Editor for this EMF model. When this feature of EMF is used to generate an Editor, Java classes are formed by EMF. More or less everything is done by EMF. But to obtain the desired Editor, there are options to change. The options used for generating the Editor for EAST-ADL is shown in **Appendix C2**. The Editor created for Fmi uses the default options. Another important parameter is to use the appropriate release of Eclipse with the correct versions of the features as they are linked to each other. The platform that has been used for generating and using the Editors is Eclipse Helios as said before. The versions of the EMF features are shown in **Appendix C2**.

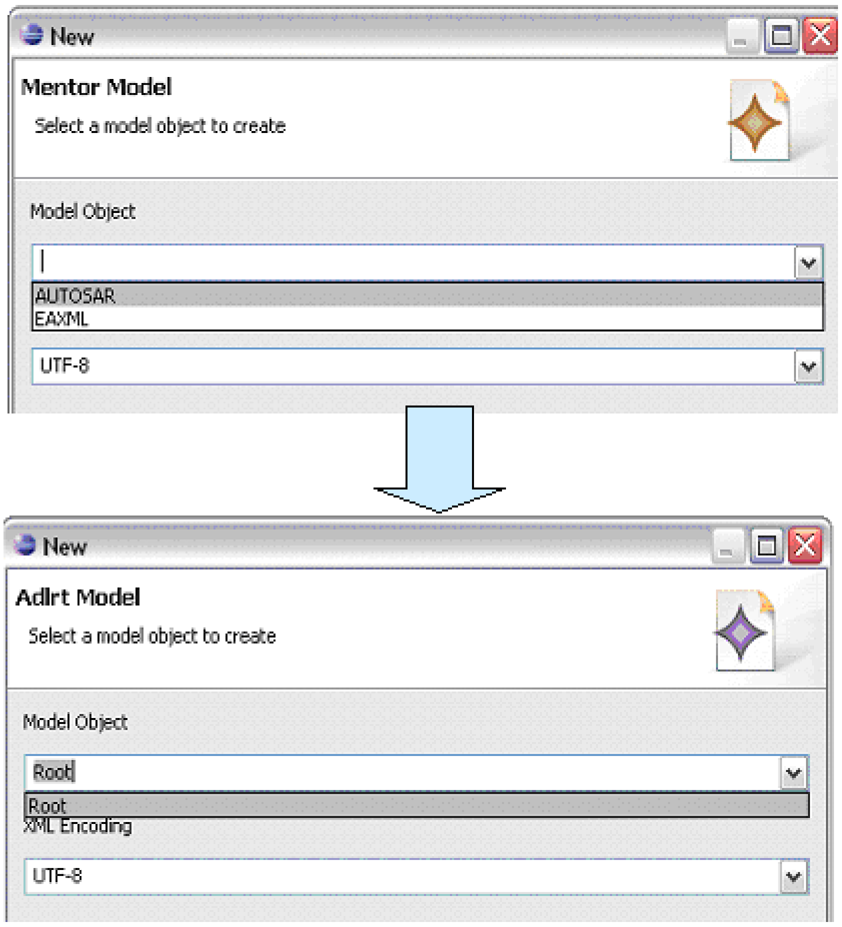


Figure 6.3.3/ Root Elements of the different Models obtained from the original and the modified XSD

An EAST-ADL and an Fmi Editor is obtained which meets the expectations but an Editor which gives the chance to the user to build a model that constitutes of both AUTOSAR and EAST-ADL elements. But the constructed Editor limited the user to choose either AUTOSAR or EAST-ADL models to build, so the user was not able to build these models concurrently. This case can be seen from the top figure in Figure 6.3.3.1. This limitation was also a little bit strange for us in the start because an XSD that constitutes of both AUTOSAR and EAST-ADL elements is used for the generation of the Editor. Then we had an idea why the editor gives 2 choices when creating the model. To test the idea, a small modification has been done to the schema and an Editor is obtained with it. This editor meets the requirements. With this editor it is possible to add AUTOSAR and EAST-ADL elements in the same model. In this new Editor there is no choice of AUTOSAR or EAXML when creating the model. This can be seen from Figure 6.3.1.1.

For obtaining the modified XSD which the editor uses, a Root element is created. The AUTOSAR and EAXML elements which are the top most and root elements of AUTOSAR and EAST-ADL are put inside the ‘Root’ element. The structure of the root element that is added to the XSD can be seen below:

```
<xsd:element name="Root">

        <xsd:complexType>

          <xsd:choice minOccurs="0" maxOccurs="1">

            <xsd:element name="AUTOSAR" type="AR:AUTOSAR"/>

            <xsd:element name="EAXML" type="AR:EAXML"/>

          </xsd:choice>

        </xsd:complexType>

      </xsd:element>
```

This makes the model generated with this editor to have only 1 root instead of 2. So this was why the editor was giving 2 options to form a model. But this is a non standard solution which does not conform to the Meta-Model of neither AUTOSAR nor EAST-ADL since an additional element is created. This case can cause incompatibilities with the other tools to open the models that are created with this Editor. But the solution is rather easy. The user has to do some steps manually to obtain a compatible format of the model. These steps can be found in “Converting the output to EAXML format” section.

The generated Editor runs as an Eclipse application in another instance of Eclipse but this can be modified. With some additional steps the generated Editor can be added to the core plug-ins. This implementation hasn’t been done in this version of the application and is put to the Future Work list.

Also an Editor is obtained for FMI with the modified MM which gives the user the ability of creating FMI’s from scratch or modifying the existing ones. This also adds the tool additional strength as the user can play with the Modelisar models in hand to obtain different EAST-ADL models even if they don’t have any Editor other than the generated one. Since the models can be exchanged through people or found in the network, it is not guaranteed that a user will have a Modelisar Editor.

6.3.4 Obtaining the Mapping Rules

This section will give you an insight about the idea behind the conversion of Modelisar models to EAST-ADL ones. For further details, refer to **Appendix A3**.

The main notion behind the conversion is the mapping between an FMU (Functional Mockup Unit) and an EAST-ADL model. The overall look to the mapping is that an FMU corresponds to a FunctionType element in EAST-ADL. The structure of the Function Type and the connected elements can be seen from the relevant parts of the EAST-ADL MetaModel which can be found in **Appendix A2**. Also the structure and components of FMI can be found in **Appendix A1**. As a general picture, a Function Type constitutes of input, output and input/output ports. Ports are the only way of connection with FunctionTypes. For instance, ports are used as the gateway for exchanging information between different Function Types. It can also have function connectors and port groups but they are out of interest for this version of the conversion. FunctionTypes and Ports have different types. A Function Type can be an AnalysisFunctionType or a DesignFunctionType. A Port can be a FunctionFlowPort, FunctionPowerPort or a FunctionClientServerPort. An FMU consists of an fmiModelDescirption element which is the root. It contains FmiType and FmiScalarVariable elements which are the important ones among others. Mainly, fmiModelDescription corresponds to a FunctionType where FmiType and FmiScalarVariable are used to form the ports.

In theory, there is no limitation for the mapping. An FMU can be mapped to an AnalysisFunctionType or a DesignFunctionType with FuncitonFlowPorts, FunctionPowerPorts and

FunctionClientServerPorts. But in practice there is no way to guess which FuntionType or Port will be used. So it is always assumed that an FMU corresponds to an AnalysisFunctionType with FunctionFlowPorts. In reality, it can also be mapped to a DesignFunctionType, but it is thought that there is no way or it will be too costly in the sense of time to predict whether an FMU is an AnalysisFunctionType or a DesignFunctionType, so it will be assumed that all FMUs are AnalysisFunctionTypes. But this is not an important issue. The user can always modify the EAST-ADL model in the way that it should be afterwards the conversion. This modification is a straightforward process. The user has to only change the type of the FunctionType or Port from an EAST-ADL editor. None of the information of the model would be affected by this change.

The FunctionFlowPorts are typed by Datatypes. In other words, every Port has a corresponding DataType. DataType of the Port determines the unit of the information exchange with the other Ports that it is connected. This DataType could be an EABoolean, EAString, Enumeration, EAFloat or an EAInteger and Enumeration can hold several EnumerationLiterals. The DataType could also be a composite data type but there is no correspondence between the elements of the FMU and composite data type. So an EAST-ADL model that is generated from an FMU can not have a port whose data type is composite data type. The Port's DataType type is decided depending on the information in the FMU, more specifically from FmiTypes and FmiScalarVariables of the FMU.

There are 2 more elements that are used in the mapping, RangeableValueType and EnumerationValueType. RangeableValueType gives reference to either an EAFLOAT or an EAINTGER. EnumerationValueType gives reference to an Enumeration. These 2 holds extra information about the DataTypes that they give reference. The information they hold is obtained from the corresponding FmiTypes and FmiScalarVariables of the FMU as well.

To sum up, FMU has all the relevant information for forming an AnalysisFunctionType with its corresponding elements. One thing that should be noted here is that FMUs has no input/output elements. They only have elements that have the input or output characteristic. This can be seen as a limitation from the first look but from the point of view of converting an FMU to an EASTADL, there is no harm since no incorrect or incomplete mapping is done. But if a conversion in the opposite way is desired to be done, then this situation can become an issue. In that case, instead of creating one element that has an input/output character, creating 2 elements which one of them is an input and the other is output can be a solution. Furhter details of the mapping can be found in the **Appendix A3**.

6.3.5 XMLtoXMI

As discussed before, the FMI of the Modelisar Model is an XML file in standard. But EMF and ATL works with XMI formatted files. So as FMI is the input of the ATL conversion, the FMI.XML file should be converted to an FMI.XMI file. This conversion is very simple and straightforward. Only the header of the document is replaced by another header and the structure of the rest of the file is left the same. The reason behind this modification from XML to XMI is that EMF and ATL looks for the ECORE model to which the model in the file conforms. So the starting line of the XML documents is changed as below to convert it to an XMI:

XML:

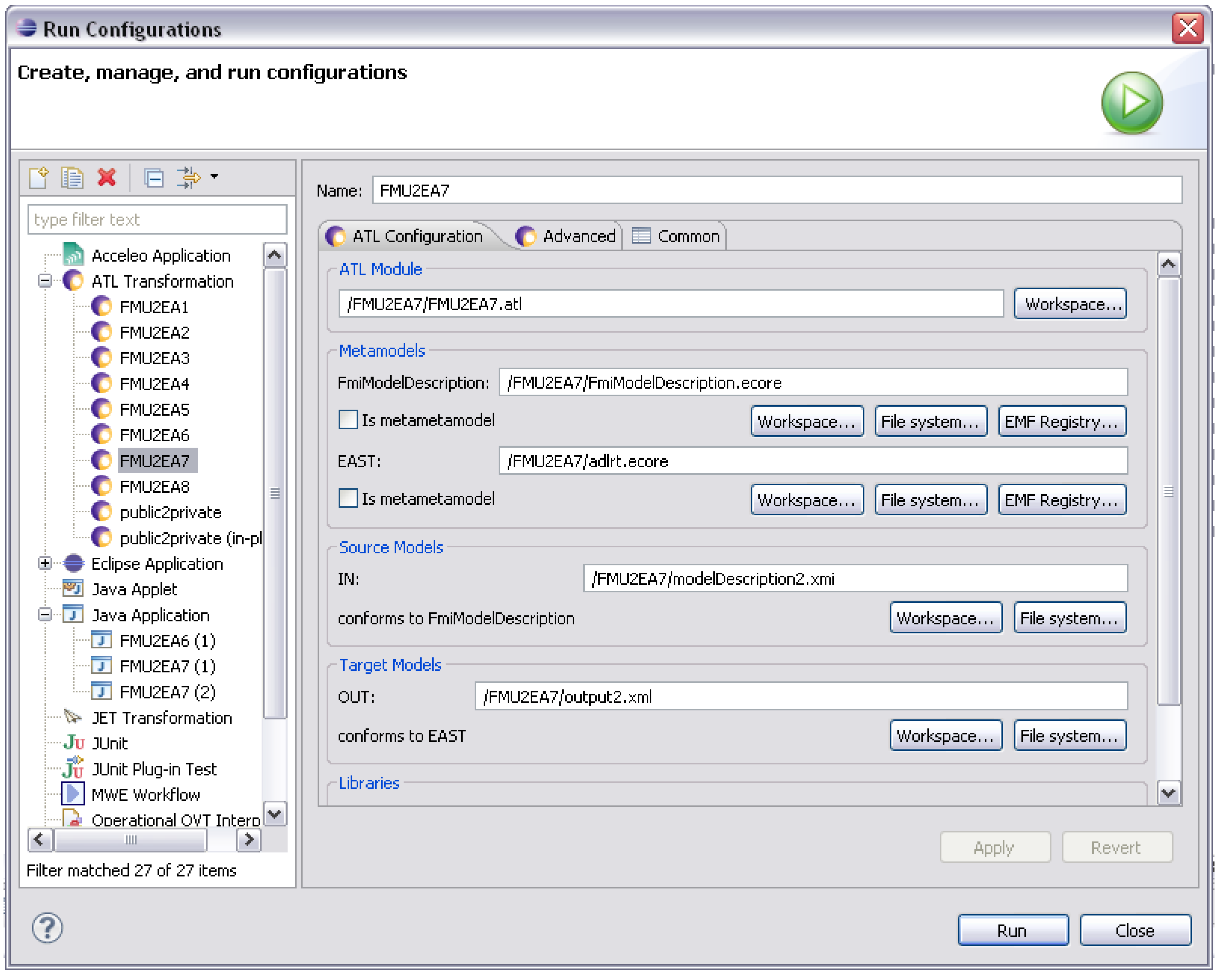
```
<?xml version="1.0" encoding="UTF-8"?>
```

XMI:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns="platform:/resource/FMIEX1/MetaModels/FmiModelDescription.ecore">
```

6.3.6 Setting up ATL

ATL has a launch configuration menu which shows up before running an ATL conversion. The user puts the necessary information in this menu to run the conversion. This menu can be seen in Figure 6.3.6. It needs the locations of the MetaModels of the input and output models. It is also necessary to point where the input model is located and where the output model should be put. There are also other advanced options which have to be adjusted for the application. This configuration has been done once and later on, it is embedded to the tool during the Plug-In implementation of the ATL conversion. So in the final version of the tool which runs as an executable, when the user enters the input that is needed for the tool, this input is taken to configure the configurations of ATL. The rest of the adjustments done in the configuration other than the ones that have been taken as input from the user are preserved in the tool.



Name: FMU2EA7

ATL Module

/FMU2EA7/FMU2EA7.atl

Metamodels

FmiModelDescription: /FMU2EA7/FmiModelDescription.ecore

☐ Is metametamodel

EAST: /FMU2EA7/adlrt.ecore

☐ Is metametamodel

Source Models

IN: /FMU2EA7/modelDescription2.xmi

conforms to FmiModelDescription

Target Models

OUT: /FMU2EA7/output2.xml

conforms to EAST

Libraries

Figure 6.3.6/ A View from ATL run configuration

6.3.7 Implementing on ATL

Implementing the mapping rules on ATL was not straightforward. This is because ATL has its own characteristics. But the mapping rules can not be changed so much for fitting to the ATL implementation easier as mapping an element of FMI to an EAST-ADL element only makes sense in one way in general. Only small changes can be made to the mapping rules such as leaving out some parts of the mapping or changing the shape of the model in the way that it won't effect the model's structure. So the main challenge was to suit the ATL rules to the mapping rules.

The following ATL implementation structure has been used to apply the mapping rules. Only conversion of a RealType in FMI is shown. For all IntegerType, BooleanType, StringType and EnumerationType a similar kind of logic is used as it is in RealType:

```
---------

For all FmiScalarVariables in fmiModelDescription that has causality = input or output

    If FmiScalarVariable constitutes of a RealType

        If RealType has no declaredType

            FUNCTIONFLOWPORT is formed by FmiScalarVariable

            EAFLOAT is formed by RealType

            RANGEABLEVALUETYPE is formed by RealType

        If RealType has a declaredType

            FUNCTIONFLOWPORT is formed by FmiScalarVariable

            EAFLOAT is formed by both RealType of FmiScalarVariable and the
            RealTypeType of the declared FmiType

            RANGEABLEVALUETYPE is formed by both RealType of FmiScalarVariable
            and the RealTypeType of the declared FmiType

    Form the corresponding EAST-ADL elements from IntegerType, BooleanType,
    StringType and EnumerationType too as it is shown for RealType.

  Form the EAXML structure:

RootType

        EAXML

          TOPLEVELPACKAGESType

            EAPACKAGE

              ELEMENTSType

                  aNALYSISFUNCTIONTYPE

                  eAFLOAT

                  eAINTEGER

                  eABOOLEAN

                  eASTRING

                  eNUMERATION

                  eNUMERATIONVALUETYPE

                  rANGEABLEVALUETYPE

 Form the ANALYSISFUNCTIONTYPE by using fmiModelDescription:

 ANALYSISFUNCTIONTYPE
```

sHORTNAME

nAME

uUID

pORTS

Put the FUNCTIONFLOWPORTs formed previously into the pORTS element of the formed ANALYSISFUNCTIONTYPE

Put the other elements formed into their corresponding elements in the previously formed EAPACKAGE element inside EAXML(Put EAFLOATs under eAFLOAT etc.)

The implementation in ATL got the above shape depending on the ATL's structure. For instance, why FUNCTIONFLOWPORT is formed separately, even though its generation procedure is the same when a RealType has a declaredType or not, is because of the ATL's characteristics and having a neater and more understandable ATL code.

Some difficulties have been faced during the implementation of the above plan. The difficulties changed the shape of the above structure a little bit. The last step of the implementation couldn't be applied. So the individual elements created other than Ports haven't been put into their corresponding locations. But this is not a big issue since according to the MetaModel of East-Adl these elements can exist as individual elements. So the generated models are still valid but they don't look so neat.

The solution to this is to select the elements that are created and putting them to their corresponding location manually. Also another way is found and implemented to fix the missing step and the desired structure has been obtained by it but this implementation is not finalised yet. Only small details have been left to have it as the final version. The final ATL code can be found in **Appendix A4**.

6.3.8 Obtaining the Plug-in

As in the generation of the Editor case, Eclipse also supports a tool for auto-generating a Plug-in for an ATL application. Certain steps are applied to obtain the Plug-in. These steps can be seen in **Appendix C3**. For this plug-in, Eclipse forms a file that constitutes of Java files and a number of other type of files. But the plug-in does not run straight forward after the auto-generation of the source files. Certain changes have to be done into a couple of source files. The change took place for the reason of pointing the plug-in where the FMI and EAST-ADL metamodels are located. For making the minimum change in the source files, also the MetaModels of FMI and EAST-ADL are added into the generated file. With these changes, a user can enter the input and output location to the console and run the main method.

6.3.9 Obtaining the Executable

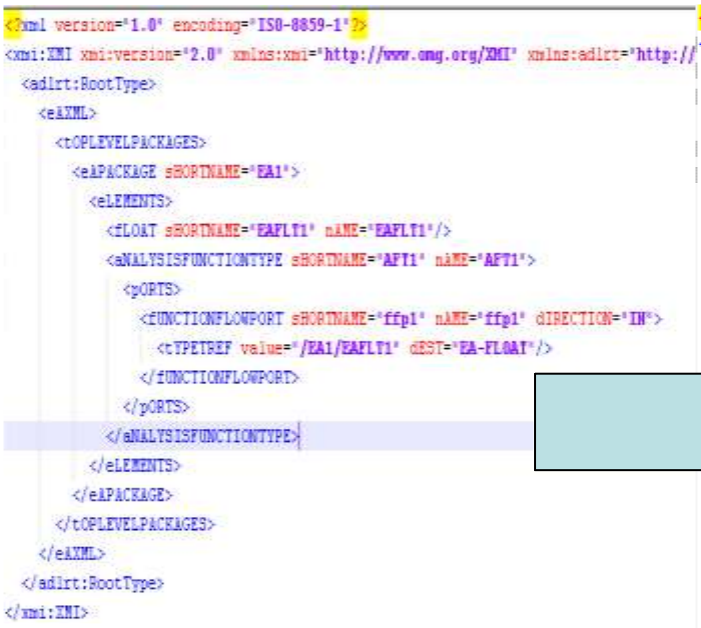
The plug-in gives a level of usability to the tool but still running an application directly from its main method is not a good option to use. It is better to put an interface between the user and the source files so that the user won't have the direct access to the source files. This is preferable since it gives a more user friendly usage of the system. Also, in this case there is a smaller chance for the user to break down the system. For these listed reasons, an executable is formed for the source files of the plug-in. This time it was rather a straight forward process. Eclipse Java export options have been used to create the jar file. This creates a stand alone application which the user even doesn't need Eclipse to run the jar file. The actual steps taken for creating the jar files have been shown in the **Appendix C4**.

6.3.10 Converting the output to EAXML format

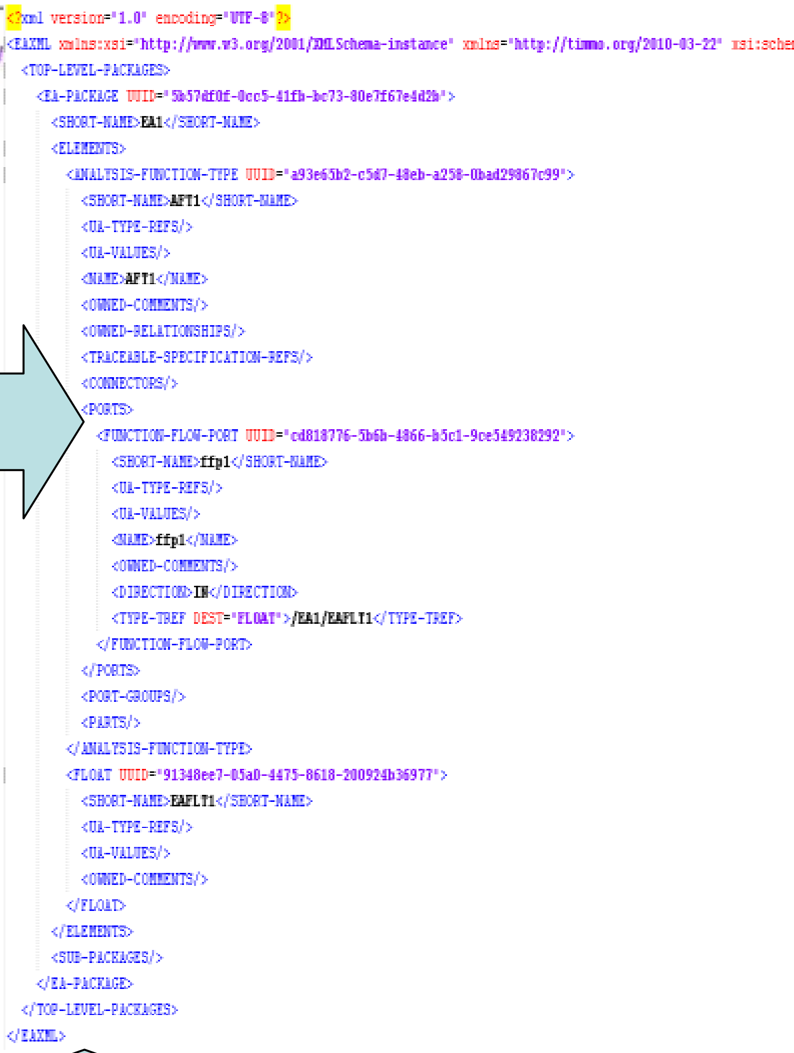
As discussed above the generated MetaModel from EMF is slightly different than the EAST-ADL XSD itself. And the ATL uses this generated MetaModel as a basis for creating the EAST-ADL models out of Modelisar ones. As a result the created EAST-ADL models have differences with the desired format. The desired format is the one that directly conforms to the XSD. This is the case for the Mentor Graphic's VSA tool. The models created by using this tool have the desired characteristics. So the structure of these models has been used as the basis to decide on which changes should be applied on the model created by the FMU2EA tool.

In Figure 6.3.10.1 there are two models which the one in the left is created by FMU2EA tool and the one in the right is created by VSA tool. If we analyze the differences step by step, the first one noticeable is the 1st, 2nd and the closing lines. These lines give information about the encoding type and the reference source which is used to create the corresponding XML. The ATL generated one has a different opening since the source used to create it is the EAST-ADL MetaModel where the source used to create the schema. The encoding difference is not important since it is an option to have different encoded models in the both sides and selecting different encodings doesn't effect how the model looks in our case.

Another difference that can be seen is that, there is a RootType element in the left model which does not exist at all in the right one. This is because the RootType element is not an element of the EAST-ADL XSD. The reason behind why this element is introduced to the XSD which is used to create the MetaModel was discussed before in the text.

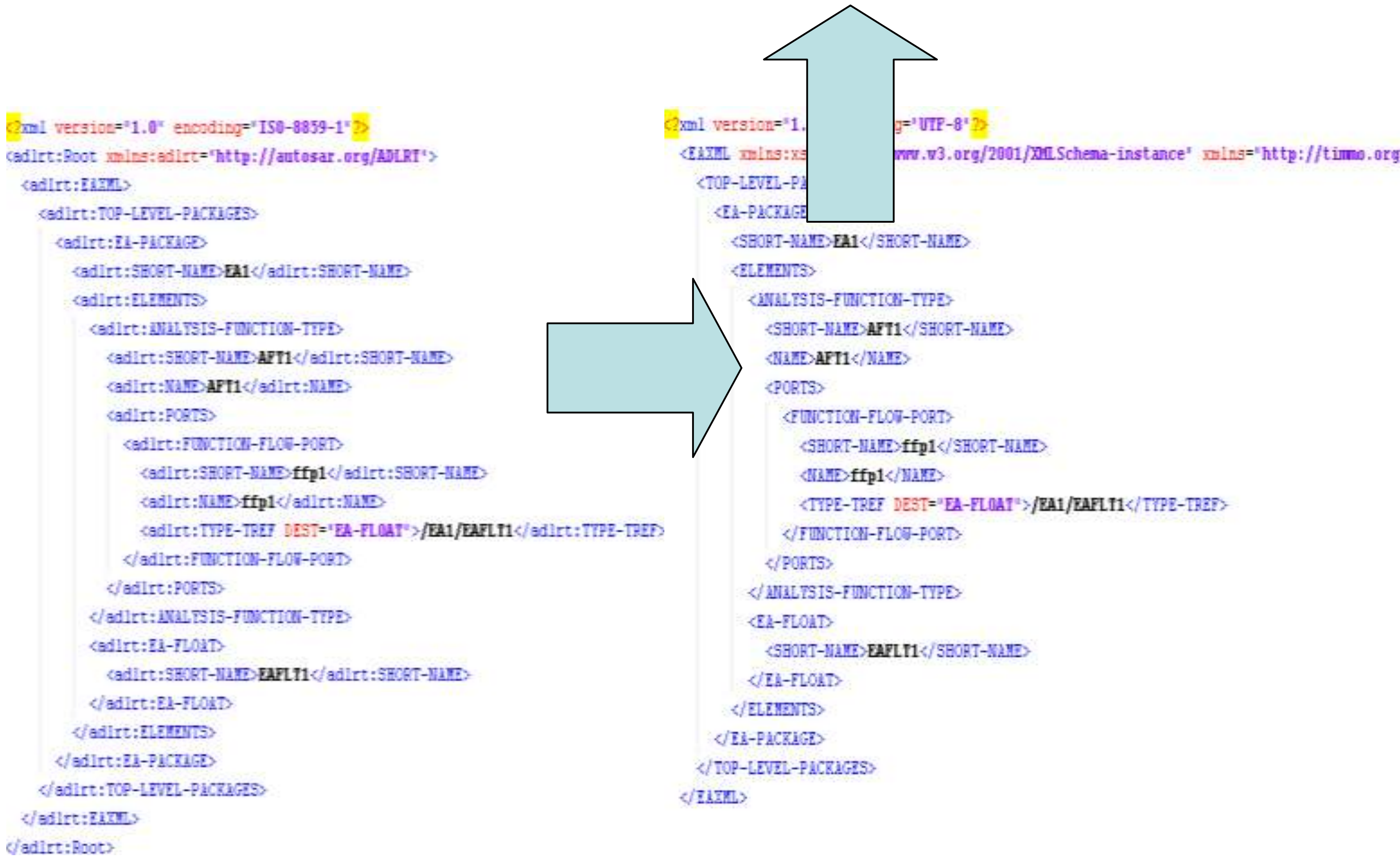


```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:adlrt="http://
  <adlrt:RootType>
    <eAXML>
      <tOPLEVELPACKAGES>
        <eAPACKAGE sHORTNAME="EA1">
          <eLEMENTS>
            <fLOAT sHORTNAME="EAFLT1" nAME="EAFLT1"/>
            <aNALYSISFUNCTIONTYPE sHORTNAME="AFT1" nAME="AFT1">
              <pORTS>
                <fUNCTIONFLOWPORT sHORTNAME="ffp1" nAME="ffp1" dIRECTION="IN">
                  <tYPETREF value="/EA1/EAFLT1" dEST="EA-FLOAT"/>
                </fUNCTIONFLOWPORT>
              </pORTS>
            </aNALYSISFUNCTIONTYPE>
          </eLEMENTS>
        </eAPACKAGE>
      </tOPLEVELPACKAGES>
    </eAXML>
  </adlrt:RootType>
</xmi:XMI>
```



```
<?xml version="1.0" encoding="UTF-8"?>
<EAXML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://timmo.org/2010-03-22" xsi:sche
  <TOP-LEVEL-PACKAGES>
    <EA-PACKAGE UUID="5b57df0f-0cc5-41fb-bc73-80e7f67e4d2b">
      <SHORT-NAME>EA1</SHORT-NAME>
      <ELEMENTS>
        <ANALYSIS-FUNCTION-TYPE UUID="a93e65b2-c5d7-48eb-a258-0bad29867c99">
          <SHORT-NAME>AFT1</SHORT-NAME>
          <UA-TYPE-REFS/>
          <UA-VALUES/>
          <NAME>AFT1</NAME>
          <OWNED-COMMENTS/>
          <OWNED-RELATIONSHIPS/>
          <TRACEABLE-SPECIFICATION-REFS/>
          <CONNECTORS/>
          <PORTS>
            <FUNCTION-FLOW-PORT UUID="cd818776-5b6b-4866-b5c1-9ce549238292">
              <SHORT-NAME>ffp1</SHORT-NAME>
              <UA-TYPE-REFS/>
              <UA-VALUES/>
              <NAME>ffp1</NAME>
              <OWNED-COMMENTS/>
              <DIRECTION>IN</DIRECTION>
              <TYPE-TREF DEST="FLOAT">/EA1/EAFLT1</TYPE-TREF>
            </FUNCTION-FLOW-PORT>
          </PORTS>
          <PORT-GROUPS/>
          <PARTS/>
        </ANALYSIS-FUNCTION-TYPE>
        <FLOAT UUID="91348ee7-05a0-4475-8618-200924b36977">
          <SHORT-NAME>EAFLT1</SHORT-NAME>
          <UA-TYPE-REFS/>
          <UA-VALUES/>
          <OWNED-COMMENTS/>
        </FLOAT>
      </ELEMENTS>
      <SUB-PACKAGES/>
    </EA-PACKAGE>
  </TOP-LEVEL-PACKAGES>
</EAXML>
```

Figure 6.3.10.1 / Modification of an EAST-ADL model which is generated by the tool to the structure valid for VSA tool



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<adlrt:Root xmlns:adlrt="http://autosar.org/ADLRT">
  <adlrt:EAXML>
    <adlrt:TOP-LEVEL-PACKAGES>
      <adlrt:EA-PACKAGE>
        <adlrt:SHORT-NAME>EA1</adlrt:SHORT-NAME>
        <adlrt:ELEMENTS>
          <adlrt:ANALYSIS-FUNCTION-TYPE>
            <adlrt:SHORT-NAME>AFT1</adlrt:SHORT-NAME>
            <adlrt:NAME>AFT1</adlrt:NAME>
            <adlrt:PORTS>
              <adlrt:FUNCTION-FLOW-PORT>
                <adlrt:SHORT-NAME>ffp1</adlrt:SHORT-NAME>
                <adlrt:NAME>ffp1</adlrt:NAME>
                <adlrt:TYPE-TREF DEST="EA-FLOAT">/EA1/EAFLT1</adlrt:TYPE-TREF>
              </adlrt:FUNCTION-FLOW-PORT>
            </adlrt:PORTS>
          </adlrt:ANALYSIS-FUNCTION-TYPE>
          <adlrt:EA-FLOAT>
            <adlrt:SHORT-NAME>EAFLT1</adlrt:SHORT-NAME>
          </adlrt:EA-FLOAT>
        </adlrt:ELEMENTS>
      </adlrt:EA-PACKAGE>
    </adlrt:TOP-LEVEL-PACKAGES>
  </adlrt:EAXML>
</adlrt:Root>
```

```
<?xml version="1.                g="UTF-8"?>
  <EAXML xmlns:xs            www.w3.org/2001/XMLSchema-instance" xmlns="http://timmo.org
    <TOP-LEVEL-PA
      <EA-PACKAGE
        <SHORT-NAME>EA1</SHORT-NAME>
        <ELEMENTS>
          <ANALYSIS-FUNCTION-TYPE>
            <SHORT-NAME>AFT1</SHORT-NAME>
            <NAME>AFT1</NAME>
            <PORTS>
              <FUNCTION-FLOW-PORT>
                <SHORT-NAME>ffp1</SHORT-NAME>
                <NAME>ffp1</NAME>
                <TYPE-TREF DEST="EA-FLOAT">/EA1/EAFLT1</TYPE-TREF>
              </FUNCTION-FLOW-PORT>
            </PORTS>
          </ANALYSIS-FUNCTION-TYPE>
          <EA-FLOAT>
            <SHORT-NAME>EAFLT1</SHORT-NAME>
          </EA-FLOAT>
        </ELEMENTS>
      </EA-PACKAGE>
    </TOP-LEVEL-PACKAGES>
  </EAXML>
```

Figure 6.3.10.2 / Modification of an EAST-ADL model, which is formed by using the generated Eclipse EAST-ADL editor, to the structure valid for VSA tool

As discussed before, the EAST-ADL MetaModel generated by EMF has differences from the XSD in the name of the elements. And no manual change has been done to it for having the same element names as in XSD. Normally this results as having slightly different Element name format in the created model too. For an example, the ANALYSIS-FUNCTION-TYPE element in the XSD is passed as aNALYSISFUNCTIONTYPE into the MetaModel so to the Model too. All of the elements and attributes have this difference. This difference can be fixed by manually changing the MetaModel but as an additional process is needed anyway for the other difference so for this version of the application no change has been done to the EAST-ADL MetaModel that is used for the conversion. The last and most critical difference is that the structures of the elements have differences. For instance, sHORTNAME and nAME are attributes of the aNALYSISFUNCTIONTYPE element for the created model by the ATL conversion, but the VSA generated model has SHORT-NAME and NAME as elements inside the ANALYSIS-FUNCTION-TYPE element. This difference is not a tolerated one. If this difference is not fixed, VSA is not able to recognize and open the model generated by ATL conversion although even all the other listed differences have been taken care of. Also this is a difference that is not possible to be fixed with changing the MetaModel manually. The reason behind this difference is thought to be sourced from the EMF model created from the XSD since this model forms the XML document. One of the possible solution paths is guessed to be in changing the generation way of the MetaModel from the XSD but implementing this solution or finding another solution is left as a future work. In this implementation version, user has to do the changes manually.

After putting some of the attributes as elements, now the file is ready to be opened by VSA which means it is modified to the desired format. The listed modifications turned the file to the one that can be seen in the right Figure of Figure 6.3.10.2. Still there is a small difference left with the VSA generated model. There are extra empty elements inside some elements. For instance, UA-TYPE-REFS in ANALYSIS-FUNCTION-TYPE in VSA generated model (Right figure of Figure 6.3.10.1). Good news is that VSA recognizes the file even if the file does not have these extra elements. So having or not having these elements is not a constraint. Also every element is supposed to have a UUID according to the EAST-ADL MetaModel. Generating a UUID is not random; it has a standard to follow. So putting UUIDs of all the elements needs an extra effort for implementing. If the models are opened in VSA, VSA has the option to generate the UUIDs of all the elements. So this process is left out instead of putting time on working on the generation of the UUIDs. If the user needs the UUIDs he can use VSA to generate it and VSA is a platform that can be assumed that it will exist on a user's PC who uses the FMU2EA tool.

There are also several differences between the desired file format of an EAST-ADL model and the created file by the generated EAST-ADL Editor. These differences are fewer and more straightforward compared to the files generated after the conversion. The structure of the XML file generated by the Editor can be seen in the left figure of Figure 6.3.10.2. The first step that has to be taken for obtaining the desired file format (as in the right figure of Figure 6.3.10.1) is to change the first 2 lines of the file and erase the Root element. The second is to erase the label 'adlrt' from all of the lines of the file. After these the file will look like as in the right figure of Figure 6.3.10.2. This is the file format obtained also after some modifications to the FMU2EA tool generated file. So, the rest of the modifications needed are the same as discussed in the previous paragraph.

7. Related approaches

There have been 2 projects that have been studied for the actual work which are KTH's Papyrus←→Simulink Exchange plug-in and Systemite AB's SystemWeaver←→Simulink plug-in as discussed before. KTH's project has been a source of inspiration for the current implementation of the FMU2EA tool whereas analyzing SystemWeaver←→Simulink application has been useful more for deriving some hints for the near future planned work of 'Integrating FMU2EA tool with Simulink'. Additionally, these projects have been an important starting point for obtaining the general motivation behind Model Driven Engineering.

7.1 Papyrus←→Simulink

There are lots of similarities between FMU2EA tool and this project. Because of this, a possible Simulink integration application can follow KTH's approach. Thus it is important to summarize the working principles of the Papyrus←→Simulink tool. The Overview of the System Mechanism from Papyrus to Simulink can be seen in Figure 7.1.2 and from Simulink to Papyrus in Figure 7.1.3.

The main notion used in the transformation can be seen in Figure 7.1.1. The process shown in this Figure is for converting a Simulink model to a Papyrus one. The exact reverse way gives the Papyrus2Simulink path. From the Tool Model to Intermediate Model, only file format (UML, XML, EMF, etc.) is changed but the model structure is kept the same. This step is called Technical Space Bridge in the Figure. When the intermediate model obtained, the structural bridge is used. This one keeps the format same but modifies the structure of the model. The application of this notion to the tool can be seen in the explanation of the system structure below.

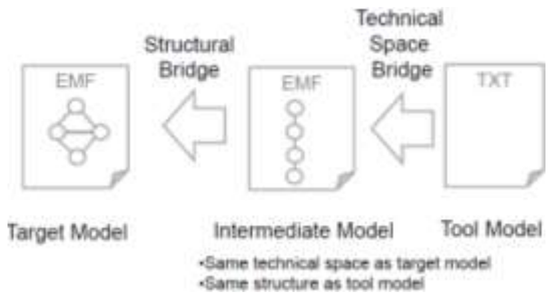


Figure 7.1.1 / Converting a Simulink Model to a Papyrus Model

→ From Papyrus 2 Matlab:

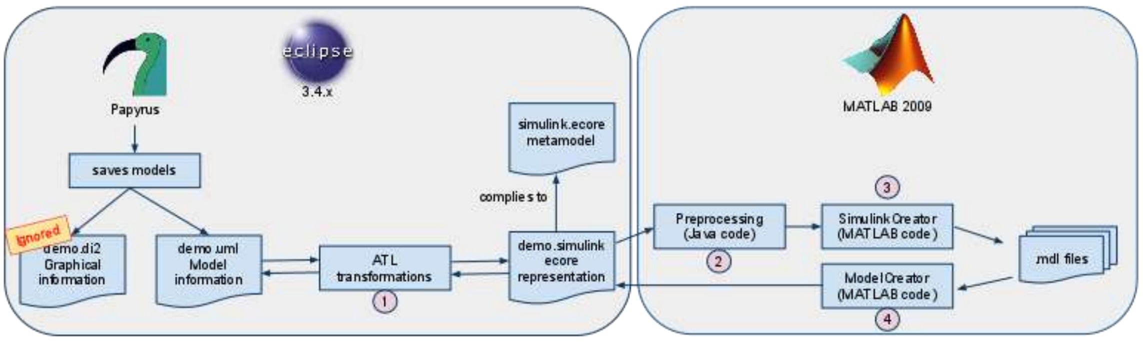

Papyrus
eclipse
3.4.x
saves models
Ignored
demo.di2 Graphical information
demo.uml Model information
ATL transformations
1
simulink.ecore metamodel
complies to
demo.simulink ecore representation
MATLAB 2009
Preprocessing (Java code)
2
Simulink Creator (MATLAB code)
3
ModelCreator (MATLAB code)
4
.mdl files

Figure 7.1.2 / Steps taken for forming a Simulink model from a Papyrus model

First a complete Papyrus Model is built up. The Papyrus Model is an East-Adl model stereotyped with UML. Papyrus model constitutes of 2 files which one of them has a .di2 and the other has a .uml extension. The .di2 file holds the graphical information of the model in hand which is not in interest for the transformation of the East-Adl model to a Simulink Model. So .di2 file is ignored in the transformation process and instead only .uml extension file is used. This file is converted to a .simulink extension file, which is a transition file, using the ATL transformation defined by KTH. This newly created file conforms to the Simulink Ecore MetaModel that is defined by KTH. A bunch of processes has to take place in Matlab before the .simulink file can be converted to an .mdl file. This is done by making use of both Java's and Matlab's features. In other words, this phase is passed by the co-operation of Java and Matlab by using the codes written by KTH. Since Matlab has a feature to use Java classes, this co-operation is not a problem. The processes in this phase can be separated into two which are Pre-processing and Simulink model creation. Pre-processing constitutes of the Java code written. In it, missing layout information is added to the .simulink file. Also it allows Matlab to query the structure of the Ecore model in the .simulink file by reading it from memory. Simulink model creation is done by the Matlab code. It sets up the Matlab environment to fulfil the process and finally creates the Simulink object by the queried structure.

→ From Matlab 2 Papyrus:

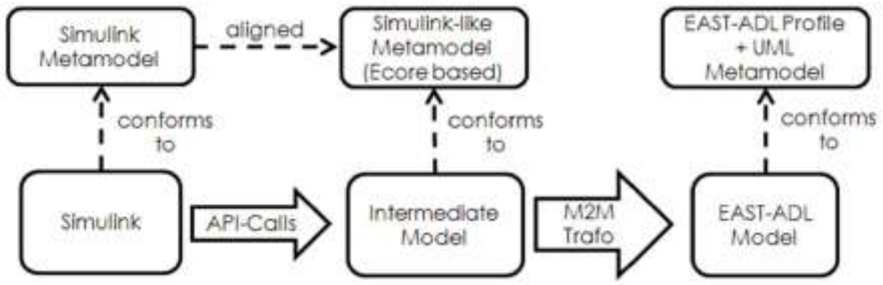


Figure 7.1.3 / Steps taken for forming a Papyrus model from a Simulink model

A similar transformation process takes place for obtaining Papyrus models from Simulink ones. Simulink communicates with Eclipse API to generate an Intermediate Model in Eclipse. This corresponds to a format transformation. The model obtained is changed in structure too for bringing in the EAST-ADL profile to the model. This model is in UML format and conforms to the EAST-ADL MetaModel.

There are limitations of the tool currently. Analyses of the current problems of the tool have been made for avoiding such possible future problems for the probable use of similar parts in the Simulink integration with the FMU2EA tool and can be found in the **Appendix D1**.

7.2 SystemWeaver←→Simulink

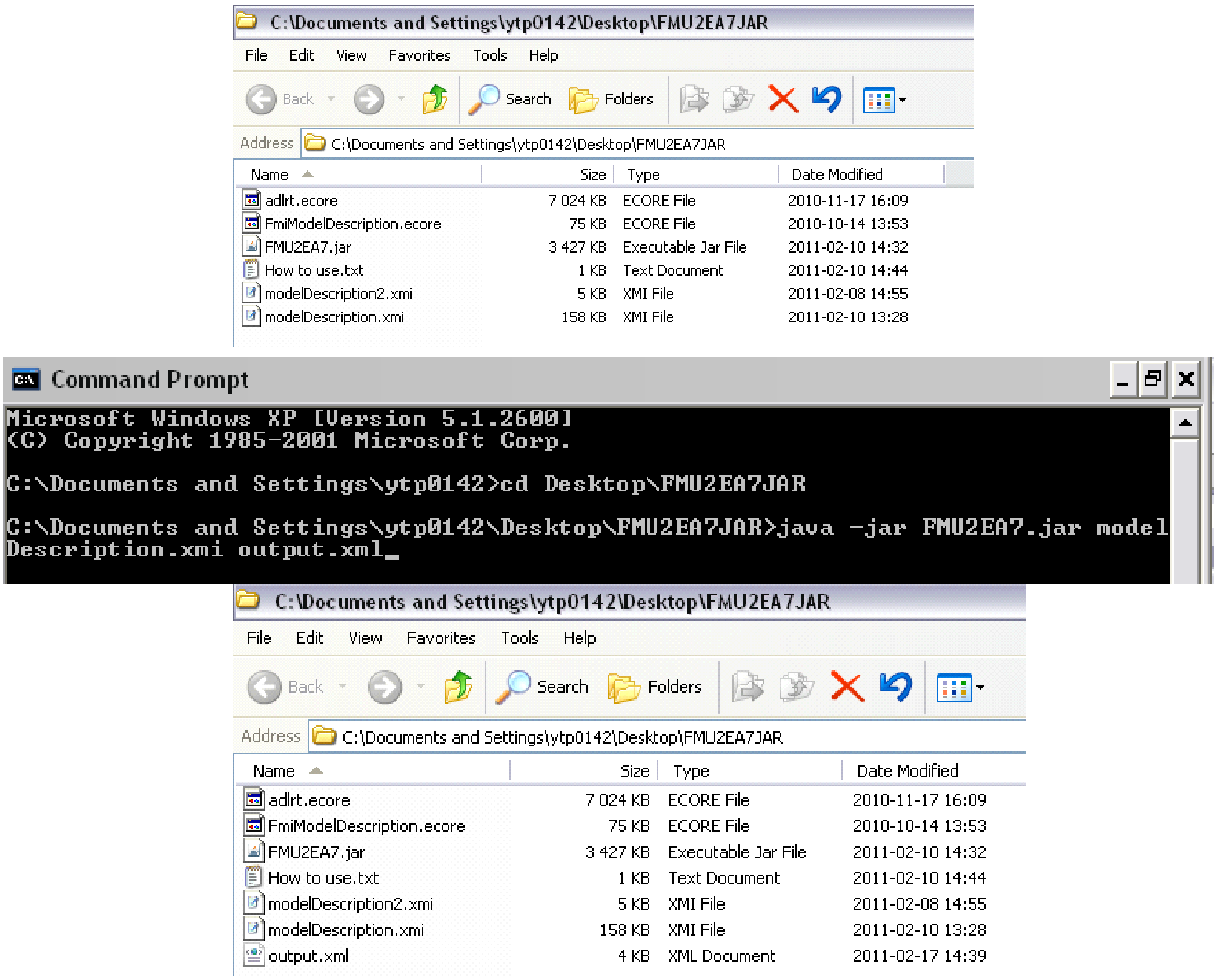
SystemWeaver is a multi user engineering information platform designed for easier product life cycle management. It has incorporated a plug-in which enables the tool to import-export Simulink models from the universal database shared from all the users. Each Simulink sub-system is converted in a SystemWeaver item, which is compliant with the SystemWeaver meta-model. All the information of the Simulink sub-system is hidden, and the created item has the same name of the Simulink sub-system with a unique identifier created when the item is exported.

8. Results

In this part, 2 use case examples will be given from the obtained FMU2EA tool and the EAST-ADL editor.These examples doesn't include the steps for converting the input file from modeldescription.xml to modeldescripton.xmi and the output files from XML to EAXML format. These steps were discussed previously in the implementation details section.

⇒ **FMU2EA TOOL USE CASE EXAMPLE**

As can be seen from the top of the Figure 8.1, there is an FMU2EA7JAR file with some files inside. This file contains the FMU2EA7.jar file and the FMI and EAST-ADL Metamodels as FmiModelDescription.ecore and adlrt.ecore. These 3 must exist and should be altogether in the same file. There are 2 other files in this example which are modelDescription.xmi and modelDescription2.xmi. These 2 files are example inputs and they don't need to be in any location since user has the ability to specify the input location. For this example, they have been put into the same file with the executable for giving a simpler showcase. One important point here is that, the input files are in XMI format rather than XML. As discussed above, the original format of the FMU is XML but the user needs to convert it to XMI manually.


C:\Documents and Settings\ytp0142\Desktop\FMU2EA7JAR
File
Edit
View
Favorites
Tools
Help
Back
Search
Folders
Address
C:\Documents and Settings\ytp0142\Desktop\FMU2EA7JAR

Name	Size	Type	Date Modified
adlrt.ecore	7 024 KB	ECORE File	2010-11-17 16:09
FmiModelDescription.ecore	75 KB	ECORE File	2010-10-14 13:53
FMU2EA7.jar	3 427 KB	Executable Jar File	2011-02-10 14:32
How to use.txt	1 KB	Text Document	2011-02-10 14:44
modelDescription2.xmi	5 KB	XMI File	2011-02-08 14:55
modelDescription.xmi	158 KB	XMI File	2011-02-10 13:28

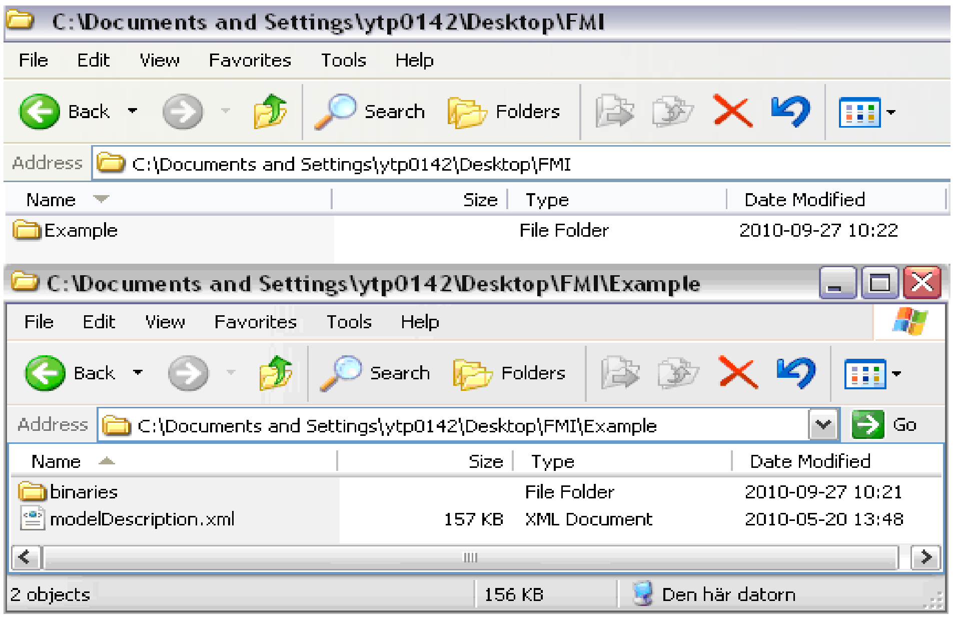
```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\ytp0142>cd Desktop\FMU2EA7JAR

C:\Documents and Settings\ytp0142\Desktop\FMU2EA7JAR>java -jar FMU2EA7.jar model
Description.xmi output.xml_
```

C:\Documents and Settings\ytp0142\Desktop\FMU2EA7JAR
File
Edit
View
Favorites
Tools
Help
Back
Search
Folders
Address
C:\Documents and Settings\ytp0142\Desktop\FMU2EA7JAR

Name	Size	Type	Date Modified
adlrt.ecore	7 024 KB	ECORE File	2010-11-17 16:09
FmiModelDescription.ecore	75 KB	ECORE File	2010-10-14 13:53
FMU2EA7.jar	3 427 KB	Executable Jar File	2011-02-10 14:32
How to use.txt	1 KB	Text Document	2011-02-10 14:44
modelDescription2.xmi	5 KB	XMI File	2011-02-08 14:55
modelDescription.xmi	158 KB	XMI File	2011-02-10 13:28
output.xml	4 KB	XML Document	2011-02-17 14:39

Figure 8.1 / An Example Input File for the Tool, The interface appearance of the Tool and the resulted output file that is obtained after the conversion

Another point is that normally the input XML file of the Modelisar Model is found in a separate file which can be seen as in Figure 8.2. This file contains the binary files of the Modelisar model plus the XML file which we are interested in. This file structure is a default for the Modelisar project. But since there is a need of modification of the XML document to XMI, it is more appropriate to put the resulting XMI into another file for protecting the structure of the original file. This desired XMI can be put anywhere wanted. If this conversion from XML to XMI is supported by the FMU2EA tool in the future then the user will be able to select the original FMI file for giving it directly as input without doing any manual operation.



C:\Documents and Settings\ytp0142\Desktop\FMI

Address C:\Documents and Settings\ytp0142\Desktop\FMI

Name	Size	Type	Date Modified
Example		File Folder	2010-09-27 10:22

C:\Documents and Settings\ytp0142\Desktop\FMI\Example

Address C:\Documents and Settings\ytp0142\Desktop\FMI\Example

Name	Size	Type	Date Modified
binaries		File Folder	2010-09-27 10:21
modelDescription.xml	157 KB	XML Document	2010-05-20 13:48

2 objects | 156 KB | Den här datorn

Figure 8.2 / An Example Modelisar Model

After the input and executable files are ready, the output can be obtained in small steps:

1. From the command prompt one should open the directory of the jar file such as:

C:\Documents and Settings\ytp0142\Desktop\FMU2EA7JAR

2. After entered the directory one should give the command:

java -jar FMU2EA7. jar

3. Leave a space and enter the input's location

4. And then leave another space and put the output's desired location

So the last line of the command prompt could look like this which will give the output:

C:\Documents and Settings\ytp0142\Desktop\FMU2EA7JAR > java -jar FMU2EA7.jar modelDescription.xmi output.xml

As can be seen from the bottom of Figure 8.1, the output.xml file appeared in the specified folder which for this example is the FMU2EA7JAR file.

The last procedure left to apply is modifying the output from XML to EAXML format. As discussed before only this structure conforms to the EAST-ADL schema and is compatible with the generated Editor and VSA tool. This process is done manually by the user for the current version but can be embedded to the tool in the future. The input and output files can be found in **Appendix B1**.

⇒ EAST-ADL EDITOR USE CASE EXAMPLE

After the Editor is started and an empty model is created, a page like in the Figure 8.3 would appear (without the opened menu). With a right click on an element, the different choices of elements that this element can include will be shown. For instance RootType can have an EAXML or AUTOSAR element inside and the Editor shows this as in the Figure 8.3. When an element is selected in the appearing menu it is added to the model tree. The Editor directs the user in additional ways when building the model such as showing the structure of the value that an attribute of an element can have such as shown in Figure 8.5. Also it has the capability of verifying the model to some extent.

An example model created by the Editor is shown in Figure 8.4. This model constitutes of separate AUTOSAR and EAST-ADL elements. Originally, it is not valid to have both elements in a common root according to the AUTOSAR and EAST-ADL MetaModel but why and how the Editor is modified in this way was discussed in “Obtaining the Editor” section previously. In the model, EAST-ADL elements are located under the EAST-ADL part of the model, more specifically under the EAXML element since the EAST-ADL elements are connected to this topmost root element. And AUTOSAR elements similarly are located under AUTOSAR element. For a more detailed step by step explanation of creating a model with the editor is given in the **Appendix B2**.

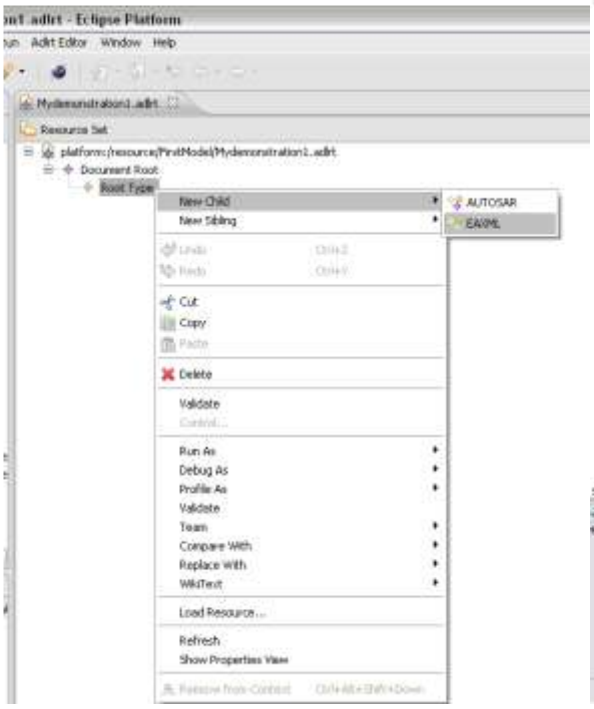


Figure 8.3 / An Empty Adlrt Model

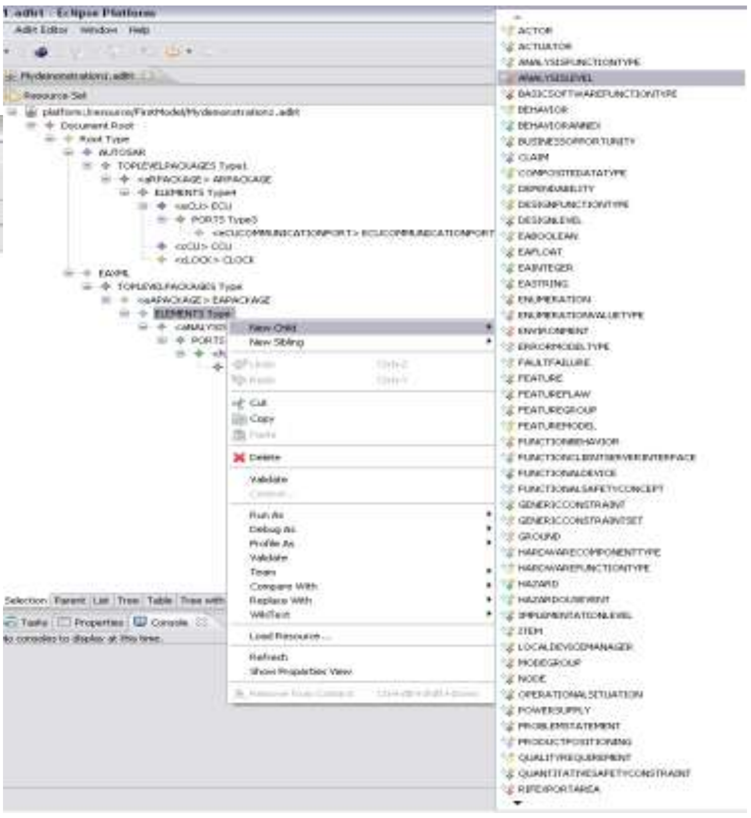
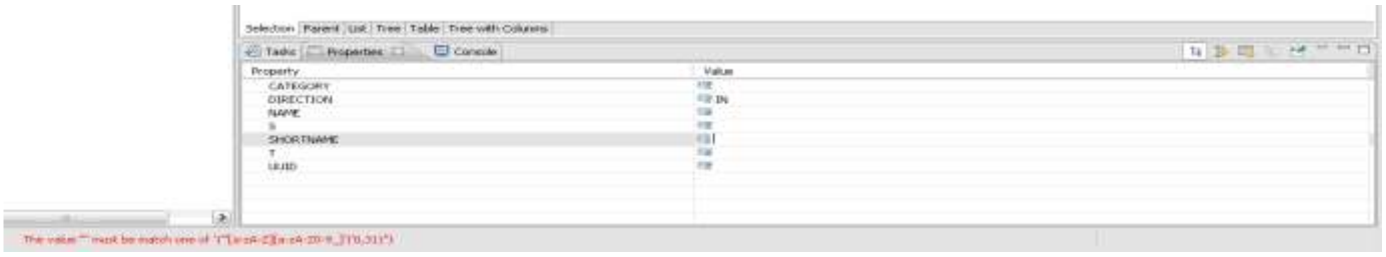


Figure 8.4 / An Example Adlrt Model with Autosar and EAST-ADL components



Property	Value
CATEGORY	
DIRECTION	IN
NAME	
S	
SHORTNAME	
T	
UUID	

The value "" must be match one of '[^[a-zA-Z][a-zA-Z0-9_]{0,31}$]'

Figure 8.5 / Editor giving information about the format of the value that should be typed to an attribute of the model

9. Conclusions

It is possible to make some conclusions based on the results of the project. For instance, one of the aims of the project was to evaluate how much Moldelisar and EAST-ADL characteristics correspond. It has been seen that a part of Modelisar fits to EAST-ADL very well in notion. And there is a meaningful way of mapping for the other part as well.

Choosing ATL for the conversion was an important choice for the project as ATL plays a crucial role. The capabilities of ATL has been an important topic of discussion. After putting some effort on the ATL implementation, a close conversion to the ideal one has been obtained. So this showed it was capable of at least doing the desired conversion. But it is a limited language; it does not give the freedom of commercial languages to the developer like Java does; it has limited tools that are supported. Also as it is not a very well known and commercial language, it has weaknesses in the documentation and example sides. Moreover, it does not give any guarantee for having compatibility with older and newer versions of ATL implementations which means that the current implementation has the risk of being incompatible with the newer versions of ATL. Although it has several disadvantages that are coming with it, it has the advantage of being a specific language for Model to Model transformation. Maybe, using a general language like Java could have taken less time for the start but in the long run ATL can become more efficient. Also looking from the compatibility perspective, using ATL had been better for integration with the other tools that has been already designed or will be designed for Maenad since they also used ATL. So, although there are disadvantages of ATL, it has been a good decision to choose this language and with the further improvements that will be made to it, the limitations and other disadvantages will be less.

Eclipse has helped the evolution of the project very much. It has been seen that Eclipse is a powerful platform which gives support to a developer with a lot of useful tools. Sometimes it lacks in the official documentation but as it is a well known platform, answers to the questions can be found from other sources of information as well. Also performance and compatibility of different tools in the platform is an issue. Sometimes the platform causes big performance shortages. To give an example, the computer crashed sometimes when using some of the tools of Eclipse but this can be dependent on the computer or on the other parameters. Using different tools together can be a challenge sometimes and unexpected behavior can be encountered which causes complex problems needed to solve. But these downsides can be tolerated since such a complex and powerful tool has a lot to offer. Accordingly, Eclipse was a nice selection especially for its EMF tools. It is also a common platform that is used by other tools of the Maenad project as well which can ease the integration.

The generated Editor during the project can be an alternative to Papyrus and VSA. Papyrus is UML based and the generated editor is XML. VSA is also XML based but it is not an open source tool. So it is definitely an advantage for the user to have the obtained EAST-ADL editor. These 3 EAST-ADL editors have different advantages and disadvantages among each other and they have a different approach of interacting with the user when building a model. So the generated Editor for sure is a handy addition to the other 2 as it has different advantages and a different way of interfacing with the user which can be preferred depending on the context.

Although not implemented yet, integration of Simulink with FMU2EA tool can be a good alternative to KTH's approach. This is because; the characteristics of the new integration will be slightly different. The possible differences that have been resulted from the pre-study are that the FMUs will be used to create S functions in Simulink instead of EASTADL blocks forming Simulink blocks (which are empty inside). The EAST-ADL models transferred to Simulink by the new Simulink integration application can also be simulated directly in Simulink as the FMUs will support the behaviour of the corresponding Simulink blocks unlike the KTH implementation which do not support direct simulation capability for the systems transferred from Papyrus to Simulink (Because they do not form any behaviour for the created Simulink blocks. They are treated as black boxes. So in order for the user to make a simulation using the transferred model, a behaviour model has to be assigned to the Simulink blocks). But it will be parallel to the KTH implementation in the way of transferring the structure information of EAST-ADL blocks to form the whole model in Simulink. Also KTH's implementation has a number of problems currently and this application can help to show the way to fix that problems and support the user with a tool that doesn’t have these problems.

To sum up, all of the conclusions that have been listed above and the other experiences gained throughout the project have been a good source of feedback for the ongoing work in Maenad and for the other related projects. Additionally, FMU2EA and EAST-ADL Editor tools have been obtained. Although currently these tools have some limitations, they can still be useful for the engineers who work with Modelisar, AUTOSAR or EAST-ADL. And the study that has been carried out for defining possible Future Work and the pre-work that has been done for some of them showed that if some of the future work plans is going to be applied, the tool can become even more powerful and functional.

10. Future Work

A number of possible future improvements has been listed throughout the project. Some of them are only ideas where the others have a rough application path in our mind. A possible future work for Simulink integration has been discussed in the Conclusions part, the rest can be seen below.

As discussed before it is assumed that a Modelisar model corresponds to an Analysis FunctionType with FunctionFlowPorts, but it can correspond to a DesignFunctionType with FunctionPowerPorts. This assumption can be removed if an interface is created which shows the possible mappings to the user. Then the user can select the appropriate mapping which will avoid assumptions. Implementing this interface can take sometime and adding this application to the tool is not so crucial, hence this improvement is not the first thing to work on.

Previously in the text, the manual steps that have to be taken by the user during the conversion process have been mentioned. Avoiding these manual steps needs modifications in the current version of the tool. These steps could be avoided with pre-processing which includes modifying the input model for making it getting into the appropriate structure and post-processing which contains the steps that are needed for obtaining the desired output structure. Pre-processing can be done with Java and post-processing can be done with the Model to Text languages Xpand or Xtend. This modification can be an important improvement since it can make the use of the tool much easier which would motivate people to use it.

The implemented ATL rules of the FMU2EA tool can be integrated to KTH's plug-in. It is thought that, swapping the KTH's ATL code with ours could be enough for the intergration besides the need of some small modifications. With this integration, the tool can support a differet user interface. In this new interface, the user can select the option "Convert to EAST-ADL" from the pop-up menu which appears when right clicked onto a Modelisar model which is opened in the Eclipse Platform, instead of executing the JAR file of the tool. When this option is used, the generated output EAST-ADL model is going to be put into the current workspace. This new interface of the FMU2EA tool could be a nice addition since some potential users are already familiar with the use of KTH´s plug-in. Also having the tool embedded in the Eclipse platform could look nicer.

As it is always good to decrease the loss of information during the transformation, making a more comprehensive mapping of FMI→EAST-ADL could be in interest. FMU can be put as the FunctionBehaviour of the generated EAST-ADL model as a first extension. Another extension could be adding the mapping of all the Modelisar elements that do not have a direct correspondance in EAST-ADL as UserAttributableElements of EAST-ADL. The structure of the FunctionBehaviour and UserAttributableElement can be seen from the **Appendix A2**. Although it can be time consuming for creating the mapping rules for every single element and implementing these in ATL, It would be good to have this modification. But perhaps the mapping of different elements and their implementations in the ATL rules will have similarities so after creating and applying the new rules for some of the elements, the rest could be easier.

A model checking process can be introduced before the conversion. For this process to be introduced, the tool should have a plug-in that opens when right clicked on an EAST-ADL model and lets the user to select a Modelisar model to be outputted on that particular EAST-ADL model. If this way of use of the tool is added then before outputting the resulted EAST-ADL elements to the selected EAST-ADL model, an analysis of that model can be done for instance by a Java code. This analysis would find the elements that have the same characteristics that are current in the EAST-ADL model and the generated model. After the analysis these elements found in the generated model would be deleted and then the rest can be outputted to the selected EAST-ADL model which would avoid creating duplicates. Another type of checking could take place in the FMU itself which is easier to implement than the previous check. Elements that would result in the same kind of EAST-ADL elements would be found and only 1 of them would be taken for the conversion which would avoid duplicates too. The implementation of these checks could be tricky. It is better to put a low priority to this feature since it does not add as much usability to the tool as the time that it will take to implement.

Doing the conversion in the opposite way would be a nice feature to add. It would let the user to make use of their already existing EAST-ADL models in the context of Modelisar. But this can be considered as a new project rather than an addition to the current one. The mapping done in the FMU→EAST-ADL perhaps won't work straightforwardly for a lot of cases in the opposite way for EAST-ADL→FMU.

It is known that all the models in Eclipse platform are kept in Ecore representation in memory and are not serialized to XML files before they are saved by the user. This is good for performance. But as mentioned before, the FMU2EA tool doesn't give the option to reach this Ecore representation because it automatically serializes the models to XML files instead of letting the user to choose. It is thought that it would be good for our tool's performance as well if the user would have the option to save a model or not. A possible way to use the tool in this way would be having an option for the conversion for either creating the ecore representation of the model or directly generating an XML. A more important benefit of this would be having compatibility with the tools that operates using such Ecore representations of models and these kinds of tools are currently available in the Maenad project. For having these advantages, a way should be found for disabling the automatic serialization of the FMU2EA tool and for reaching the Ecore representation of the model. ATL does the serialization task in the FMU2EA tool but it is not known how to direct ATL to create the Ecore representation instead of the XML file since ATL creates the XML files without giving any other options in default. But it is guessed that communicating with ATL-VM during the conversion can work for obtaining the desired kind of behaviour but more research should be done on how ATL-VM works for validating the approach.

11. References

For UML:

[1] Available: http://www.omg.org/spec/UML/2.0/ [2010-10-27]

[2] Available: http://www.sparxsystems.com/resources/uml2_tutorial/uml2_classdiagram.html [2010-09-15]

[3] Available: http://www.objectmentor.com/resources/articles/umlClassDiagrams.pdf [2010-09-12]

[4] Available: http://www.visualcase.com/tutorials/class-diagram.htm [2010-11-10]

[5] Available: http://www.omg.org/spec/UML/20090901/Infrastructure.cmof [2010-11-10]

[5] Available: http://www.omg.org/spec/UML/20090901/Superstructure.cmof [2010-11-10]

For XML:

[7] Available: http://sv.wikipedia.org/wiki/XML [2010-09-01]

[8] Available: http://schema.omg.org/spec/XMI/2.1/PDF/ [2010-12-01]

For AUTOSAR:

[9] Available: http://www.autosar.org/download/AUTOSAR_TemplateModelingGuide.pdf [2011-01-27]

For EAST-ADL:

[10] Available: http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf [2010-09-02]

[11] Available: http://www.atesst.org/home/liblocal/docs/ATESSTBrochure2010_FINAL.pdf [2011-09-02]

[12] Available: http://www.papyrusuml.org/scripts/home/publigen/content/templates/show.asp?P=146&L=EN&ITEMID=14 [2011-09-05]

For SystemWeaver:

[13] Jan Söderberg, Mats Larsson, LifeCycle Management oF Simulation Models, 2008

For KTH Simulink←→EAST-ADL exchange implementation:

[14] Available: http://code.google.com/p/kth-simulink-exchange/ [2010-09-15]

For ATL:

[15] Available: http://www.eclipse.org/m2m/atl/doc/ATL_VMSpecification%5Bv00.01%5D.pdf [2011-11-07]

[16] Available: http://wiki.eclipse.org/ATL [2010-11-08]

[17] Available: http://www.eclipse.org/m2m/atl/doc/ATL_Starter_Guide.pdf [2010-11-10]

[18] Available: http://wiki.eclipse.org/ATL/User_Guide_-_The_ATL_Language [2010-11-12]

[19] Available: http://homepages.cwi.nl/~koehler/talks/model-transformation.pdf [2010-11-24]

[20] Available: http://wiki.eclipse.org/ATL_3.1.0_New_and_Noteworthy [2010-12-16]

For Eclipse:

[21] Available: http://www.openarchitectureware.org/pub/documentation/4.3.1/html/contents/emf_tutorial.html [2010-10-15]

[22] Available: http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.jdt.doc.user/tasks/tasks-33.htm [2010-12-11]

For FMU and FMI:

[23] Available: http://www.functional-mockup-interface.org/ [2010-11-18]

[24] Available: http://www.modelica.org/ [2011-01-16]

[25] Available: http://www.modelica.org/publications/newsletters/2010-1/index_html#item9 [2011-01-16]

[26] Available: http://www.qtronic.de/en/fmusdk.html [2010-12-05]

General Background Reading:

[27] Available: http://www.dspace.de/shared/data/pdf/tl_timeline/TargetLink-2002-Volvo.pdf [2010-09-01]

[28] Available: http://www.md.kth.se/~biehl/files/papers/toolint.pdf [2010-09-10]

12. Appendix

Appendix includes the information that supports the explanation of the topics discussed previously. The details that are not covered before are uncovered here.

A. Appendix A

This section gives further information about the topic “Correspondence of Modelisar with EAST-ADL”.

1. FMI MetaModel

In this appendix, the general structure of the Fmi MetaModel will be shown.

The Figure A1.1 shows the top view of the structure of the FMI MM. All the elements of the FMI MM can be seen here, and the attributes of the elements can be found in the following Figures. It should be noted that UnitDefinitions, DefaultExperiment and VendorAnnotions are not used in this version of the mapping.

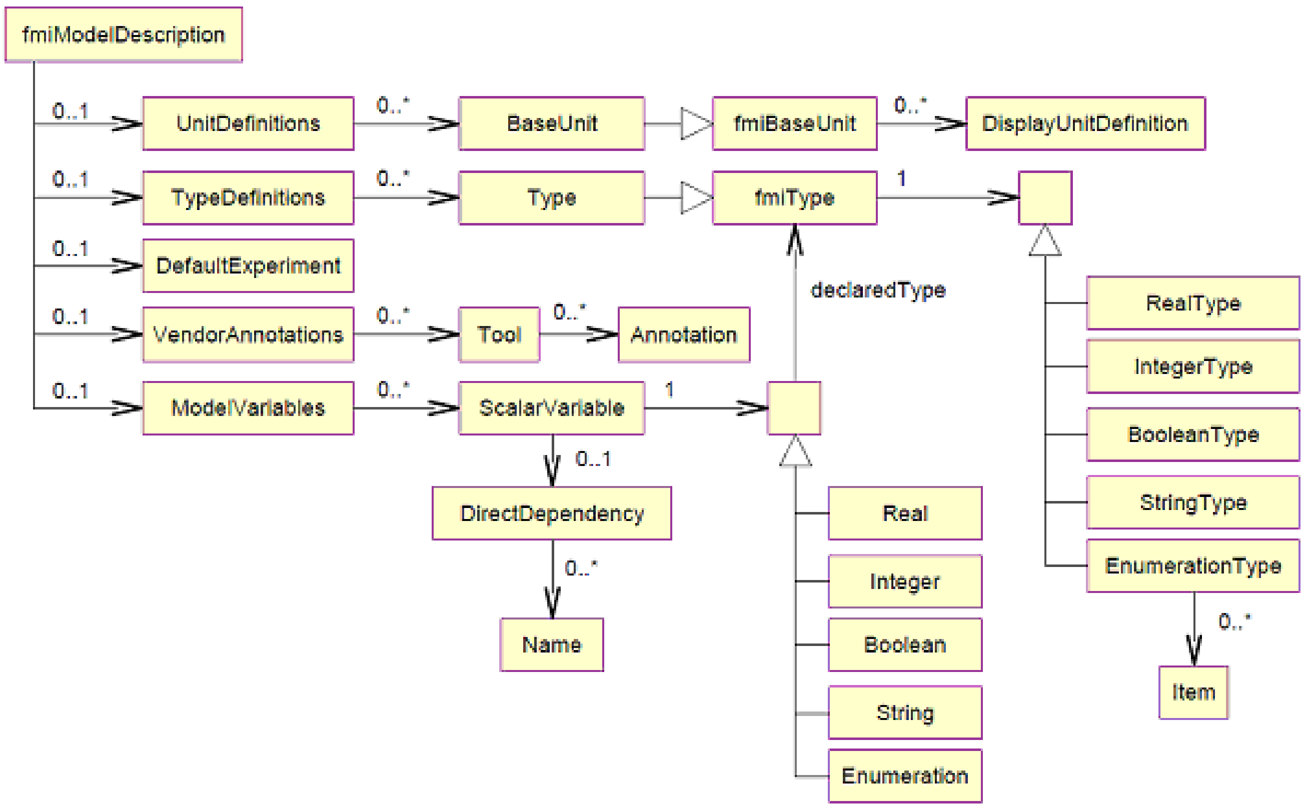


Figure A1.1

In Figure A1.2, Unit Definitions Element is shown. This element can have zero or several BaseUnit Elements. The BaseUnit Element has an attribute and can have zero or several DisplayUnitDefinition. This element is not used in the mapping.

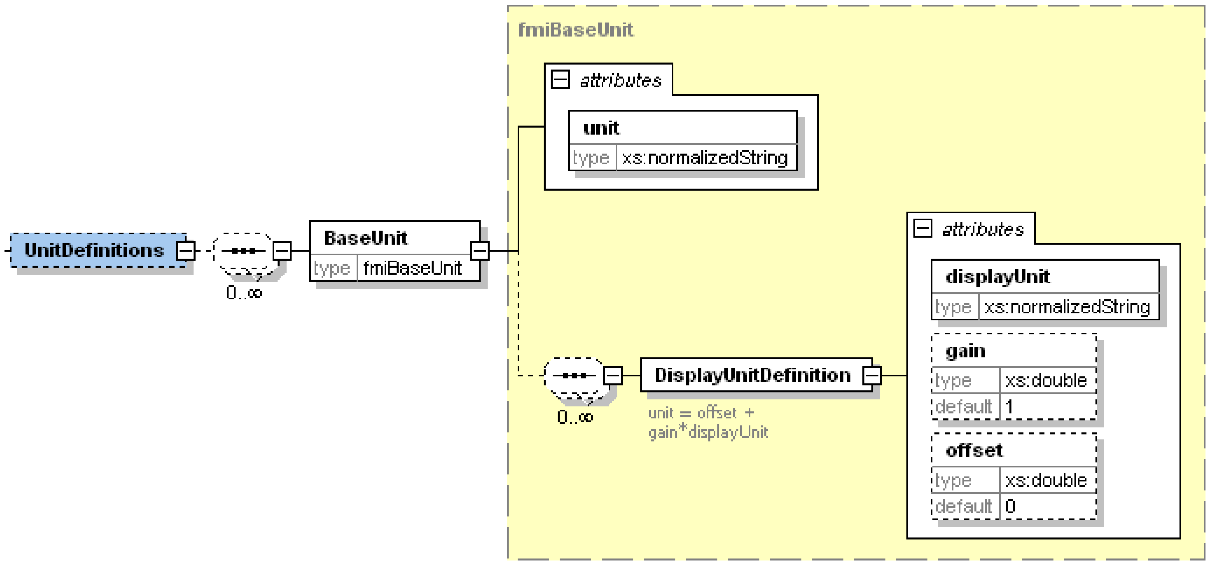


Figure A1.2

In Figure A1.3, DefaultExperiment element is shown. This element has a number of attributes but holds no elements. This element is not used in the mapping.

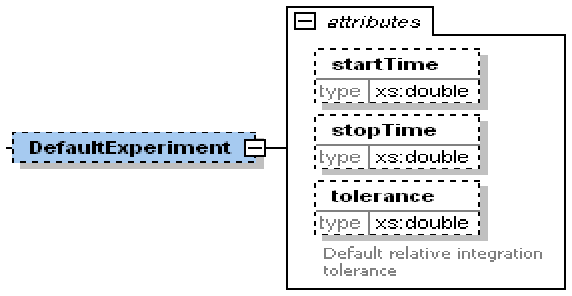


Figure A1.3

In Figure A1.4, VendorAnnotations Element is shown. This element can have zero or several Tool Elements. The Tool Element has an attribute and can have zero or several Annotation elements. This element is not used in the mapping.

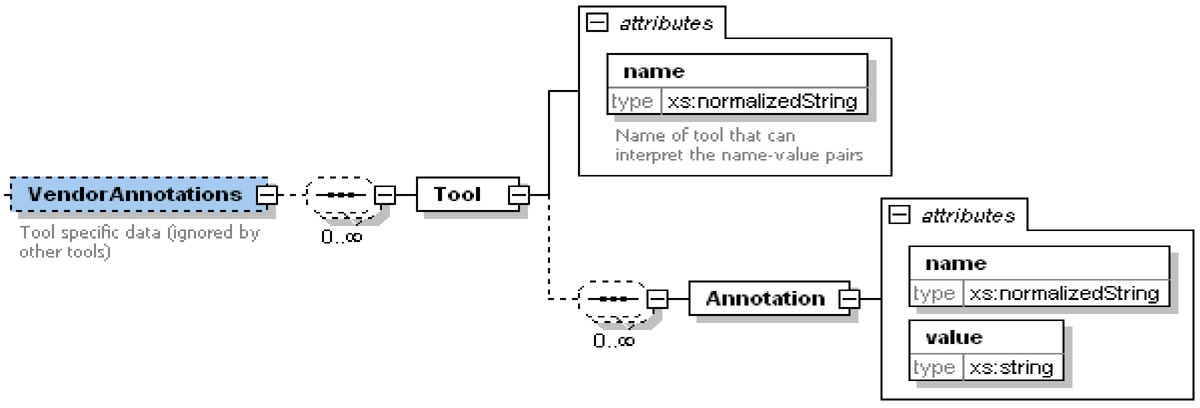


Figure A1.4

In Figure A1.5, TypeDefinitions Element is shown. This element can have zero or several TypeI Elements. The Type Element has attributes and can have one of RealType, IntegerType, BooleanType, StringType or EnumerationType. This element is rather important for the mapping. They represent the variables of the Modelisar model.

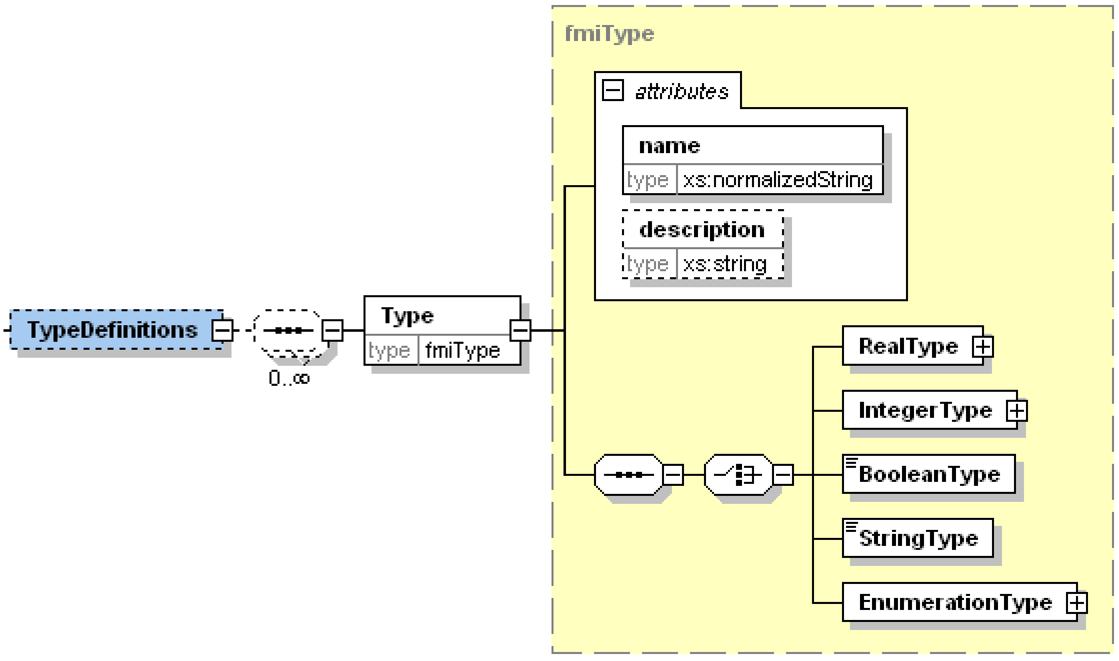


Figure A1.5

In FigureA1.6, RealType and IntegerType elements of Type Element is shown. They have some attributes and part of them are used in the mapping but not all.

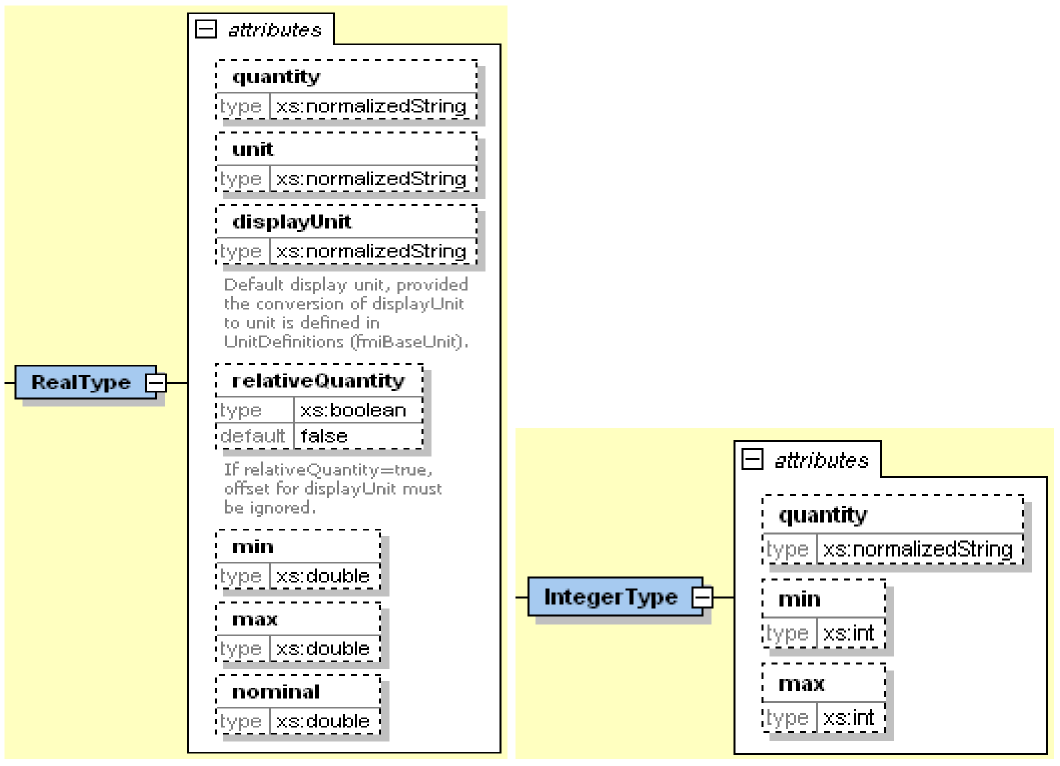


Figure A1.6

In Figure A1.7, EnumerationType element of Type Element is shown. It has some attributes and it can hold several Item Elements.

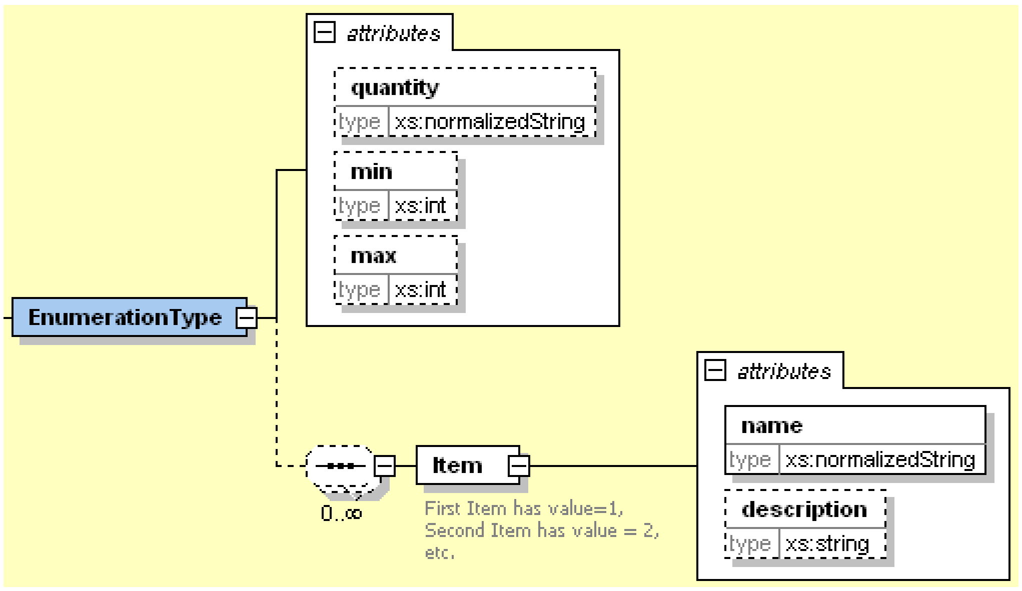


Figure A1.7

In Figure A1.8, ModelVariables Element is shown. This element can have zero or several ScalarVariable Elements. The ScalarVariable Element has attributes and can have one of Real, Integer, Boolean, String or Enumeration. This element is rather important for the mapping. They represent the variables of the Modelisar model as well. They can give reference to Type Elements and has the information that shows whether a variable is used inside the model or it is a port, the gateway of connection with other models.

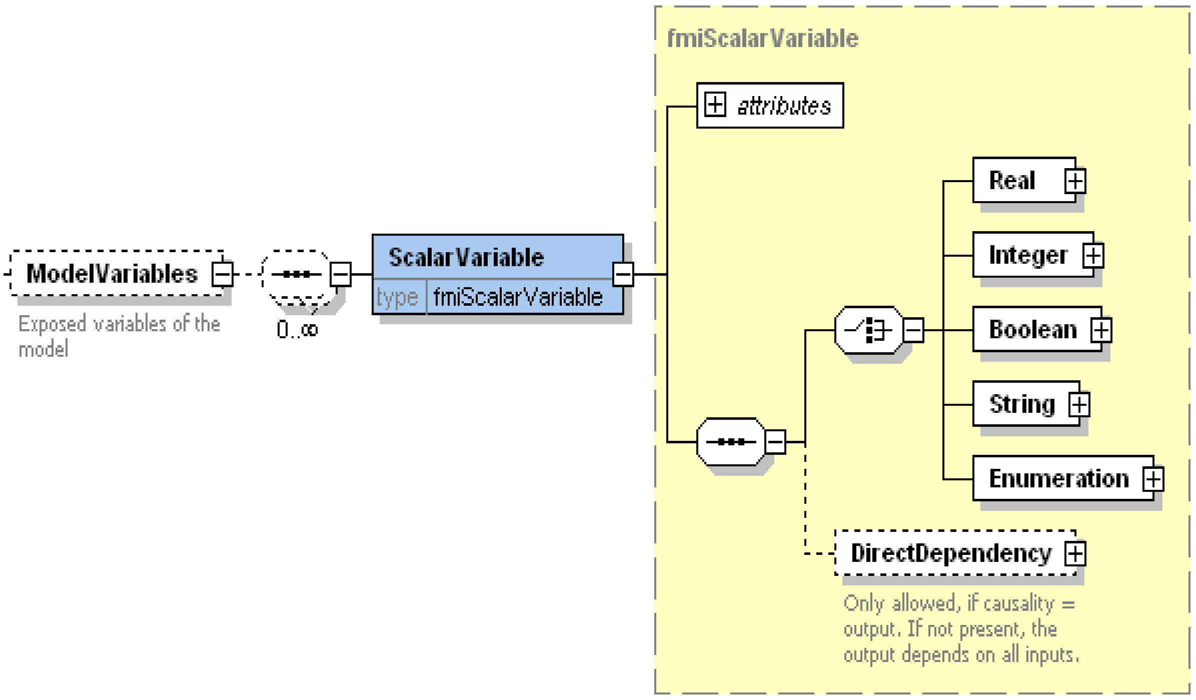


Figure A1.8

In Figure A1.9 attributes of the ScalarVariable can be seen. The attribute that shows whether the current ScalarVariable is a port or not is 'causality' which makes an important one.

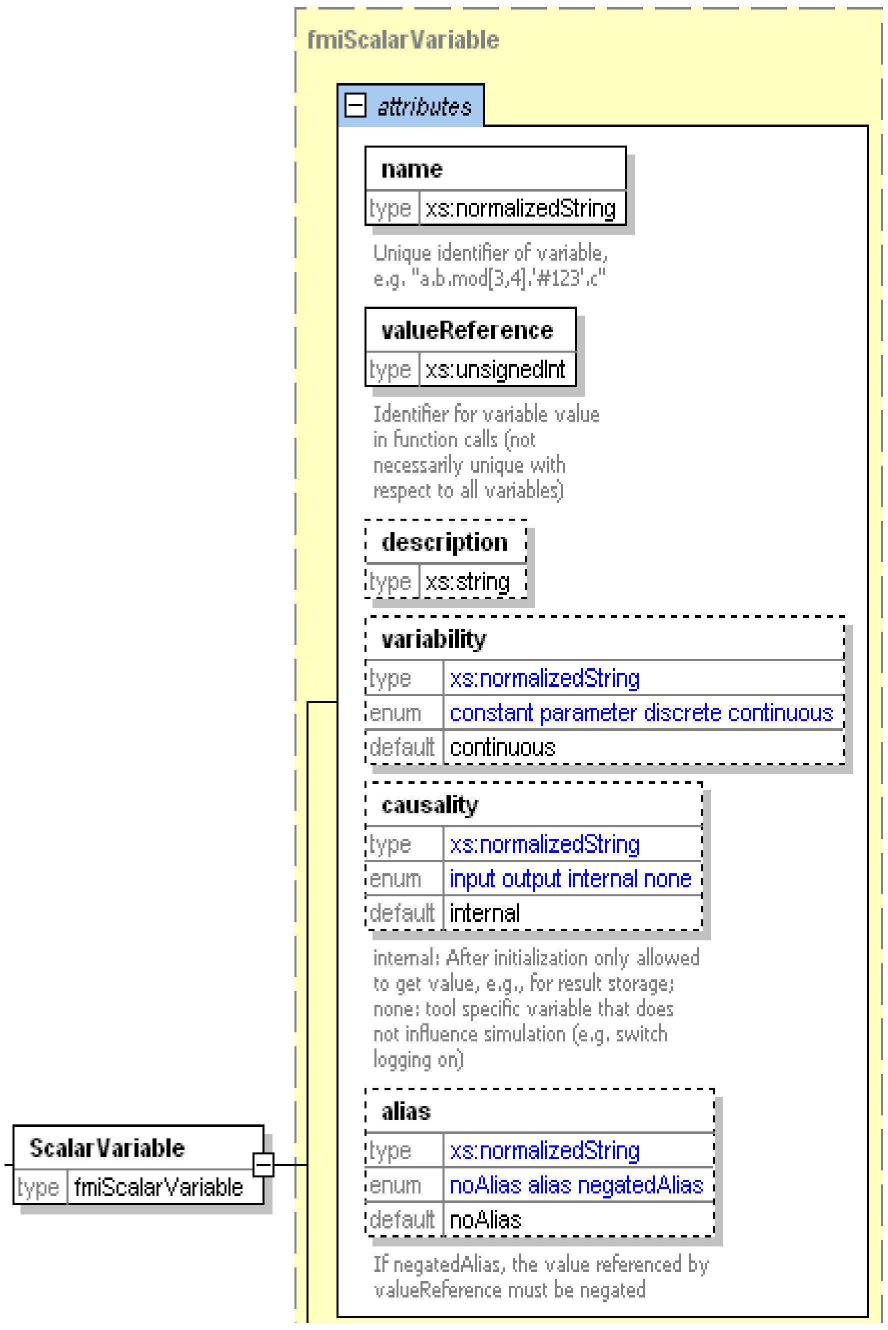

fmiScalarVariable
attributes
name
type xs:normalizedString
Unique identifier of variable, e.g. "a.b.mod[3,4].'#123'.c"
valueReference
type xs:unsignedInt
Identifier for variable value in function calls (not necessarily unique with respect to all variables)
description
type xs:string
variability
type xs:normalizedString
enum constant parameter discrete continuous
default continuous
causality
type xs:normalizedString
enum input output internal none
default internal
internal: After initialization only allowed to get value, e.g., for result storage; none: tool specific variable that does not influence simulation (e.g. switch logging on)
alias
type xs:normalizedString
enum noAlias alias negatedAlias
default noAlias
If negatedAlias, the value referenced by valueReference must be negated
ScalarVariable
type fmiScalarVariable

Figure A1.9

In Figure A1.10 Real, Integer, Boolean and String elements are shown which are a part of the ScalarVariable Element. Not all of the elements are used in the mapping.

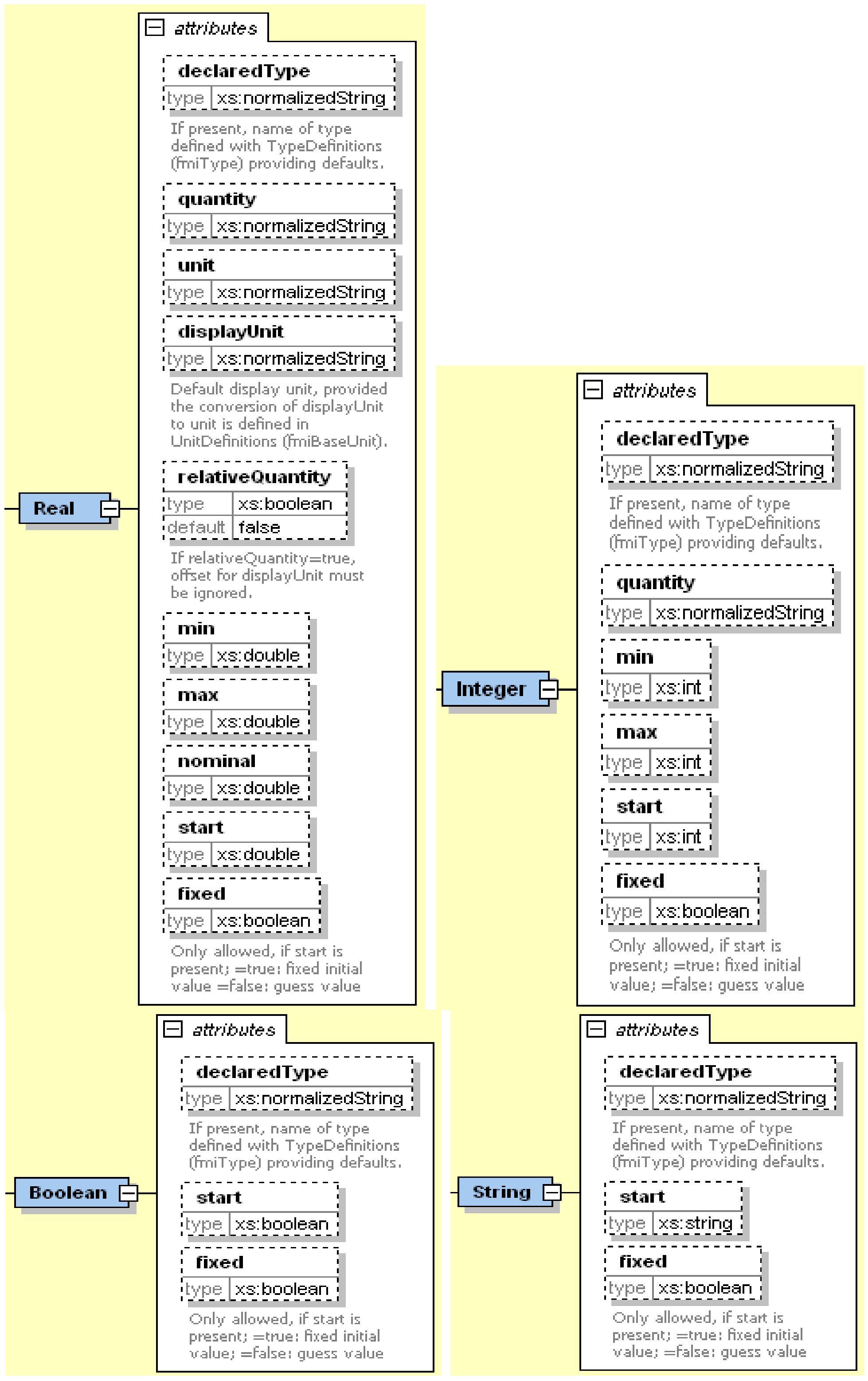
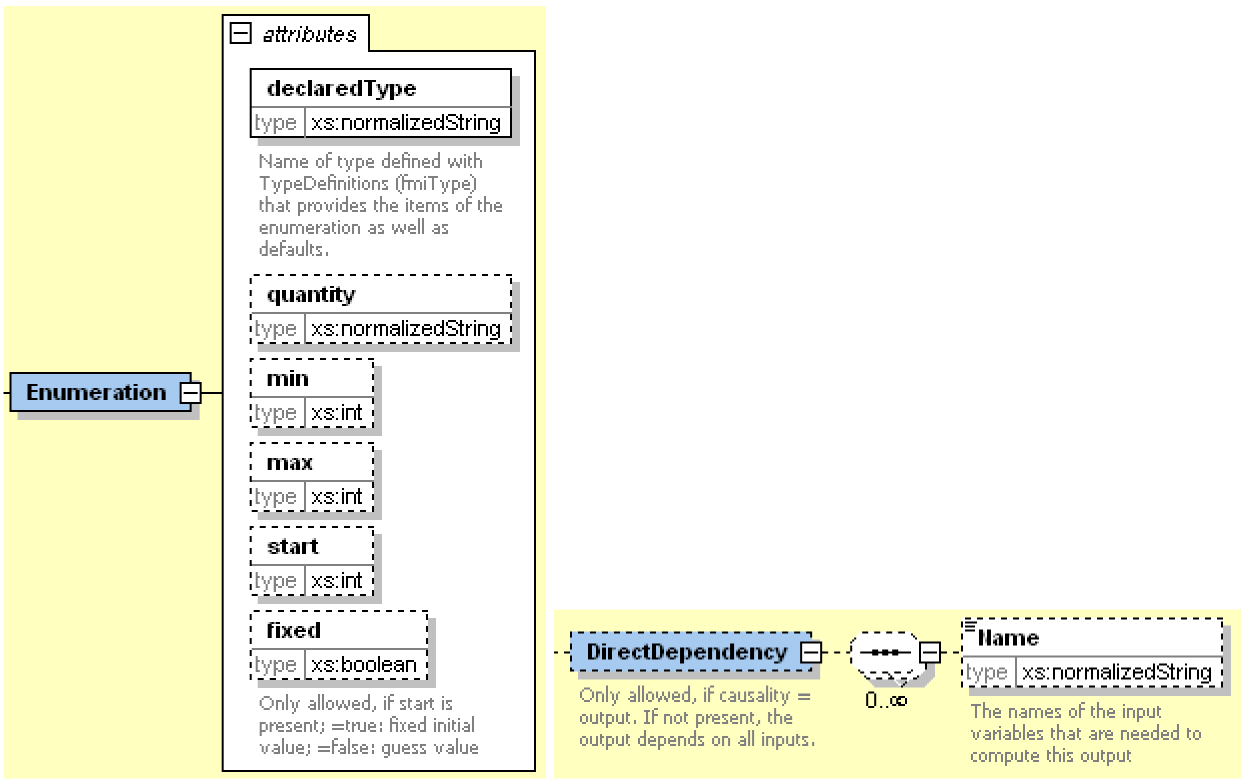

Real
attributes
declaredType
type xs:normalizedString
If present, name of type defined with TypeDefinitions (fmiType) providing defaults.
quantity
type xs:normalizedString
unit
type xs:normalizedString
displayUnit
type xs:normalizedString
Default display unit, provided the conversion of displayUnit to unit is defined in UnitDefinitions (fmiBaseUnit).
relativeQuantity
type xs:boolean
default false
If relativeQuantity=true, offset for displayUnit must be ignored.
min
type xs:double
max
type xs:double
nominal
type xs:double
start
type xs:double
fixed
type xs:boolean
Only allowed, if start is present; =true: fixed initial value =false: guess value
Integer
attributes
declaredType
type xs:normalizedString
If present, name of type defined with TypeDefinitions (fmiType) providing defaults.
quantity
type xs:normalizedString
min
type xs:int
max
type xs:int
start
type xs:int
fixed
type xs:boolean
Only allowed, if start is present; =true: fixed initial value; =false: guess value
Boolean
attributes
declaredType
type xs:normalizedString
If present, name of type defined with TypeDefinitions (fmiType) providing defaults.
start
type xs:boolean
fixed
type xs:boolean
Only allowed, if start is present; =true: fixed initial value; =false: guess value
String
attributes
declaredType
type xs:normalizedString
If present, name of type defined with TypeDefinitions (fmiType) providing defaults.
start
type xs:string
fixed
type xs:boolean
Only allowed, if start is present; =true: fixed initial value; =false: guess value

Figure A1.10

In Figure A1.11 Enumeration and DirectDependency elements are shown which are a part of the ScalarVariable Element. Enumeration is used but DirectDependency is not used in the mapping.



Name of type defined with TypeDefinitions (fmiType) that provides the items of the enumeration as well as defaults.

Only allowed, if start is present; =true: fixed initial value; =false: guess value

Only allowed, if causality = output. If not present, the output depends on all inputs.

The names of the input variables that are needed to compute this output

Figure A1.11

2. EAST-ADL MetaModel

In this section the part of the EAST-ADL MetaModel that is used in the mapping is shown.

Figure A2.1 shows the possible element types. Every element of the EAST-ADL should be a member of one of these. As can be seen, all the elements are inheriting from the Identifiable element which represents an AUTOSAR element. So every element in EAST-ADL carries AUTOSAR characteristics. This part of the MetaModel also defines the structure of an EAST-ADL model.

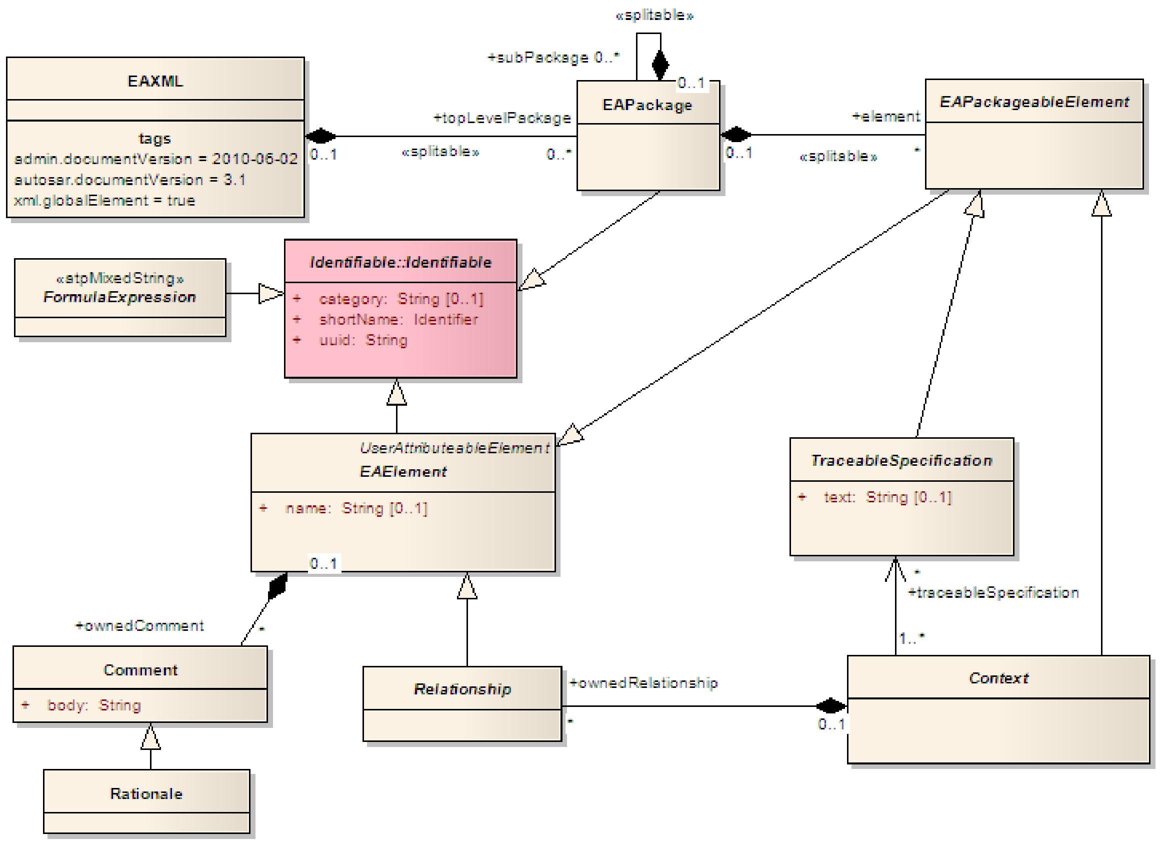


Figure A2.1

Figure A2.2 shows the structure of a FunctionType. Since AnalysisFunctionType is one of the core elements of the mapping, this part of the MetaModel plays a key role. As mentioned before, there is also a DesignFunctionType which is a possible element to be formed from the Modelisar model but as it was decided to map every Modelisar model to an AnalysisFunctionType, this element is not used. Another important element here is FunctionPort which is a part of the FunctionType.

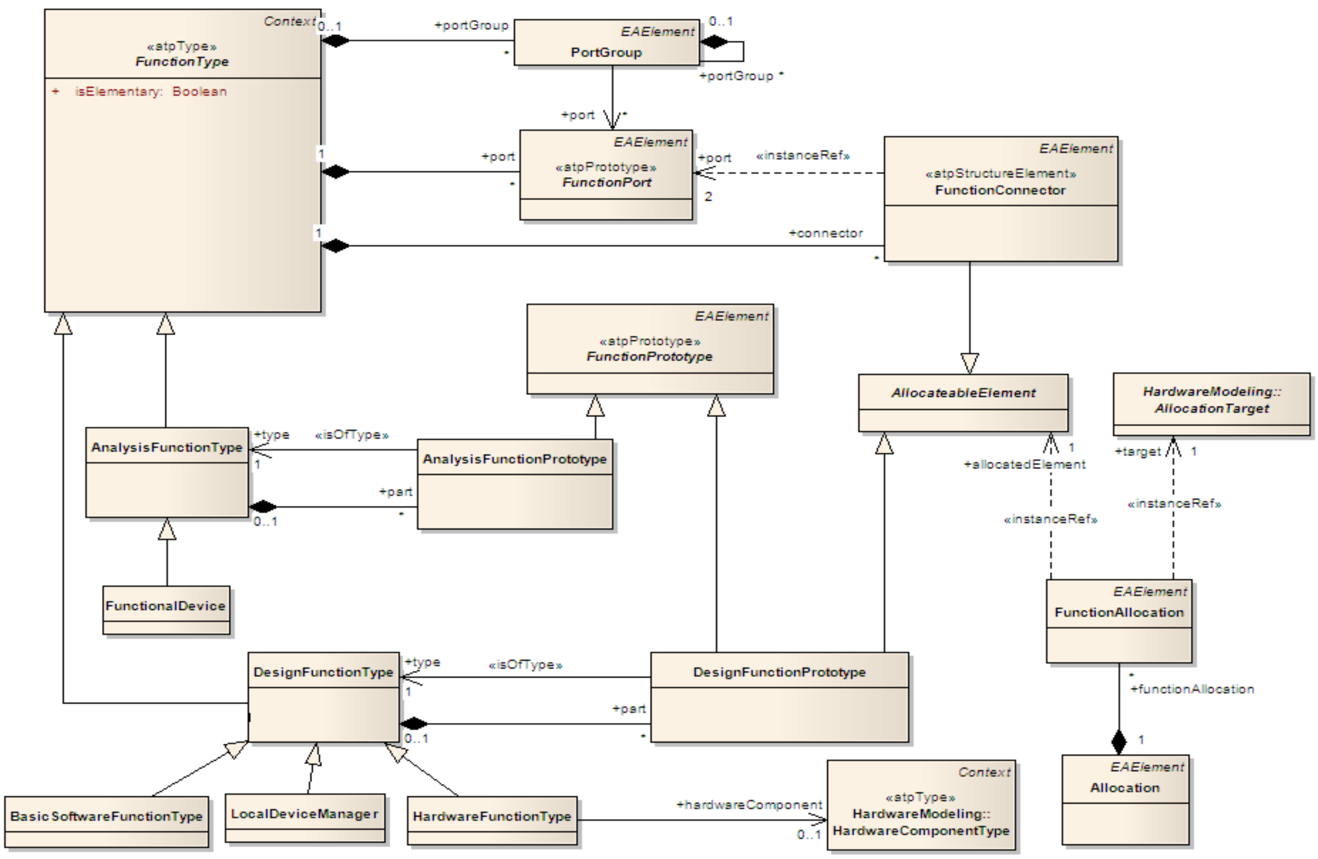


Figure A2.2

Figure A2.3 shows the structure of a FunctionPort. As mentioned before a FunctionPort can be either a FlowPort, a PowerPort or a ClientServerPort but it is decided to only use FlowPorts. FunctionFlowPort should have a DataType type.

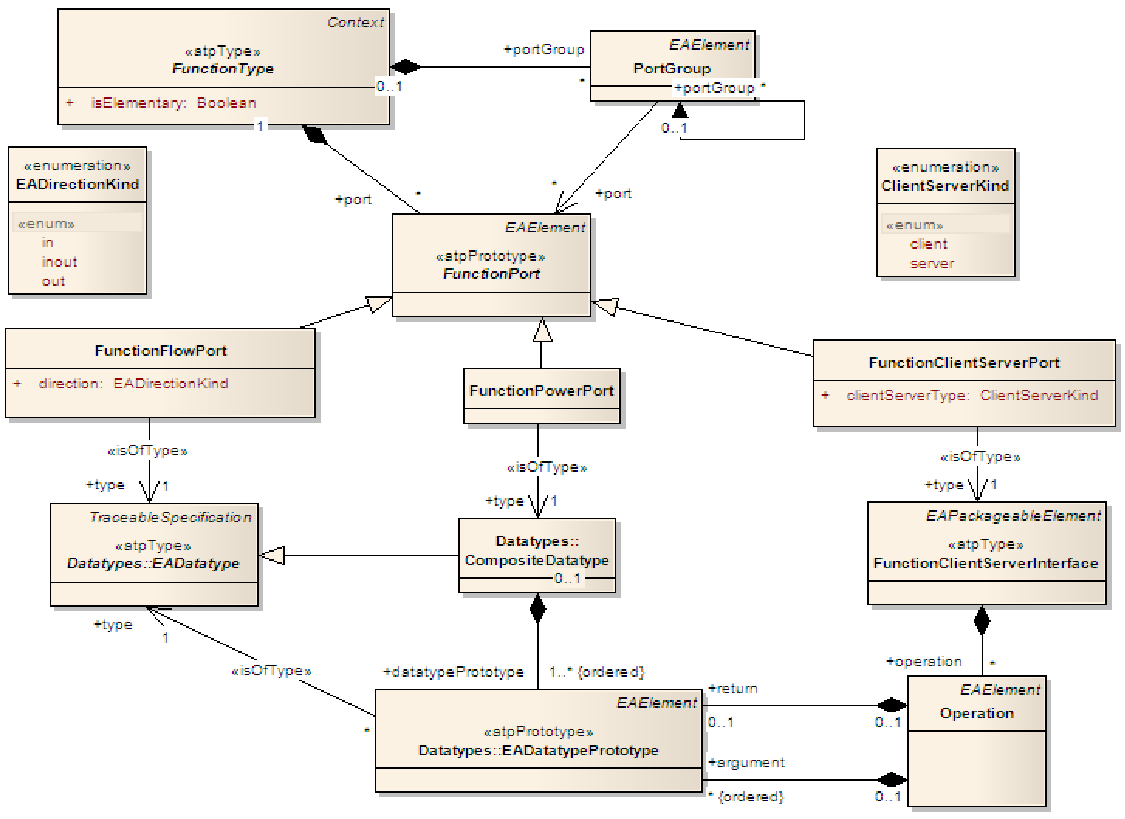


Figure A2.3

In this figure EaDatatype element is shown. This element can be either EABoolean, EAString, EAFloat, EAInteger or Enumeration for the mapping. Also EnumerationValueType and RangeableValueType are used for supporting the information of the DataTypes.

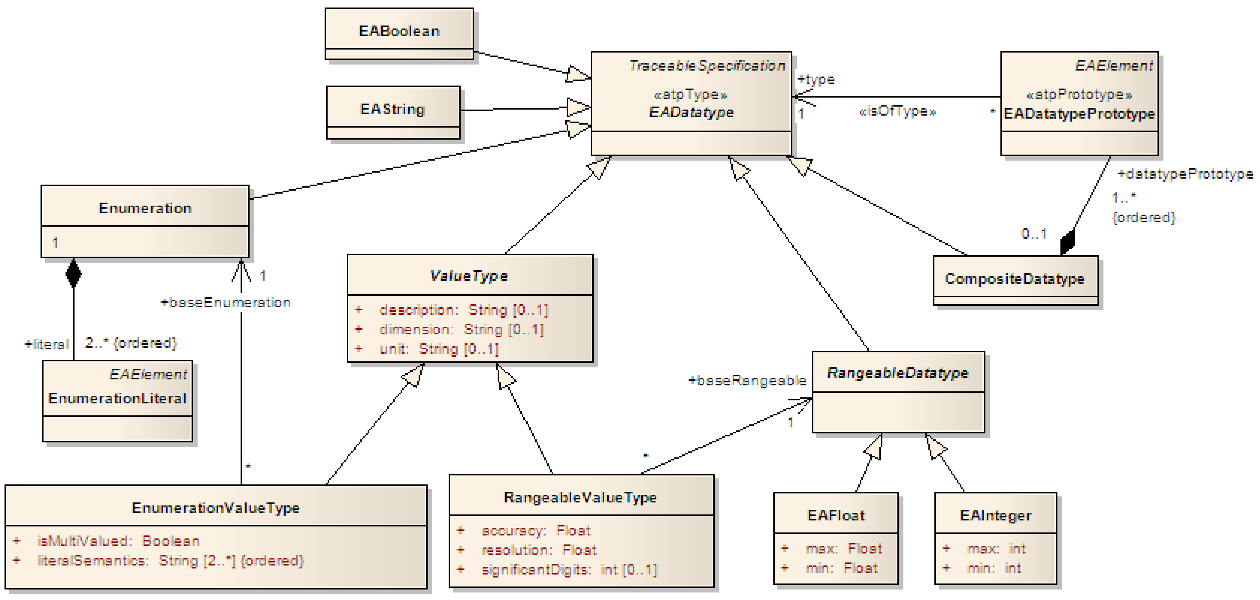


Figure A2.4

Figure A2.5 shows another view of FunctionType. This time FunctionTrigger and FunctionBehaviours' relations with FunctionType are shown. These elements and their relation with FunctionType is not used in the mapping rules but can be a part of FutureWork. For instance, FunctionBehaviour of the Model could be the mapped Modelisar model.

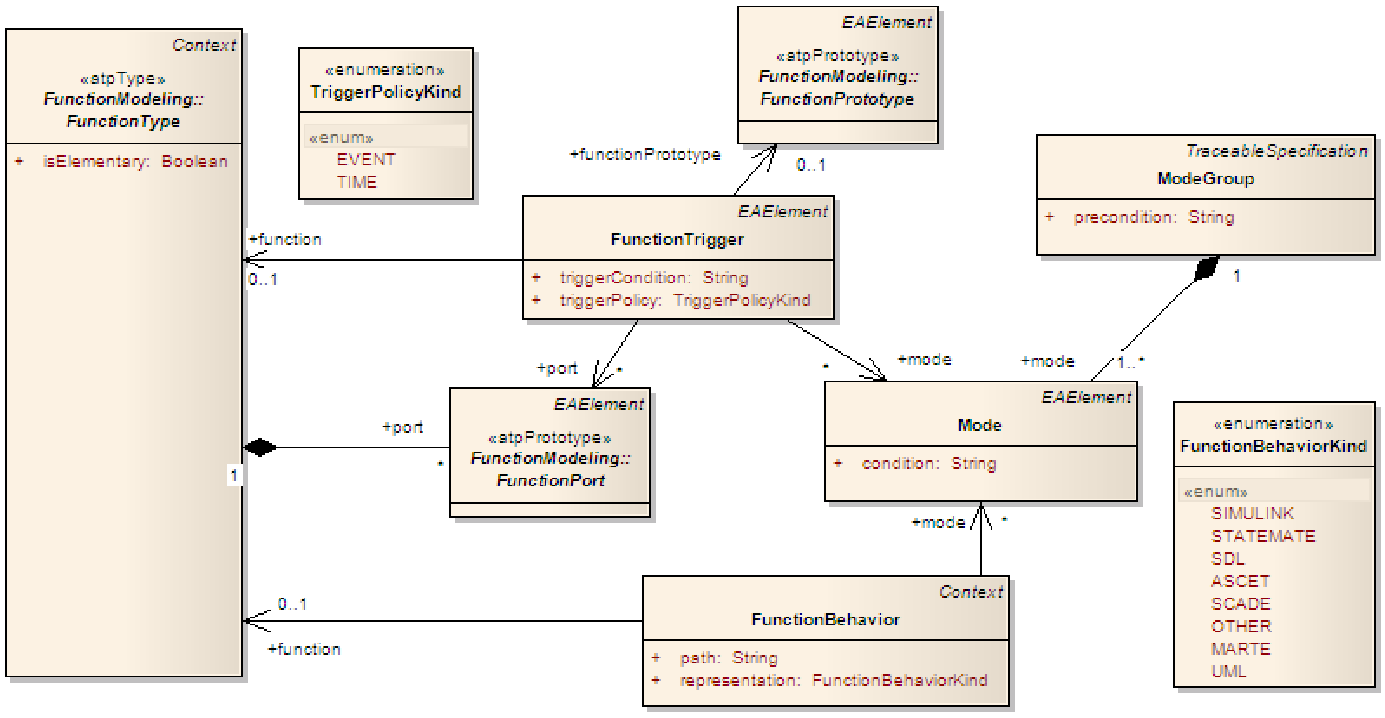


Figure A2.5

This figure shows UserAttributableElement which is not a part of the current implementation but could be included in the FutureWork. Every element in Modelisar that does not have a direct mapping can be mapped to a UserAttributableElement.

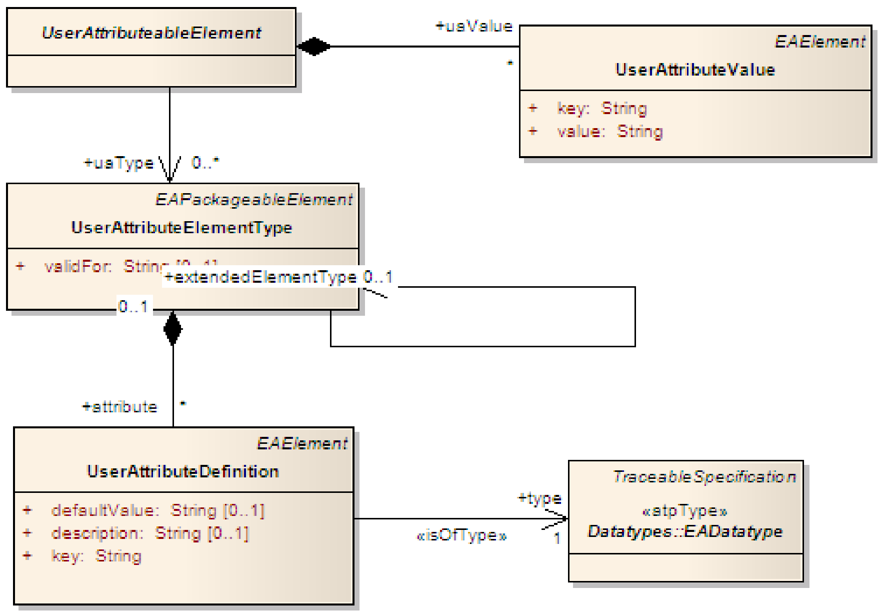


Figure A2.6

3. Mapping Rules

Mapping Rules

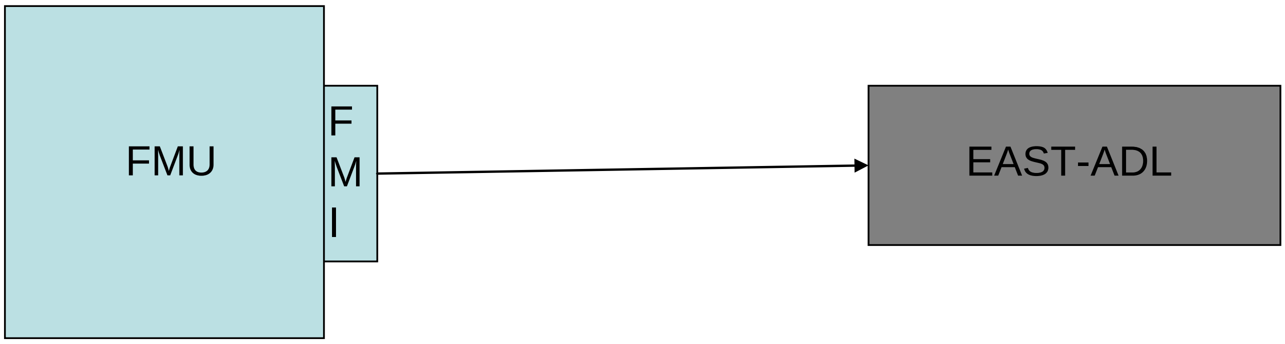


Figure A3.1

How is the conversion done

The main notion behind the conversion is the mapping between an FMU (Functional Mockup Unit) and an EAST-ADL model. The overall look to the mapping is that an FMU corresponds to an AnalysisFunctionType element with (I/O)Ports in EAST-ADL metamodel. In reality, an Fmu could also be mapped to a DesignFunctionType, but in this implementation it will be assumed that all FMUs are AFTs. This assumption has to be done as it is not possible to make a decision whether the mapping should be done to an AFT or a DFT but the user can modify the EAST-ADL model in the way that it should be afterwards the conversion. The same situation applies for selecting whether a port is FunctionFlowPort, FunctionPowerPort or FunctionClientServerPort, and it is assumed that all ports are FunctionFlowPorts during the mapping process.

In other words, FMU has all the relevant information which is needed to form the Ports, the Data Types which corresponds to Ports' types. And mainly, with Ports and their corresponding Data Type elements, one can form an Analysis Function Type with some additional specifications which can still be found in FMU and mapped.

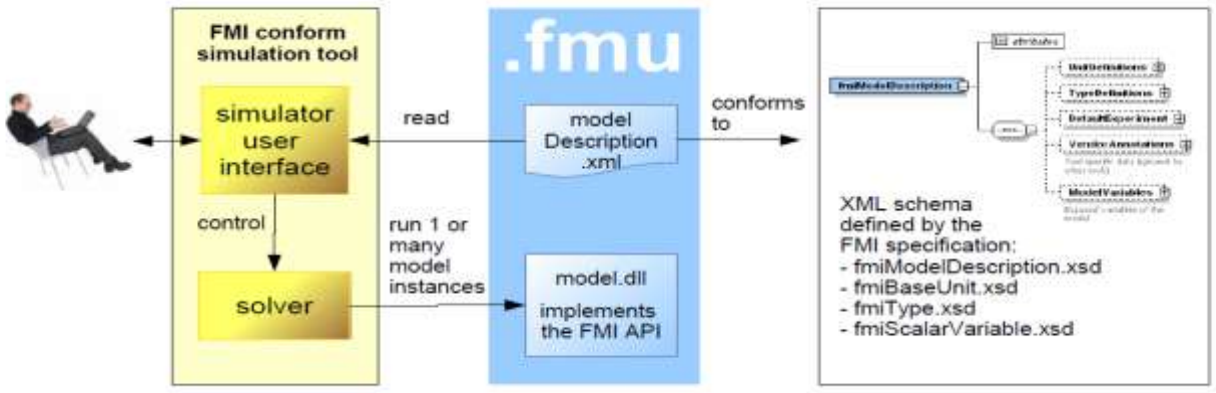

FMI conform simulation tool
simulator user interface
control
solver
read
run 1 or many model instances
.fmu
model Description .xml
model.dll implements the FMI API
conforms to
XML schema defined by the FMI specification:
- fmiModelDescription.xsd
- fmiBaseUnit.xsd
- fmiType.xsd
- fmiScalarVariable.xsd

Figure A3.2

Modeldescription.xml (which is located in the FMU.zip file is the FMI of the corresponding FMU) holds all the information about the FMU. It follows the FMI metamodel and FMI is the interface of FMU with the outer world. In other words, reaching an FMU is done by using its FMI. So from FMI, one can obtain all of the properties of the FMU. Because of this fact, FMI MetaModel and so only FMI's components are used for the mapping process. fmiModelDescription is the root element of FMI and every property of the FMU is hidden inside this box.

All the below Figures in this section of Appendix shows the mapping rules, in other words they show how we related the Modelisar model elements to EAST-ADL model elements. All the Figures have their explanations in themselves.

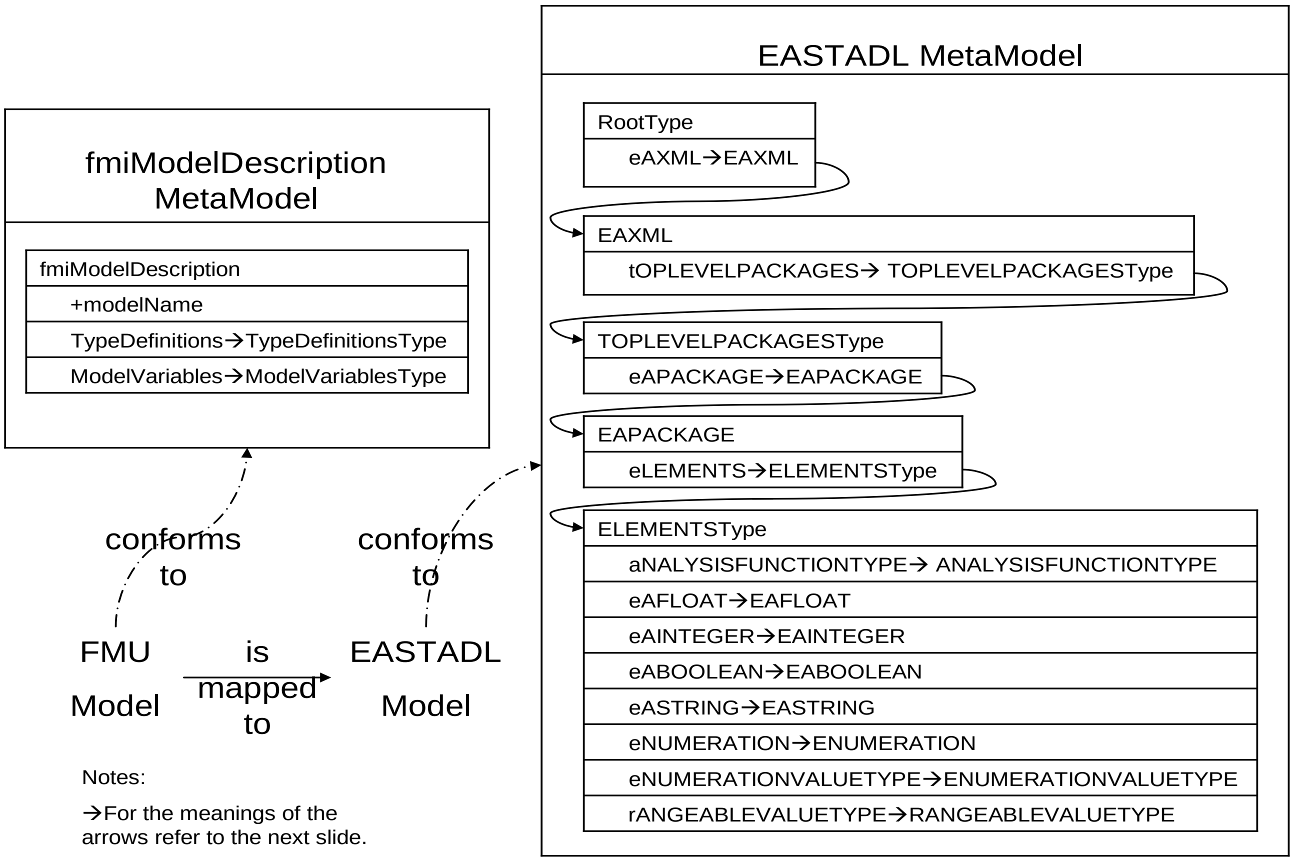
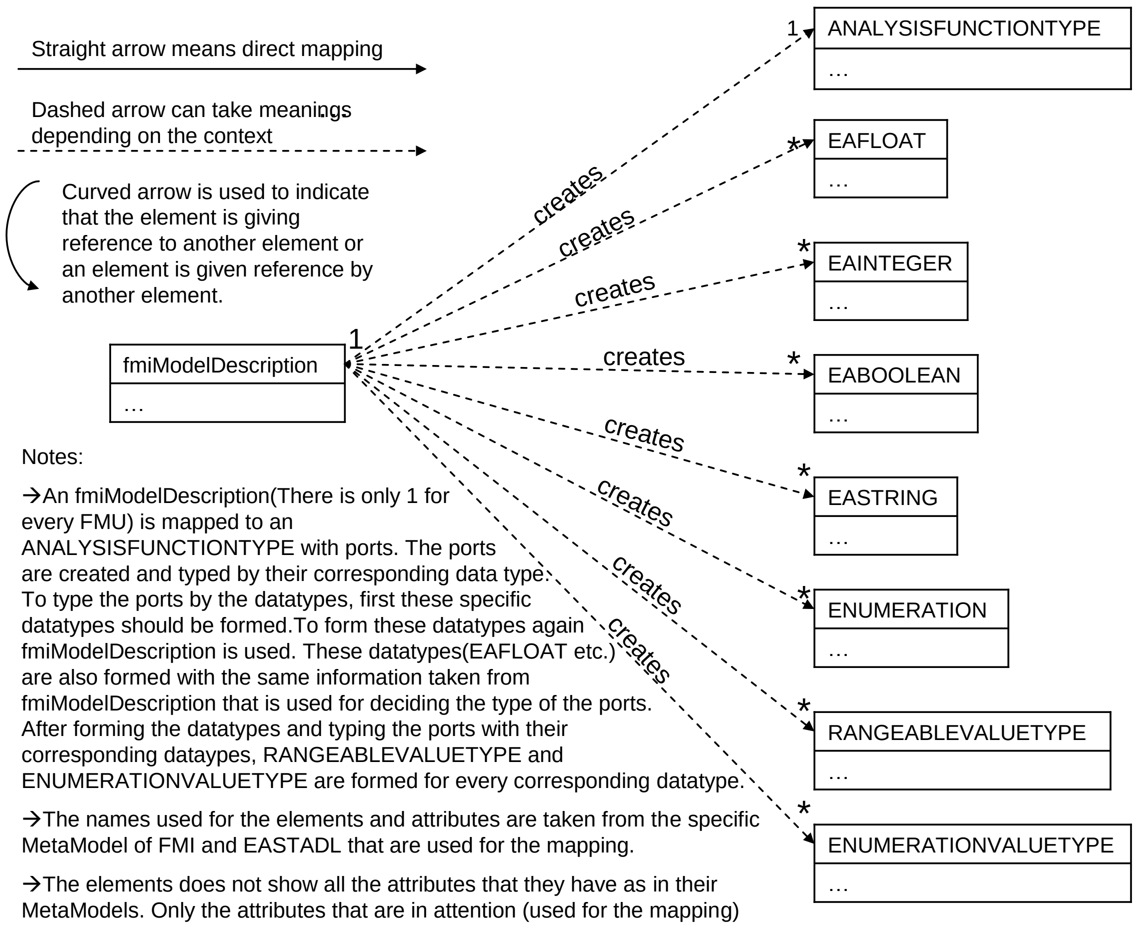

fmiModelDescription MetaModel
fmiModelDescription
+modelName
TypeDefinitions→TypeDefinitionsType
ModelVariables→ModelVariablesType
conforms to
conforms to
FMU Model
is mapped to
EASTADL Model
Notes:
→For the meanings of the arrows refer to the next slide.
EASTADL MetaModel
RootType
eAXML→EAXML
EAXML
tOPLEVELPACKAGES→ TOPLEVELPACKAGESType
TOPLEVELPACKAGESType
eAPACKAGE→EAPACKAGE
EAPACKAGE
eLEMENTS→ELEMENTSType
ELEMENTSType
aNALYSISFUNCTIONTYPE→ ANALYSISFUNCTIONTYPE
eAFLOAT→EAFLOAT
eAINTEGER→EAINTEGER
eABOOLEAN→EABOOLEAN
eASTRING→EASTRING
eNUMERATION→ENUMERATION
eNUMERATIONVALUETYPE→ENUMERATIONVALUETYPE
rANGEABLEVALUETYPE→RANGEABLEVALUETYPE

Figure A3.3


Straight arrow means direct mapping
Dashed arrow can take meanings depending on the context
Curved arrow is used to indicate that the element is giving reference to another element or an element is given reference by another element.
fmiModelDescription
…
1
creates
creates
creates
creates
creates
creates
creates
creates
1
ANALYSISFUNCTIONTYPE
…
*
EAFLOAT
…
*
EAINTEGER
…
*
EABOOLEAN
…
*
EASTRING
…
*
ENUMERATION
…
*
RANGEABLEVALUETYPE
…
*
ENUMERATIONVALUETYPE
…

Notes:

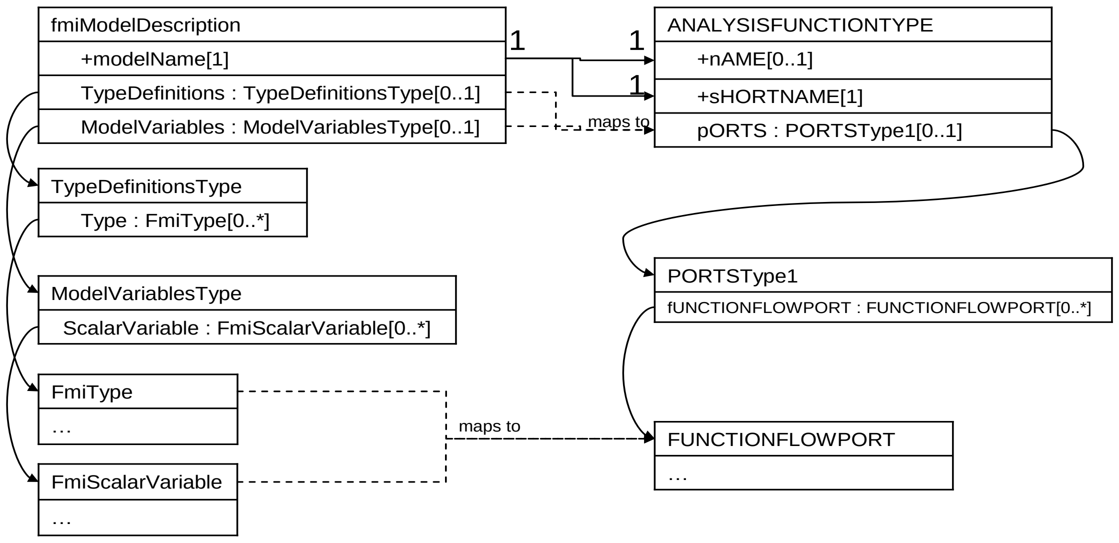
→An fmiModelDescription(There is only 1 for every FMU) is mapped to an ANALYSISFUNCTIONTYPE with ports. The ports are created and typed by their corresponding data type. To type the ports by the datatypes, first these specific datatypes should be formed.To form these datatypes again fmiModelDescription is used. These datatypes(EAFLOAT etc.) are also formed with the same information taken from fmiModelDescription that is used for deciding the type of the ports. After forming the datatypes and typing the ports with their corresponding dataypes, RANGEABLEVALUETYPE and ENUMERATIONVALUETYPE are formed for every corresponding datatype.

→The names used for the elements and attributes are taken from the specific MetaModel of FMI and EASTADL that are used for the mapping.

→The elements does not show all the attributes that they have as in their MetaModels. Only the attributes that are in attention (used for the mapping) are shown.

Figure A3.4

➔"…" notation that is also used in this table either means that there are more attributes of the elements used for the mapping but they are not used in this mapping table so they will take place in the others **OR** the element shown is given reference by its root element in the previous pages. This notation is used in the same way for all of the document.



Notes:

➔In default FmiScalarVariables are used for mapping to FUNCTIONFLOWPORT but FmiScalarVariables can give reference to FmiTypes. In this case if the values missing in the FmiScalarVariable are avaiable in FmiType, they are used otherwise always FmiScalarVariables' values are used for mapping. So if there is no reference given from FmiScalarVariable to an FmiType, directly FmiScalarVariable is used in mapping and FmiType is not taken into consideration otherwise for the values that are not available in FmiScalarVariable will be taken from FmiType if they are available in it.

Figure A3.5

Notes:

➔For instance, RealType and RealTypeType are nearly identical in structure. They have the same attributes. An EAFLOAT is created either by the RealType or RealTypeType. In default, it is created by RealType but if RealType gives reference to an FmiType which should hold a RealTypeType by using its attribute 'declaredType' then still the information in the RealType is used for the conversion but this time if there is an additional information in the referenced FmiType's RealTypeType, this additional information is used too.

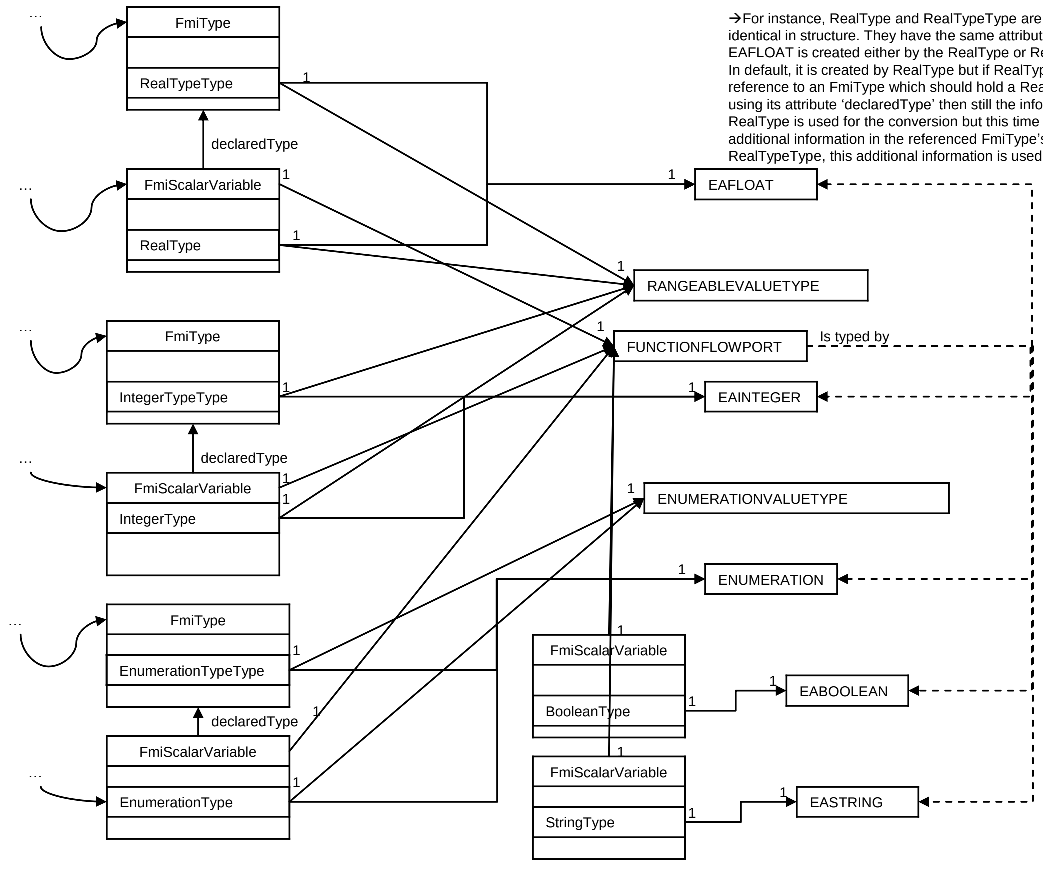
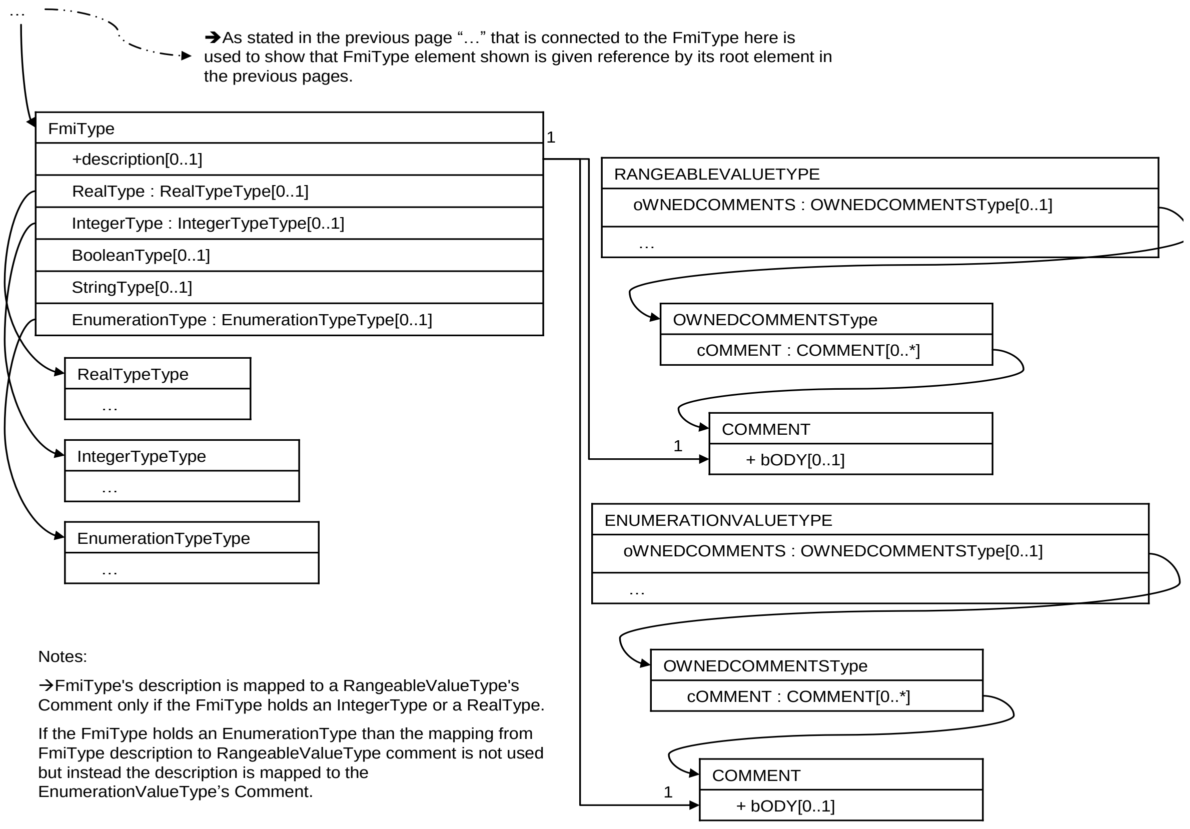


Figure A3.6



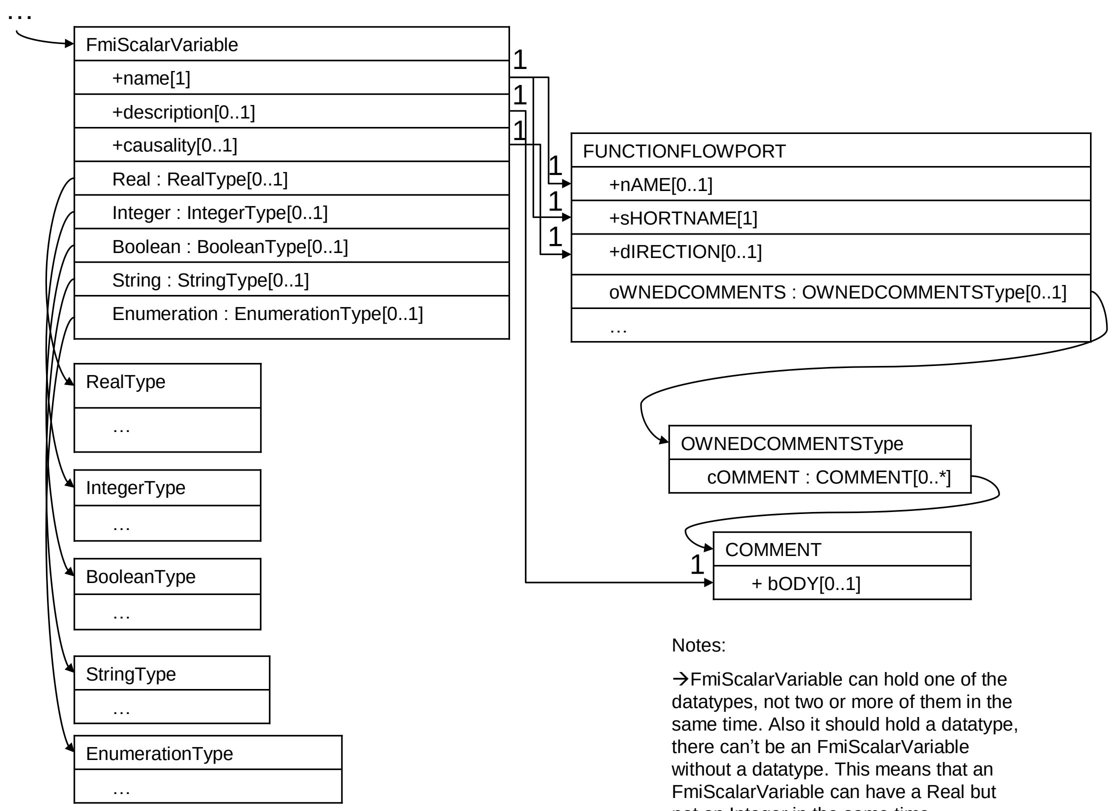
➔As stated in the previous page "…" that is connected to the FmiType here is used to show that FmiType element shown is given reference by its root element in the previous pages.

Notes:

→FmiType's description is mapped to a RangeableValueType's Comment only if the FmiType holds an IntegerType or a RealType.

If the FmiType holds an EnumerationType than the mapping from FmiType description to RangeableValueType comment is not used but instead the description is mapped to the EnumerationValueType's Comment.

Figure A3.7



Notes:

→FmiScalarVariable can hold one of the datatypes, not two or more of them in the same time. Also it should hold a datatype, there can't be an FmiScalarVariable without a datatype. This means that an FmiScalarVariable can have a Real but not an Integer in the same time.

Figure A3.8

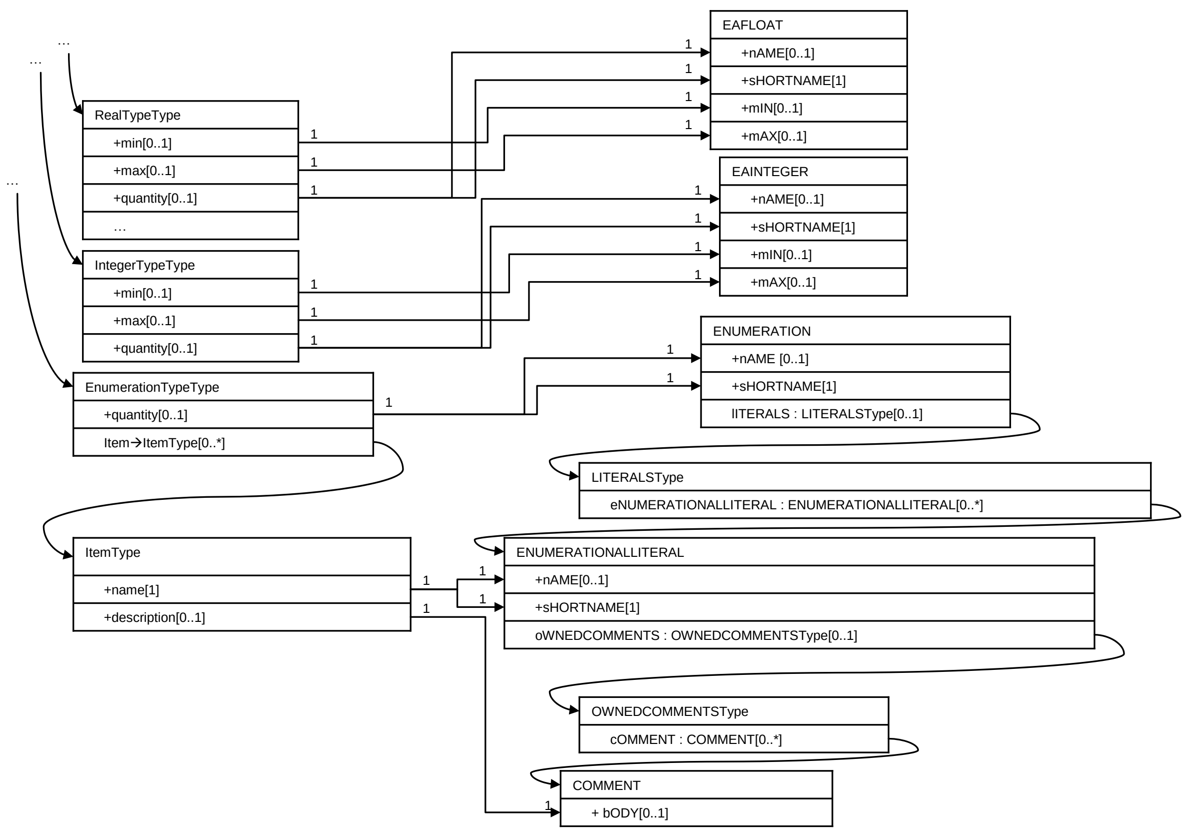


Figure A3.9

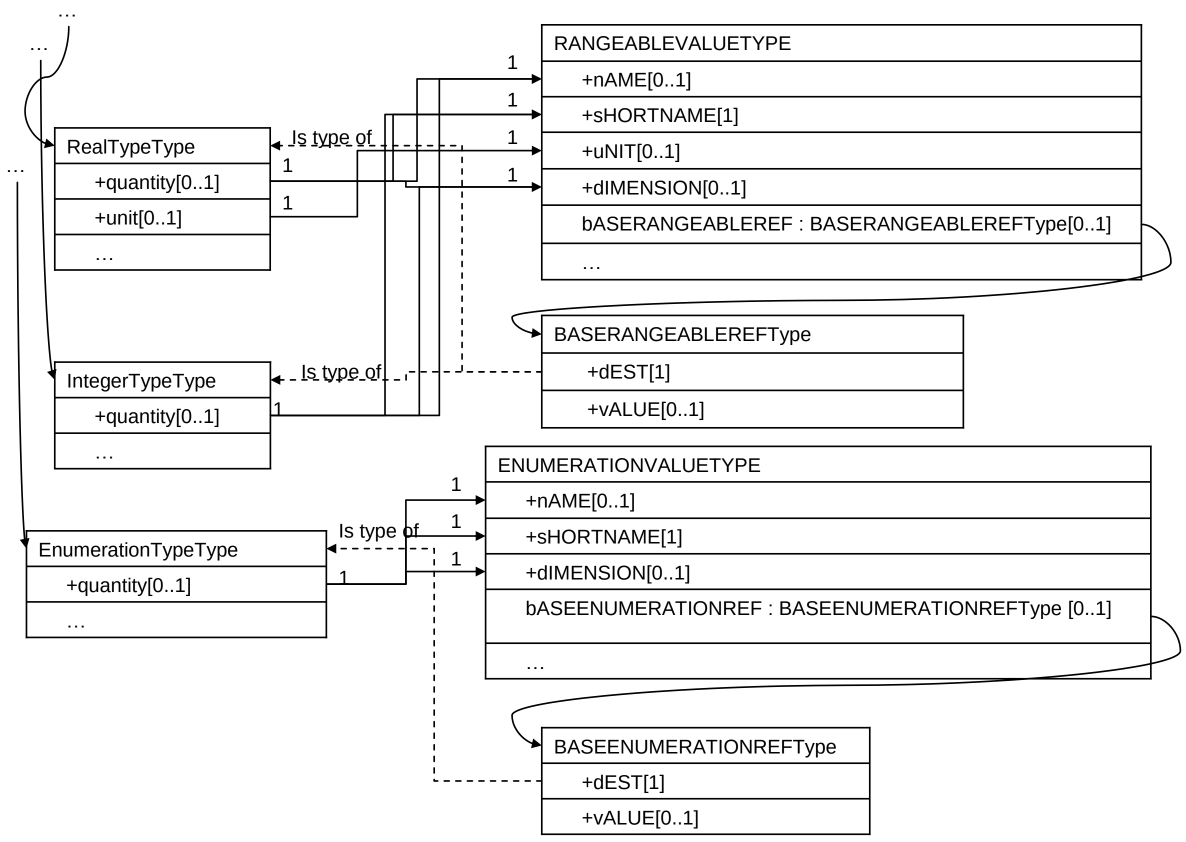
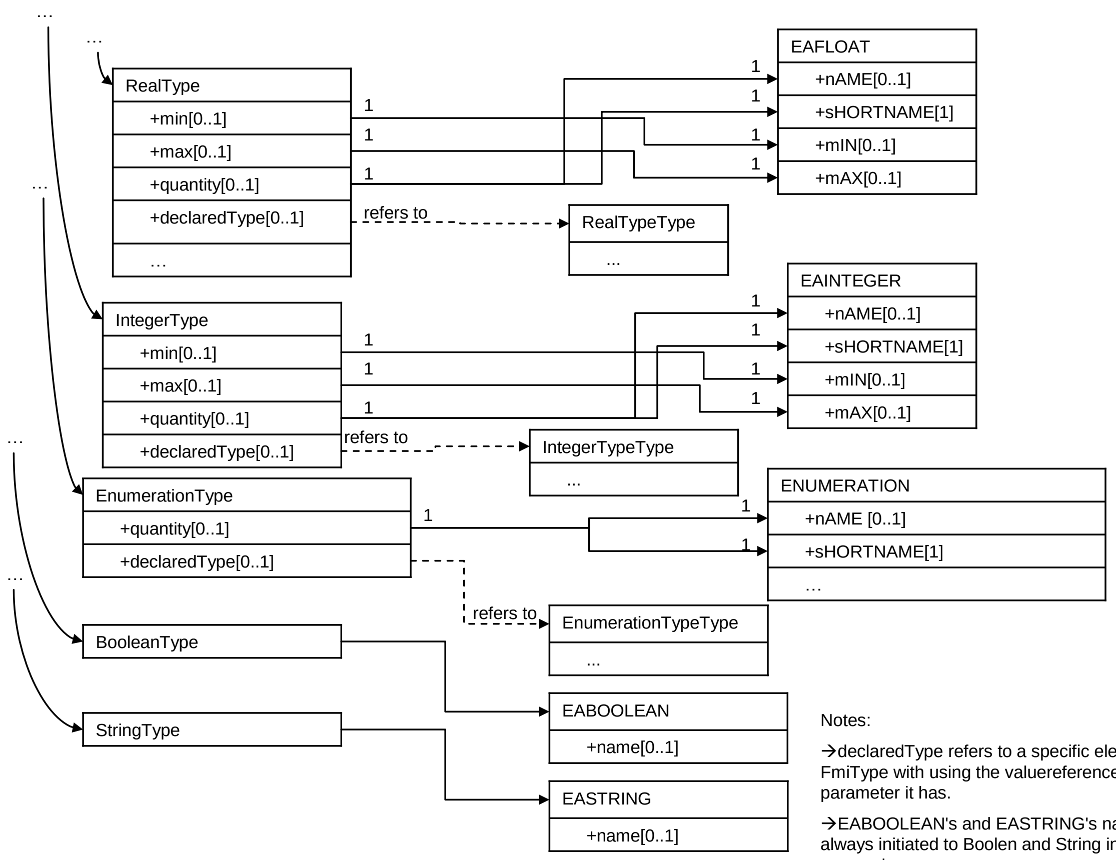


Figure A3.10

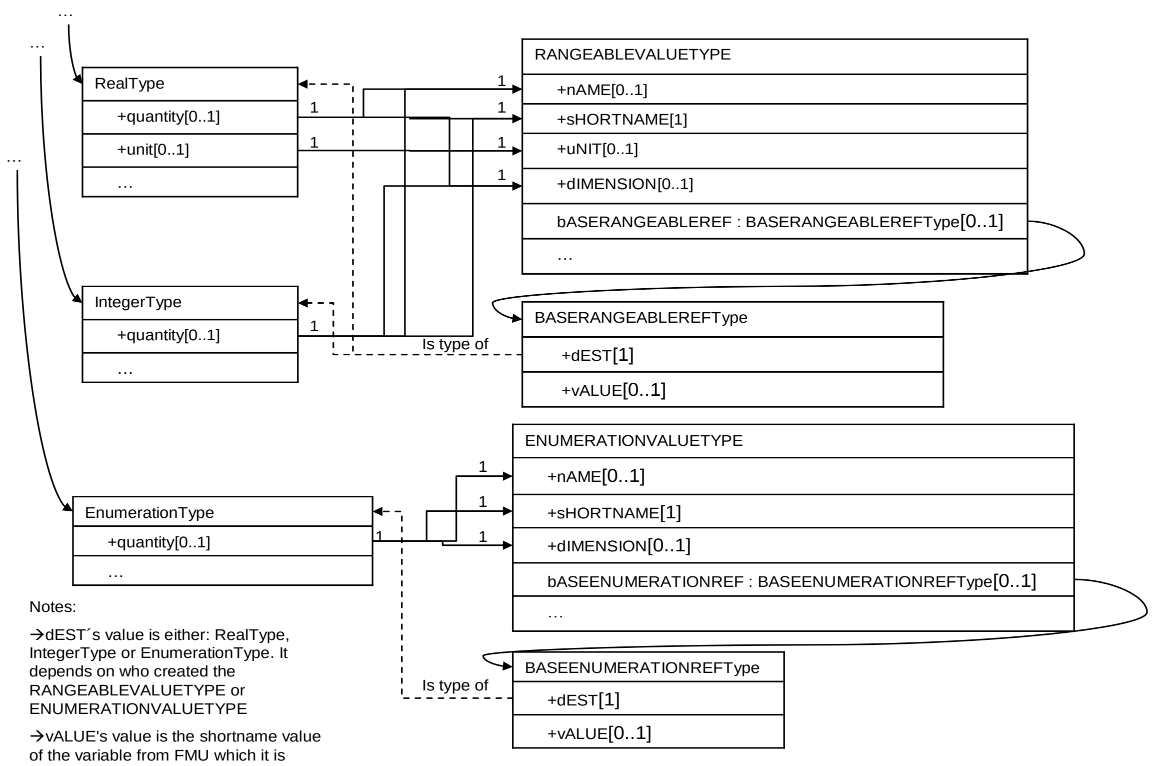


Notes:

→declaredType refers to a specific element in FmiType with using the valuereference parameter it has.

→EABOOLEAN's and EASTRING's name is always initiated to Boolen and String in the conversion.

Figure A3.11

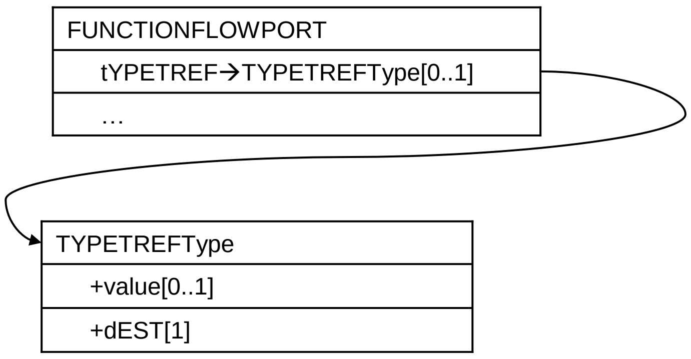


Notes:

→dEST´s value is either: RealType, IntegerType or EnumerationType. It depends on who created the RANGEABLEVALUETYPE or ENUMERATIONVALUETYPE

→vALUE's value is the shortname value of the variable from FMU which it is created.

Figure A3.12



Notes:

→value refers to a specific data type element with using the valuereference parameter it has. ValueReference here is a ShortName of the referenced element. The referenced element is the element which it is typed by.

→dEST´s value is either: EAFLOAT, EAINTEGER, EABOOLEAN, EASTRING or ENUMERATION. It depends on which the FUNCTIONFLOWPORT is created from. FUNCTIONFLOWPORT is created from either a type in FmiType or FmiScalarVariabl as shown in the above slides. And these types is mapped to one of the data types as also shown in the above slides. So dEST takes the type of this data type.

Figure A3.13

Future Work for the Mapping Rules

• All the elements that do not have a direct mapping from FMU to EAST-ADL will be mapped to a User Attributable Element in EAST-ADL, so the user will have still all the information of the FMU. This feature can prevent the loss of information and besides it can be used for different purposes such as EAST-ADL→FMU conversion(this conversion can be useful in some cases such as that, an FMU is wanted to be created with the additional elements in an EAST-ADL model. Another example can be that the FMU used in the conversion is not in hand and it is wanted to be generated again.).
• FMU can be put as the FunctionBehavior of the obtained EAST-ADL model.

4. ATL code

```
-- @path FmiModelDescription=/FMU2EA7/FmiModelDescription.ecore

-- @path EAST=/FMU2EA7/adlrt.ecore


module FMU2EA7;


create OUT : EAST from IN : FmiModelDescription;


helper context FmiModelDescription!FmiScalarVariable def: type : String =

	if not self."Real".oclIsUndefined() then

		'EAFLOAT'

	else

		if not self."Integer".oclIsUndefined() then

			'EAINTEGER'

		else

			if not self."Boolean".oclIsUndefined() then

				'EABOOLEAN'

			else

				if not self."Enumeration".oclIsUndefined() then

					'ENUMERATION'

				else

					'EASTRING'

			endif

		endif

	endif

endif;


helper context FmiModelDescription!FmiScalarVariable def: portReal: Boolean =

	if self.causality =#input and not self."Real".oclIsUndefined() then
```

```
          if not self."Real".declaredType.oclIsUndefined() then

               false

          else

               true

          endif

     else

          if self.causality =#output and not self."Real".oclIsUndefined()
then

               if not self."Real".declaredType.oclIsUndefined() then

                    false

               else

                    true

               endif

          else

               false

          endif

endif;


helper context FmiModelDescription!FmiScalarVariable def: portRealDeclared:
Boolean =

     if self.causality =#input and not self."Real".oclIsUndefined() then

          if not self."Real".declaredType.oclIsUndefined() then

               true

          else

               false

          endif

     else

          if self.causality =#output and not self."Real".oclIsUndefined()
then

               if not self."Real".declaredType.oclIsUndefined() then

                    true
```

```
                else
                    false
                endif
            else
                false
            endif
endif;


helper context FmiModelDescription!FmiScalarVariable def: portInteger:
Boolean =
    if self.causality =#input and not self."Integer".oclIsUndefined() then
        if not self."Integer".declaredType.oclIsUndefined() then
            false
        else
            true
        endif
    else
        if self.causality =#output and not self."Integer".oclIsUndefined()
then
            if not self."Integer".declaredType.oclIsUndefined() then
                false
            else
                true
            endif
        else
            false
        endif
endif;


helper context FmiModelDescription!FmiScalarVariable def:
portIntegerDeclared: Boolean =
     if self.causality =#input and not self."Integer".oclIsUndefined() then

          if not self."Integer".declaredType.oclIsUndefined() then

               true

          else

               false

          endif

     else

          if self.causality =#output and not self."Integer".oclIsUndefined() 
then

               if not self."Integer".declaredType.oclIsUndefined() then

                    true

               else

                    false

               endif

          else

               false

          endif

endif;


helper context FmiModelDescription!FmiScalarVariable def: portEnumeration:
Boolean =

     if self.causality =#input and not self.Enumeration.oclIsUndefined() then

          if not self.Enumeration.declaredType.oclIsUndefined() then

               false

          else

               true

          endif

     else

          if self.causality =#output and not 
self.Enumeration.oclIsUndefined() then

               if not self.Enumeration.declaredType.oclIsUndefined() then
```

```
                  false
              else
                  true
              endif
          else
              false
          endif
endif;


helper context FmiModelDescription!FmiScalarVariable def:
portEnumerationDeclared: Boolean =
      if self.causality =#input and not self.Enumeration.oclIsUndefined()  then
          if not self.Enumeration.declaredType.oclIsUndefined()  then
              true
          else
              false
          endif
      else
          if self.causality =#output and not
self.Enumeration.oclIsUndefined()  then
              if not self.Enumeration.declaredType.oclIsUndefined()  then
                  true
              else
                  false
              endif
          else
              false
          endif
endif;
```

```
helper context FmiModelDescription!FmiScalarVariable def: portType : String =
     if self.causality =#input then
          'IN'
     else
          if self.causality =#output then
               'OUT'
          else
               'INOUT'
          endif
endif;


rule fmiModelDescription2EAXML {
     from
          s : FmiModelDescription!fmiModelDescription
     to
          t: EAST!RootType(
               eAXML <- EAXML
          ),
          EAXML : EAST!EAXML(
               tOPLEVELPACKAGES <- TOPLEVELPACKAGESType
          ),
          TOPLEVELPACKAGESType: EAST!TOPLEVELPACKAGESType(
               eAPACKAGE <- EAPACKAGE
          ),
          EAPACKAGE: EAST!EAPACKAGE(
               eLEMENTS <- ELEMENTSTYPE
          ),
          ELEMENTSTYPE: EAST!ELEMENTSType(
               aNALYSISFUNCTIONTYPE <- ANALYSISFUNCTIONTYPE
```

```
            ),

            ANALYSISFUNCTIONTYPE: EAST!ANALYSISFUNCTIONTYPE(

                  nAME <- s.modelIdentifier,

                  sHORTNAME <- s.modelIdentifier,

                  pORTS <- PORTSType1

            ),

            PORTSType1: EAST!PORTSType1(

                  fUNCTIONFLOWPORT <-
FmiModelDescription!FmiScalarVariable.allInstances() ->select(c | c.portReal
or c.portInteger or c.portEnumeration or c.portRealDeclared or c.portBoolean
or c.portString or c.portIntegerDeclared or c.portEnumerationDeclared)

            )

}

rule FmiScalarVariable2EAREAL {

      from

            s : FmiModelDescription!FmiScalarVariable(s.portReal)

      to

            FUNCTIONFLOWPORT : EAST!FUNCTIONFLOWPORT(

                  nAME <- s.name,

                  sHORTNAME <- s.name,

                  dIRECTION <- s.portType,

                  oWNEDCOMMENTS <- OWNEDCOMMENTSType,

                  tYPETREF <- TYPETREFTYPE

            ),

            OWNEDCOMMENTSType: EAST!OWNEDCOMMENTSType(

                  cOMMENT<-COMMENT

            ),

            COMMENT: EAST!COMMENT(

                  bODY<- s.description

            ),

            TYPETREFTYPE: EAST!TYPETREFType(
                  value <- s.Real.quantity,

                  dEST <- s.type

            ),

            EAFLOAT: EAST!EAFLOAT (

                  nAME <- s.Real.quantity,

                  sHORTNAME <- s.Real.quantity,

                  mIN <- s.Real.min,

                  mAX <- s.Real.max

            ),

            RANGEABLEVALUETYPE: EAST!RANGEABLEVALUETYPE(

                  nAME <- s.Real.quantity,

                  sHORTNAME <- s.Real.quantity,

                  uNIT <- s.Real.unit,

                  dIMENSION <- s.Real.unit,

                  bASERANGEABLEREF <- BASERANGEABLEREF

            ),

            BASERANGEABLEREF: EAST!BASERANGEABLEREFType(

                  value <- s.Real.quantity,

                  dEST <- s.type

            )

}


helper context FmiModelDescription!FmiScalarVariable def:
nameDeclared(declared:String): String =

      --Take all FmiTypes and search for declared

      if not self.name.oclIsUndefined() then

            self.name

      else

            let fmiSeq : Sequence(OclAny) =
FmiModelDescription!FmiType.allInstances()->asSequence() in
```

```
                if fmiSeq->one(e|e.name=declared) then

                    fmiSeq->select(e|e.name=declared)->collect(e|e.name)-
>first()

                else

                    'No matching declared type found or More than one
FmiType has the same name'

                endif

endif;


helper context FmiModelDescription!FmiScalarVariable def:
descriptionDeclared(declared:String): String =

     let fmiSeq : Sequence(OclAny) =
FmiModelDescription!FmiType.allInstances()->asSequence() in

          if fmiSeq->one(e|e.name=declared) then

               if not fmiSeq->select(e|e.name=declared)-
>collect(e|e.description).oclIsUndefined() then

                    fmiSeq->select(e|e.name=declared)-
>collect(e|e.description)->first()

               else

                    'No description found'

               endif

          else

               'No matching declared type found or More than one FmiType has
the same name'

          endif;


helper context FmiModelDescription!FmiScalarVariable def:
quantityDeclared(declared:String): String =

     if not self."Real".quantity.oclIsUndefined() then

          self."Real".quantity

     else

          let fmiSeq : Sequence(OclAny) =
FmiModelDescription!FmiType.allInstances()->asSequence() in

               if fmiSeq->one(e|e.name=declared) then
```

```
                    if not fmiSeq->select(e|e.name=declared)->collect(e|e."RealType".quantity).oclIsUndefined() then

                         fmiSeq->select(e|e.name=declared)->collect(e|e."RealType".quantity)->first()

                    else

                         'No description found'

                    endif

               else

                    'No matching declared type found or More than one FmiType has the same name'

               endif

endif;


helper context FmiModelDescription!FmiScalarVariable def: minDeclared(declared:String): Real =

     if (self."Real".min > 0.0) or (self."Real".min < 0.0) then

          self."Real".min

     else

          let fmiSeq : Sequence(OclAny) = FmiModelDescription!FmiType.allInstances()->asSequence() in

               if fmiSeq->one(e|e.name=declared) then

                    fmiSeq->select(e|e.name=declared)->collect(e|e."RealType".min)->first()

               else

                    'No matching declared type found or More than one FmiType has the same name'

               endif

endif;


helper context FmiModelDescription!FmiScalarVariable def: maxDeclared(declared:String): Real =

     if (self."Real".max > 0.0) or (self."Real".max < 0.0) then

          self."Real".max
     else

          let fmiSeq : Sequence(OclAny) =
FmiModelDescription!FmiType.allInstances()->asSequence() in

               if fmiSeq->one(e|e.name=declared) then

                    fmiSeq->select(e|e.name=declared)-
>collect(e|e."RealType".max)->first()

               else

                    'No matching declared type found or More than one
FmiType has the same name'

               endif

endif;


helper context FmiModelDescription!FmiScalarVariable def:
unitDeclared(declared:String): Real =

     if not self."Real".unit.oclIsUndefined() then

          self."Real".unit

     else

          let fmiSeq : Sequence(OclAny) =
FmiModelDescription!FmiType.allInstances()->asSequence() in

               if fmiSeq->one(e|e.name=declared) then

                    if not fmiSeq->select(e|e.name=declared)-
>collect(e|e."RealType".unit).oclIsUndefined() then

                         fmiSeq->select(e|e.name=declared)-
>collect(e|e."RealType".unit)->first()

                    else

                         'No description found'

                    endif

               else

                    'No matching declared type found or More than one
FmiType has the same name'

               endif

endif;
```

```
rule FmiScalarVariable2EAREALDeclared {

     from

          s : FmiModelDescription!FmiScalarVariable(s.portRealDeclared)

     to

          FUNCTIONFLOWPORT : EAST!FUNCTIONFLOWPORT(

               nAME <- s.name,

               sHORTNAME <- s.name,

               dIRECTION <- s.portType,

               oWNEDCOMMENTS <- OWNEDCOMMENTSType,

               tYPETREF <- TYPETREFTYPE

          ),

          OWNEDCOMMENTSType: EAST!OWNEDCOMMENTSType(

               cOMMENT<-COMMENT

          ),

          COMMENT: EAST!COMMENT(

               bODY<- s.description

          ),

          TYPETREFTYPE: EAST!TYPETREFType(

               value <- s.quantityDeclared(s.Real.declaredType),

               dEST <- s.type

          ),

          EAFLOAT: EAST!EAFLOAT (

               nAME <- s.quantityDeclared(s.Real.declaredType),

               sHORTNAME <- s.quantityDeclared(s.Real.declaredType),

               mIN <- s.minDeclared(s.Real.declaredType),

               mAX <- s.maxDeclared(s.Real.declaredType)

          ),

          RANGEABLEVALUETYPE: EAST!RANGEABLEVALUETYPE(

               nAME <- s.quantityDeclared(s.Real.declaredType),
                sHORTNAME <- s.quantityDeclared(s.Real.declaredType),

                uNIT <- s.unitDeclared(s.Real.declaredType),

                dIMENSION <- s.unitDeclared(s.Real.declaredType),

                oWNEDCOMMENTS <- OWNEDCOMMENTSTypeRangeable,

                bASERANGEABLEREF <- BASERANGEABLEREF

        ),

        OWNEDCOMMENTSTypeRangeable: EAST!OWNEDCOMMENTSType(

                cOMMENT<-COMMENTRangeable

        ),

        COMMENTRangeable: EAST!COMMENT(

                bODY<- s.descriptionDeclared(s.Real.declaredType)

        ),

        BASERANGEABLEREF: EAST!BASERANGEABLEREFType(

                value <- s.quantityDeclared(s.Real.declaredType),

                dEST <- s.type

        )

}


rule FmiScalarVariable2EAINTEGER {

        from

                t : FmiModelDescription!FmiScalarVariable (t.portInteger)

        to

                FUNCTIONFLOWPORT : EAST!FUNCTIONFLOWPORT(

                        nAME <- t.name,

                        sHORTNAME <- t.name,

                        dIRECTION <- t.portType,

                        oWNEDCOMMENTS <- OWNEDCOMMENTSType,

                        tYPETREF <- TYPETREFTYPE

                ),
          OWNEDCOMMENTSType: EAST!OWNEDCOMMENTSType(

               cOMMENT<-COMMENT

          ),

          COMMENT: EAST!COMMENT(

               bODY<- t.description

          ),

          TYPETREFTYPE: EAST!TYPETREFType(

               value <- t.Integer.quantity,

               dEST <- t.type

          ),

          EAINTEGER: EAST!EAINTEGER (

               nAME <- t.Integer.quantity,

               mIN <- t.Integer.min.toString(),

               mAX <- t.Integer.max.toString(),

               sHORTNAME <- t.Integer.quantity

          ),

          RANGEABLEVALUETYPE: EAST!RANGEABLEVALUETYPE(

               nAME <- t.Integer.quantity,

               sHORTNAME <- t.Integer.quantity,

               dIMENSION <- t.Integer.quantity,

               bASERANGEABLEREF <- BASERANGEABLEREF

          ),

          BASERANGEABLEREF: EAST!BASERANGEABLEREFType(

               value <- t.Integer.quantity,

               dEST <- t.type

          )

}


helper context FmiModelDescription!FmiScalarVariable def:
quantityDeclaredInteger(declared:String): String =
     if not self."Integer".quantity.oclIsUndefined() then

          self."Integer".quantity

     else

          let fmiSeq : Sequence(OclAny) =
FmiModelDescription!FmiType.allInstances()->asSequence() in

               if fmiSeq->one(e|e.name=declared) then

                    if not fmiSeq->select(e|e.name=declared)-
>collect(e|e."IntegerType".quantity).oclIsUndefined() then

                         fmiSeq->select(e|e.name=declared)-
>collect(e|e."IntegerType".quantity)->first()

                    else

                         'No description found'

                    endif

               else

                    'No matching declared type found or More than one
FmiType has the same name'

               endif

endif;


helper context FmiModelDescription!FmiScalarVariable def:
minDeclaredInteger(declared:Integer): String =

     if (self.Integer.min > 0.0) or (self.Integer.min < 0.0) then

          self.Integer.min

     else

          let fmiSeq : Sequence(OclAny) =
FmiModelDescription!FmiType.allInstances()->asSequence() in

               if fmiSeq->one(e|e.name=declared) then

                    fmiSeq->select(e|e.name=declared)-
>collect(e|e."IntegerType".min)->first()

               else

                    'No matching declared type found or More than one
FmiType has the same name'

               endif
```

```
endif;


helper context FmiModelDescription!FmiScalarVariable def:
maxDeclaredInteger(declared:Integer): String =

      if (self.Integer.max > 0.0) or (self.Integer.max < 0.0) then

            self.Integer.max

      else

            let fmiSeq : Sequence(OclAny) =
FmiModelDescription!FmiType.allInstances()->asSequence() in

                  if fmiSeq->one(e|e.name=declared) then

                        fmiSeq->select(e|e.name=declared)-
>collect(e|e."IntegerType".max)->first()

                  else

                        'No matching declared type found or More than one
FmiType has the same name'

                  endif

endif;


rule FmiScalarVariable2EAINTEGERDeclared {

      from

            t : FmiModelDescription!FmiScalarVariable (t.portIntegerDeclared)

      to

            FUNCTIONFLOWPORT : EAST!FUNCTIONFLOWPORT(

                  nAME <- t.name,

                  sHORTNAME <- t.name,

                  dIRECTION <- t.portType,

                  oWNEDCOMMENTS <- OWNEDCOMMENTSType,

                  tYPETREF <- TYPETREFTYPE

            ),

            OWNEDCOMMENTSType: EAST!OWNEDCOMMENTSType(

                  cOMMENT<-COMMENT
```

```
          ),

          COMMENT: EAST!COMMENT(

               bODY<- t.description

          ),

          TYPETREFTYPE: EAST!TYPETREFType(

               value <- t.quantityDeclaredInteger(t.Integer.declaredType),

               dEST <- t.type

          ),

          EAINTEGER: EAST!EAINTEGER (

               nAME <- t.quantityDeclaredInteger(t.Integer.declaredType),

               mIN <-
t.minDeclaredInteger(t.Integer.declaredType).toString(),

               mAX <-
t.maxDeclaredInteger(t.Integer.declaredType).toString(),

               sHORTNAME <-
t.quantityDeclaredInteger(t.Integer.declaredType)

          ),

          RANGEABLEVALUETYPE: EAST!RANGEABLEVALUETYPE(

               nAME <- t.quantityDeclaredInteger(t.Integer.declaredType),

               sHORTNAME <-
t.quantityDeclaredInteger(t.Integer.declaredType),

               dIMENSION <-
t.quantityDeclaredInteger(t.Integer.declaredType),

               bASERANGEABLEREF <- BASERANGEABLEREF

          ),

          BASERANGEABLEREF: EAST!BASERANGEABLEREFType(

               value <- t.quantityDeclaredInteger(t.Integer.declaredType),

               dEST <- t.type

          )

}

rule FmiScalarVariable2ENUMERATION {

     from
```

```
        t : FmiModelDescription!FmiScalarVariable (t.portEnumeration)
to
        FUNCTIONFLOWPORT : EAST!FUNCTIONFLOWPORT(
                nAME <- t.name,
                sHORTNAME <- t.name,
                dIRECTION <- t.portType,
                oWNEDCOMMENTS <- OWNEDCOMMENTSType,
                tYPETREF <- TYPETREFTYPE
        ),
        OWNEDCOMMENTSType: EAST!OWNEDCOMMENTSType(
                cOMMENT<-COMMENT
        ),
        COMMENT: EAST!COMMENT(
                bODY<- t.description
        ),
        TYPETREFTYPE: EAST!TYPETREFType(
                value <- t.Enumeration.quantity,
                dEST <- t.type
        ),
        ENUMERATION: EAST!ENUMERATION (
                nAME <- t.Enumeration.quantity,
                sHORTNAME <- t.Enumeration.quantity
        ),
        ENUMERATIONVALUETYPE: EAST!ENUMERATIONVALUETYPE(
                nAME <- t.Enumeration.quantity,
                sHORTNAME <- t.Enumeration.quantity,
                dIMENSION <- t.Enumeration.quantity,
                bASEENUMERATIONREF <- BASEENUMERATIONREF
        ),
          BASEENUMERATIONREF: EAST!BASEENUMERATIONREFType(

               value <- t.Enumeration.quantity,

               dEST <- t.type

          )

}


helper context FmiModelDescription!FmiScalarVariable def:
quantityDeclaredEnumeration(declared:String): String =

     if not self.Enumeration.quantity.oclIsUndefined() then

          self.Enumeration.quantity

     else

          let fmiSeq : Sequence(OclAny) =
FmiModelDescription!FmiType.allInstances()->asSequence() in

               if fmiSeq->one(e|e.name=declared) then

                    if not fmiSeq->select(e|e.name=declared)-
>collect(e|e.EnumerationType.quantity).oclIsUndefined() then

                         fmiSeq->select(e|e.name=declared)-
>collect(e|e.EnumerationType.quantity)->first()

                    else

                         'No description found'

                    endif

               else

                    'No matching declared type found or More than one
FmiType has the same name'

               endif

endif;


helper context FmiModelDescription!FmiScalarVariable def:
ItemType(declared:String): Sequence(OclAny) =

     --Take all FmiTypes and search for declared

     let fmiSeq : Sequence(OclAny) =
FmiModelDescription!FmiType.allInstances()->asSequence() in

          if fmiSeq->one(e|e.name=declared) then
```

```
                fmiSeq->select(e|e.name=declared)-
>collect(e|e.EnumerationType.Item)

           else

                'No matching declared type found or More than one FmiType has
the same name'

           endif;


rule FmiScalarVariable2ENUMERATIONDeclared {

     from

           t : FmiModelDescription!FmiScalarVariable
(t.portEnumerationDeclared)

     using{

           EnumerationTypeType: Sequence(OclAny) =
t.ItemType(t.Enumeration.declaredType)->first();

           literals: Sequence(OclAny)= EnumerationTypeType-
>collect(e|e.name);

     }

     to

           FUNCTIONFLOWPORT : EAST!FUNCTIONFLOWPORT(

                nAME <- t.name,

                sHORTNAME <- t.name,

                dIRECTION <- t.portType,

                oWNEDCOMMENTS <- OWNEDCOMMENTSType,

                tYPETREF <- TYPETREFTYPE

           ),

           OWNEDCOMMENTSType: EAST!OWNEDCOMMENTSType(

                cOMMENT<-COMMENT

           ),

           COMMENT: EAST!COMMENT(

                bODY<- t.description

           ),

           TYPETREFTYPE: EAST!TYPETREFType(
                value <-
t.quantityDeclaredEnumeration(t.Enumeration.declaredType),

                dEST <- t.type

            ),

            ENUMERATION: EAST!ENUMERATION (

                nAME <-
t.quantityDeclaredEnumeration(t.Enumeration.declaredType),

                sHORTNAME <-
t.quantityDeclaredEnumeration(t.Enumeration.declaredType),

                lITERALS <- LITERALSType

            ),

            LITERALSType: EAST!LITERALSType(

                eNUMERATIONLITERAL <- ENUMERATIONLITERAL

            ),

            ENUMERATIONLITERAL: distinct EAST!ENUMERATIONLITERAL foreach(e in
EnumerationTypeType)(

                nAME <-e.name,

                sHORTNAME <-e.name,

                oWNEDCOMMENTS <- OWNEDCOMMENTSTypeEnum

            ),

            OWNEDCOMMENTSTypeEnum: distinct EAST!OWNEDCOMMENTSType foreach(e
in EnumerationTypeType)(

                cOMMENT<-COMMENTEnum

            ),

            COMMENTEnum: distinct EAST!COMMENT foreach(e in
EnumerationTypeType)(

                bODY<- e.description

            ),

            ENUMERATIONVALUETYPE: EAST!ENUMERATIONVALUETYPE(

                nAME <-
t.quantityDeclaredEnumeration(t.Enumeration.declaredType),

                sHORTNAME <-
t.quantityDeclaredEnumeration(t.Enumeration.declaredType),
                dIMENSION <-
t.quantityDeclaredEnumeration(t.Enumeration.declaredType),

                bASEENUMERATIONREF <- BASEENUMERATIONREF

           ),

           BASEENUMERATIONREF: EAST!BASEENUMERATIONREFType(

                value <-
t.quantityDeclaredEnumeration(t.Enumeration.declaredType),

                dEST <- t.type

           )

}


helper context FmiModelDescription!FmiScalarVariable def: portBoolean:
Boolean =

     if self.causality =#input and not self."Boolean".oclIsUndefined() then

           true

     else

           if self.causality =#output and not self."Boolean".oclIsUndefined()
then

                true

           else

                false

           endif

endif;


rule FmiScalarVariable2EABOOLEAN {

     from

           s : FmiModelDescription!FmiScalarVariable(s.portBoolean)

     to

           FUNCTIONFLOWPORT : EAST!FUNCTIONFLOWPORT(

                nAME <- s.name,

                sHORTNAME <- s.name,
                dIRECTION <- s.portType,

                oWNEDCOMMENTS <- OWNEDCOMMENTSType,

                tYPETREF <- TYPETREFTYPE

        ),

        OWNEDCOMMENTSType: EAST!OWNEDCOMMENTSType(

                cOMMENT<-COMMENT

        ),

        COMMENT: EAST!COMMENT(

                bODY<- s.description

        ),

        TYPETREFTYPE: EAST!TYPETREFType(

                value <- 'Boolean1',

                dEST <- s.type

        ),

        EABOOLEAN: EAST!EABOOLEAN (

                nAME <- 'Boolean'

        )

}


helper context FmiModelDescription!FmiScalarVariable def: portString: Boolean
=

        if self.causality =#input and not self."String".oclIsUndefined() then

                true

        else

                if self.causality =#output and not self."String".oclIsUndefined()
then

                        true

                else

                        false

                endif
```

```
endif;


rule FmiScalarVariable2EASTRING {

      from

            s : FmiModelDescription!FmiScalarVariable(s.portString)

      to

            FUNCTIONFLOWPORT : EAST!FUNCTIONFLOWPORT(

                  nAME <- s.name,

                  sHORTNAME <- s.name,

                  dIRECTION <- s.portType,

                  oWNEDCOMMENTS <- OWNEDCOMMENTSType,

                  tYPETREF <- TYPETREFTYPE

            ),

            OWNEDCOMMENTSType: EAST!OWNEDCOMMENTSType(

                  cOMMENT<-COMMENT

            ),

            COMMENT: EAST!COMMENT(

                  bODY<- s.description

            ),

            TYPETREFTYPE: EAST!TYPETREFType(

                  value <- 'String1',

                  dEST <- s.type

            ),

            EASTRING: EAST!EASTRING(

                  nAME <- 'String'

            )

}
```

B. Appendix B

This section shows example input and output files for the FMU2EA tool and gives a step by step explanation of creation of an example model in the generated EAST-ADL editor.

1. Input and Output Files of the FMU2EA tool

Input

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns="platform:/resource/FMU2EA7/FmiModelDescription.ecore">

<fmiModelDescription

  fmiVersion="1.0"

  modelName="Modelisar.ees"

  modelIdentifier="Modelisar_ees"

  guid="{d2df86d5-13b4-4620-bad6-58ff29ecdb65}"

  generationTool="Dymola Version 7.4, 2010-02-08"

  generationDateAndTime="2010-05-20T14:48:18Z"

  variableNamingConvention="structured"

  numberOfContinuousStates="3"

  numberOfEventIndicators="4">

  <UnitDefinitions>

    <BaseUnit

      unit="K">

      <DisplayUnitDefinition

        displayUnit="degC"

        offset="-273.15"/>

    </BaseUnit>

  </UnitDefinitions>

  <TypeDefinitions>

    <Type

      name="Modelica.Blocks.Types.Init">

      <EnumerationType min="1"
```

```
        max="4">

        <Item name="NoInit"

          description="No initialization (start values are used as guess
values with fixed=false)"/>

        <Item name="SteadyState"

          description="Steady state initialization (derivatives of states are
zero)"/>

        <Item name="InitialState"

          description="Initialization with initial states"/>

        <Item name="InitialOutput"

          description="Initialization with initial outputs (and steady state
of the states if possibles)"/>

      </EnumerationType>

    </Type>

    <Type

      name="Modelica.Blocks.Types.Smoothness">

      <EnumerationType min="1"

        max="2">

        <Item name="LinearSegments"

          description="Table points are linearly interpolated"/>

        <Item name="ContinuousDerivative"

          description="Table points are interpolated such that the first
derivative is continuous"/>

      </EnumerationType>

    </Type>

    <Type

     name="RealType1">

     <RealType

          min="1"

          max="2"

          quantity = "RealType1"
```

```
         unit="FmiTypeReal1"

     >

     </RealType>

     </Type>

     <Type

     name="IntegerType1">

     <IntegerType

          min="1"

          max="18"

          quantity = "IntegerType1"

     >

     </IntegerType>

    </Type>

    <Type

      name="EnumerationType1">

      <EnumerationType min="1"

        max="2"

        quantity = "EnumerationType1" >

        <Item name="LinearSegments"

          description="Table points are linearly interpolated"/>

        <Item name="ContinuousDerivative"

          description="Table points are interpolated such that the first
derivative is continuous"/>

      </EnumerationType>

    </Type>

  </TypeDefinitions>

  <DefaultExperiment startTime="0.0"

    stopTime="12.0"

    tolerance="0.0001"/>

  <ModelVariables>
```

```
<ScalarVariable
  name="EesI"
  valueReference="352321536"
  causality="input">
  <Integer
   min = "5"
  max = "10"
  quantity = "Integer1"/>
</ScalarVariable>
<ScalarVariable
  name="EesSoC"
  valueReference="335544323"
  causality="output">
  <Real
  min = "5.0"
  max = "10.0"
  quantity = "Real2"
  />
  <DirectDependency/>
</ScalarVariable>
    <ScalarVariable
  name="EesReal3"
  valueReference="335544323"
  causality="output">
  <Real
  min = "5.0"
  max = "10.0"
  quantity = "Real3"
  />
```

```
    <DirectDependency/>

  </ScalarVariable>

      <ScalarVariable

    name="EesReal4"

    valueReference="335544323"

    causality="output">

    <Real

    min = "5.0"

    max = "10.0"

    quantity = "Real4"

    />

    <DirectDependency/>

  </ScalarVariable>

  <ScalarVariable

    name = "EESoReal"

    valueReference="335544320"

    causality="output">

    <Real

         declaredType="RealType1"

         max="10"

      />

  </ScalarVariable>

  <ScalarVariable

    name = "EESoString"

    valueReference="335544320"

    causality="output">

    <String

      />

  </ScalarVariable>
```

```
<ScalarVariable
  name = "EESBoolean"
  valueReference="335544320"
  causality="input">
  <Boolean
    />
</ScalarVariable>
<ScalarVariable
   name = "EESoEnumeration"
  valueReference="335544320"
  causality="output">
  <Enumeration
       declaredType="EnumerationType1"
    />
</ScalarVariable>
<ScalarVariable
  name="EesIAct"
  valueReference="335544321"
  causality="output">
  <Integer
  declaredType="IntegerType1"
 />
</ScalarVariable>
<ScalarVariable
  name="EesTAct"
  valueReference="335544322"
  causality="output">
  <Integer
  min = "5"
```

```
      max = "10"

      quantity = "Integer3"/>

      <DirectDependency/>

    </ScalarVariable>

  </ModelVariables>

</fmiModelDescription>

</xmi:XMI>
```

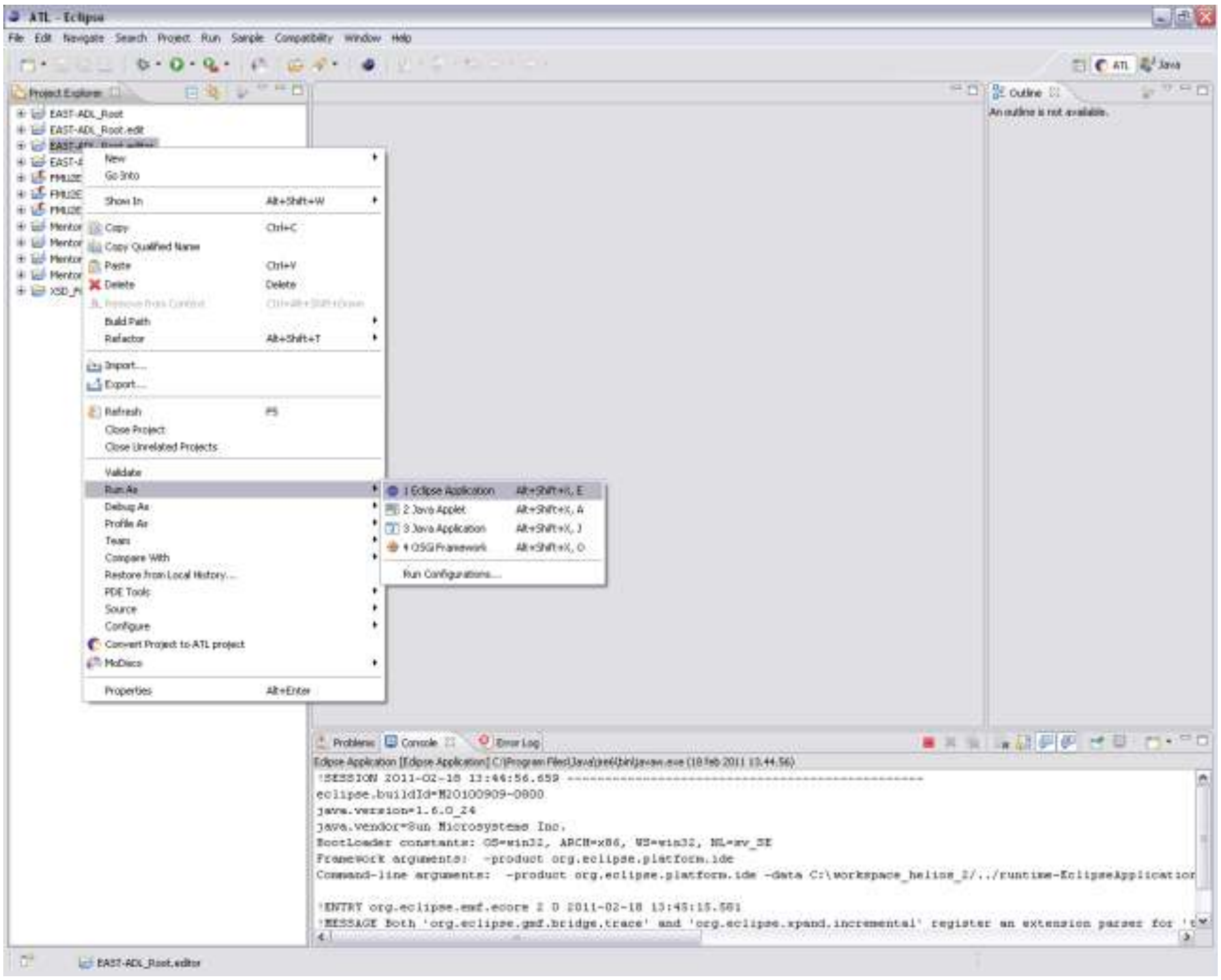
Output

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:adlrt="http://autosar.org/ADLRT">

  <adlrt:RootType>

    <eAXML>

      <tOPLEVELPACKAGES>

        <eAPACKAGE>

          <eLEMENTS>

            <aNALYSISFUNCTIONTYPE sHORTNAME="Modelisar_ees"
nAME="Modelisar_ees">

              <pORTS>

                <fUNCTIONFLOWPORT sHORTNAME="EesI" nAME="EesI"
dIRECTION="IN">

                  <oWNEDCOMMENTS>

                    <cOMMENT/>

                  </oWNEDCOMMENTS>

                  <tYPETREF value="Integer1" dEST="EA-INTEGER"/>

                </fUNCTIONFLOWPORT>

                <fUNCTIONFLOWPORT sHORTNAME="EesSoC" nAME="EesSoC"
dIRECTION="OUT">

                  <oWNEDCOMMENTS>

                    <cOMMENT/>

                  </oWNEDCOMMENTS>

                  <tYPETREF value="Real2" dEST="EA-FLOAT"/>

                </fUNCTIONFLOWPORT>

                <fUNCTIONFLOWPORT sHORTNAME="EesReal3" nAME="EesReal3"
dIRECTION="OUT">

                  <oWNEDCOMMENTS>

                    <cOMMENT/>

                  </oWNEDCOMMENTS>

                  <tYPETREF value="Real3" dEST="EA-FLOAT"/>
                </fUNCTIONFLOWPORT>

                <fUNCTIONFLOWPORT sHORTNAME="EesReal4" nAME="EesReal4"
dIRECTION="OUT">

                  <oWNEDCOMMENTS>

                    <cOMMENT/>

                  </oWNEDCOMMENTS>

                  <tYPETREF value="Real4" dEST="EA-FLOAT"/>

                </fUNCTIONFLOWPORT>

                <fUNCTIONFLOWPORT sHORTNAME="EESoReal" nAME="EESoReal"
dIRECTION="OUT">

                  <oWNEDCOMMENTS>

                    <cOMMENT/>

                  </oWNEDCOMMENTS>

                  <tYPETREF value="RealType1" dEST="EA-FLOAT"/>

                </fUNCTIONFLOWPORT>

                <fUNCTIONFLOWPORT sHORTNAME="EESoString" nAME="EESoString"
dIRECTION="OUT">

                  <oWNEDCOMMENTS>

                    <cOMMENT/>

                  </oWNEDCOMMENTS>

                  <tYPETREF value="String1" dEST="EA-STRING"/>

                </fUNCTIONFLOWPORT>

                <fUNCTIONFLOWPORT sHORTNAME="EESBoolean" nAME="EESBoolean"
dIRECTION="IN">

                  <oWNEDCOMMENTS>

                    <cOMMENT/>

                  </oWNEDCOMMENTS>

                  <tYPETREF value="Boolean1" dEST="EA-BOOLEAN"/>

                </fUNCTIONFLOWPORT>

                <fUNCTIONFLOWPORT sHORTNAME="EESoEnumeration"
nAME="EESoEnumeration" dIRECTION="OUT">

                  <oWNEDCOMMENTS>
```

```
                    <cOMMENT/>
                  </oWNEDCOMMENTS>
                  <tYPETREF value="EnumerationType1" dEST="ENUMERATION"/>
                </fUNCTIONFLOWPORT>
                <fUNCTIONFLOWPORT sHORTNAME="EesIAct" nAME="EesIAct"
dIRECTION="OUT">
                  <oWNEDCOMMENTS>
                    <cOMMENT/>
                  </oWNEDCOMMENTS>
                  <tYPETREF value="IntegerType1" dEST="EA-INTEGER"/>
                </fUNCTIONFLOWPORT>
                <fUNCTIONFLOWPORT sHORTNAME="EesTAct" nAME="EesTAct"
dIRECTION="OUT">
                  <oWNEDCOMMENTS>
                    <cOMMENT/>
                  </oWNEDCOMMENTS>
                  <tYPETREF value="Integer3" dEST="EA-INTEGER"/>
                </fUNCTIONFLOWPORT>
              </pORTS>
            </aNALYSISFUNCTIONTYPE>
          </eLEMENTS>
        </eAPACKAGE>
      </tOPLEVELPACKAGES>
    </eAXML>
  </adlrt:RootType>
  <adlrt:EAFLOAT sHORTNAME="Real2" nAME="Real2" mAX="10.0" mIN="5.0"/>
  <adlrt:RANGEABLEVALUETYPE sHORTNAME="Real2" nAME="Real2">
    <bASERANGEABLEREF value="Real2" dEST="EA-FLOAT"/>
  </adlrt:RANGEABLEVALUETYPE>
  <adlrt:EAFLOAT sHORTNAME="Real3" nAME="Real3" mAX="10.0" mIN="5.0"/>
  <adlrt:RANGEABLEVALUETYPE sHORTNAME="Real3" nAME="Real3">

    <bASERANGEABLEREF value="Real3" dEST="EA-FLOAT"/>

  </adlrt:RANGEABLEVALUETYPE>

  <adlrt:EAFLOAT sHORTNAME="Real4" nAME="Real4" mAX="10.0" mIN="5.0"/>

  <adlrt:RANGEABLEVALUETYPE sHORTNAME="Real4" nAME="Real4">

    <bASERANGEABLEREF value="Real4" dEST="EA-FLOAT"/>

  </adlrt:RANGEABLEVALUETYPE>

  <adlrt:EAFLOAT sHORTNAME="RealType1" nAME="RealType1" mAX="10.0"
mIN="1.0"/>

  <adlrt:RANGEABLEVALUETYPE sHORTNAME="RealType1" nAME="RealType1"
dIMENSION="FmiTypeReal1" uNIT="FmiTypeReal1">

    <oWNEDCOMMENTS>

      <cOMMENT/>

    </oWNEDCOMMENTS>

    <bASERANGEABLEREF value="RealType1" dEST="EA-FLOAT"/>

  </adlrt:RANGEABLEVALUETYPE>

  <adlrt:EAINTEGER sHORTNAME="Integer1" nAME="Integer1" mAX="10" mIN="5"/>

  <adlrt:RANGEABLEVALUETYPE sHORTNAME="Integer1" nAME="Integer1"
dIMENSION="Integer1">

    <bASERANGEABLEREF value="Integer1" dEST="EA-INTEGER"/>

  </adlrt:RANGEABLEVALUETYPE>

  <adlrt:EAINTEGER sHORTNAME="Integer3" nAME="Integer3" mAX="10" mIN="5"/>

  <adlrt:RANGEABLEVALUETYPE sHORTNAME="Integer3" nAME="Integer3"
dIMENSION="Integer3">

    <bASERANGEABLEREF value="Integer3" dEST="EA-INTEGER"/>

  </adlrt:RANGEABLEVALUETYPE>

  <adlrt:EAINTEGER sHORTNAME="IntegerType1" nAME="IntegerType1" mAX="18"
mIN="1"/>

  <adlrt:RANGEABLEVALUETYPE sHORTNAME="IntegerType1" nAME="IntegerType1"
dIMENSION="IntegerType1">

    <bASERANGEABLEREF value="IntegerType1" dEST="EA-INTEGER"/>

  </adlrt:RANGEABLEVALUETYPE>
  <adlrt:ENUMERATION sHORTNAME="EnumerationType1" nAME="EnumerationType1">

    <lITERALS>

      <eNUMERATIONLITERAL sHORTNAME="LinearSegments" nAME="LinearSegments">

        <oWNEDCOMMENTS>

          <cOMMENT bODY="Table points are linearly interpolated"/>

        </oWNEDCOMMENTS>

      </eNUMERATIONLITERAL>

      <eNUMERATIONLITERAL sHORTNAME="ContinuousDerivative"
nAME="ContinuousDerivative">

        <oWNEDCOMMENTS>

          <cOMMENT bODY="Table points are interpolated such that the first
derivative is continuous"/>

        </oWNEDCOMMENTS>

      </eNUMERATIONLITERAL>

    </lITERALS>

  </adlrt:ENUMERATION>

  <adlrt:ENUMERATIONVALUETYPE sHORTNAME="EnumerationType1"
nAME="EnumerationType1" dIMENSION="EnumerationType1">

    <bASEENUMERATIONREF value="EnumerationType1" dEST="ENUMERATION"/>

  </adlrt:ENUMERATIONVALUETYPE>

  <adlrt:EABOOLEAN nAME="Boolean"/>

  <adlrt:EASTRING nAME="String"/>

</xmi:XMI>
```

2. Generating a Model Using the Editor

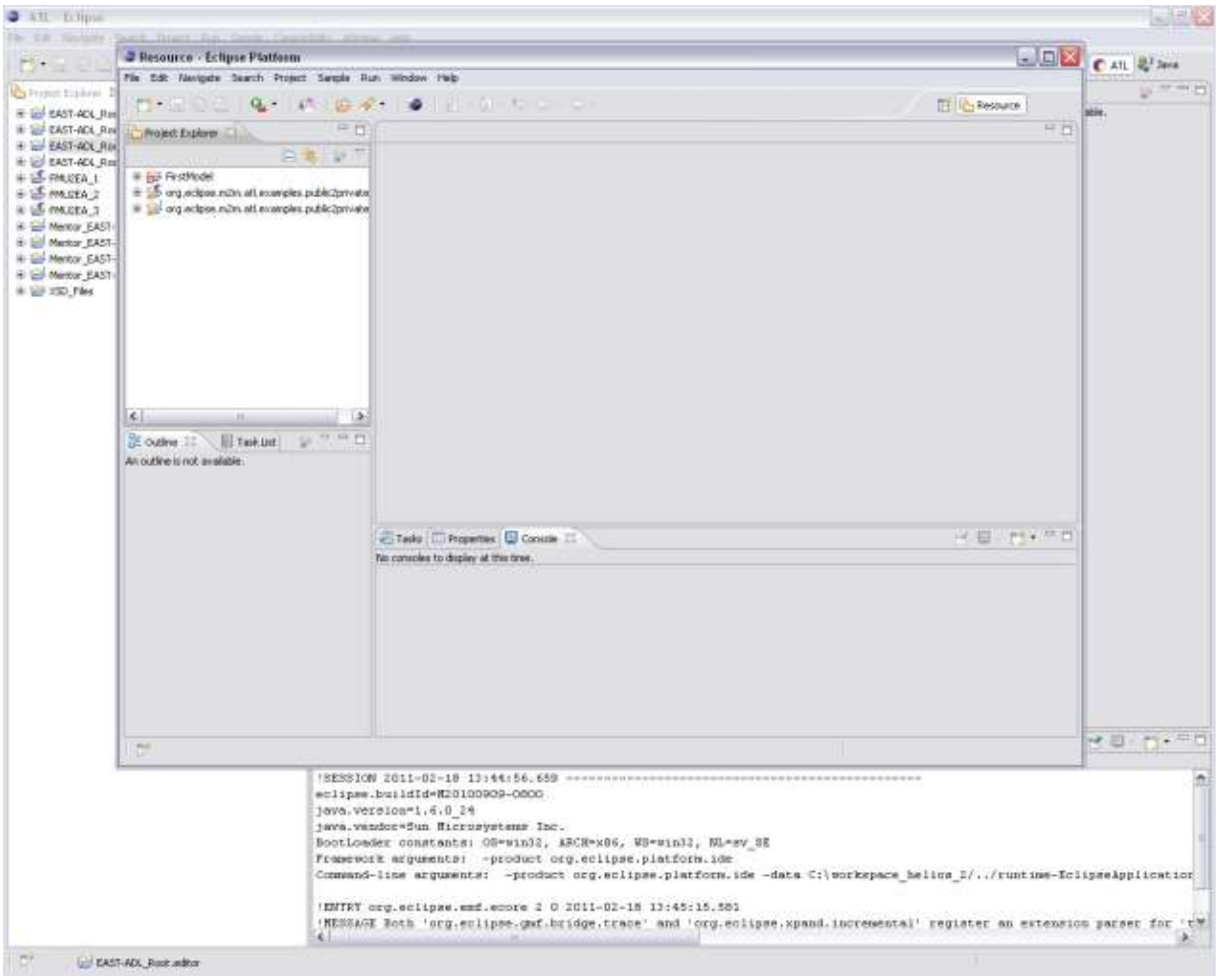
This Figure shows the start page of Eclipse. In the Eclipse that can be seen from the Figure, generated Editor's source code is available. These source codes are mainly Java Files. In order to run the editor, these Java files should be run, which is what is shown in Figure below.



```
Eclipse Application [Eclipse Application] C:\Program Files\Java\jre6\bin\javaw.exe (18 feb 2011 13.44.56)
!SESSION 2011-02-18 13:44:56.659 -----------------------------------------------
eclipse.buildId=M20100909-0800
java.version=1.6.0_24
java.vendor=Sun Microsystems Inc.
BootLoader constants: OS=win32, ARCH=x86, WS=win32, NL=sv_SE
Framework arguments:  -product org.eclipse.platform.ide
Command-line arguments:  -product org.eclipse.platform.ide -data C:\workspace_helios_2/../runtime-EclipseApplication

!ENTRY org.eclipse.emf.ecore 2 0 2011-02-18 13:45:15.581
!MESSAGE Both 'org.eclipse.gmf.bridge.trace' and 'org.eclipse.xpand.incremental' register an extension parser for 't
```

Figure B2.1

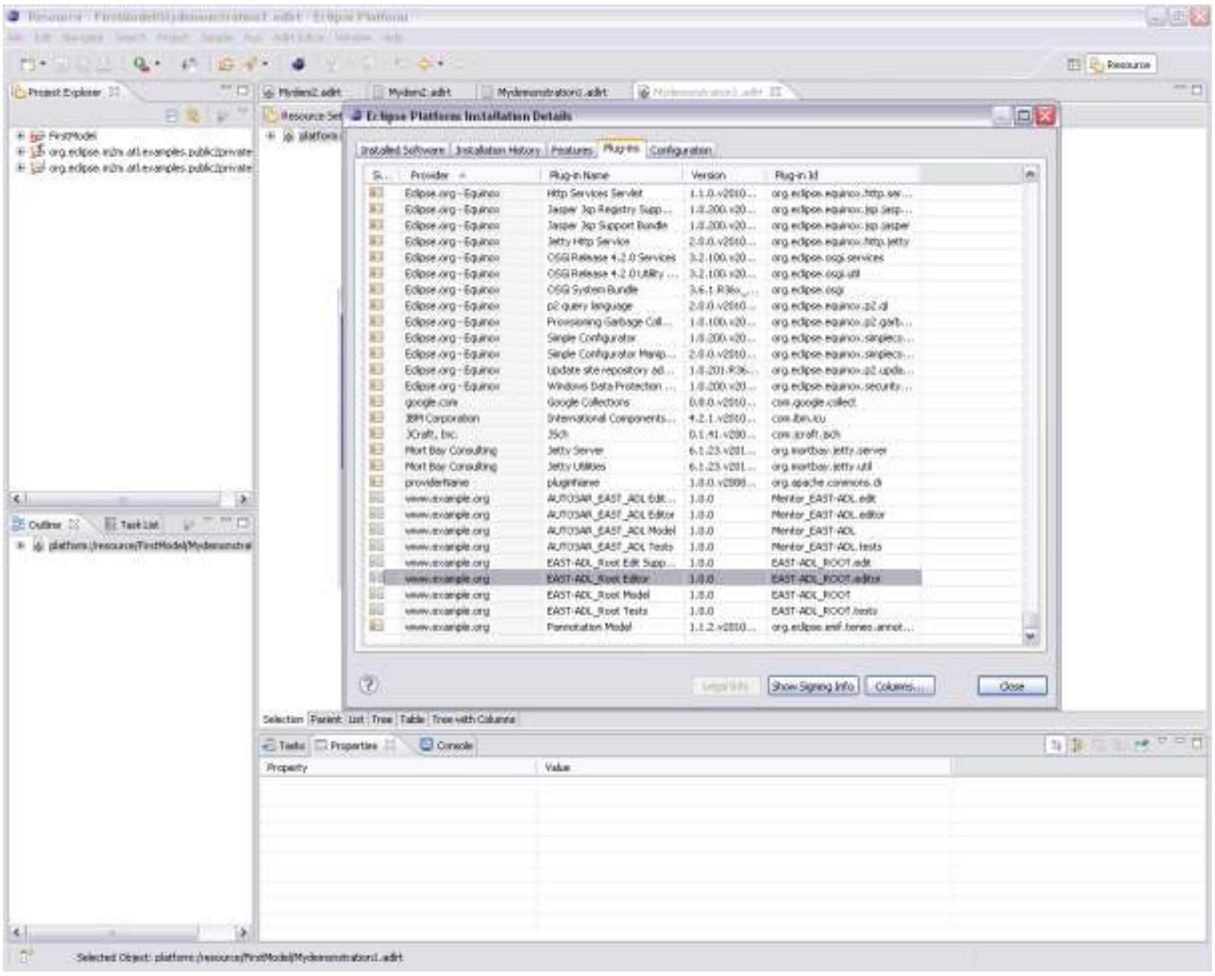
Figure B2.2 shows the start page of the 2nd instance of Eclipse which is running on the Eclipse that has the source files. This 2nd instance has the Editor as a plug-in.



```
!SESSION 2011-02-18 13:44:56.659 -----------------------------------------------
eclipse.buildId=M20100909-0800
java.version=1.6.0_24
java.vendor=Sun Microsystems Inc.
BootLoader constants: OS=win32, ARCH=x86, WS=win32, NL=sv_SE
Framework arguments:  -product org.eclipse.platform.ide
Command-line arguments:  -product org.eclipse.platform.ide -data C:\workspace_helios_2/../runtime-EclipseApplicatior

!ENTRY org.eclipse.emf.ecore 2 0 2011-02-18 13:45:15.581
!MESSAGE Both 'org.eclipse.gmf.bridge.trace' and 'org.eclipse.xpand.incremental' register an extension parser for 't
```

Figure B2.2

As can be seen from the Figure, the Editor is shown as a Plug-In.



Eclipse Platform Installation Details

Si...	Provider	Plug-in Name	Version	Plug-in Id
	Eclipse.org - Equinox	Http Services Servlet	1.1.0.v2010...	org.eclipse.equinox.http.ser...
	Eclipse.org - Equinox	Jasper Jsp Registry Supp...	1.0.200.v20...	org.eclipse.equinox.jsp.jasp...
	Eclipse.org - Equinox	Jasper Jsp Support Bundle	1.0.200.v20...	org.eclipse.equinox.jsp.jasper
	Eclipse.org - Equinox	Jetty Http Service	2.0.0.v2010...	org.eclipse.equinox.http.jetty
	Eclipse.org - Equinox	OSGi Release 4.2.0 Services	3.2.100.v20...	org.eclipse.osgi.services
	Eclipse.org - Equinox	OSGi Release 4.2.0 Utility ...	3.2.100.v20...	org.eclipse.osgi.util
	Eclipse.org - Equinox	OSGi System Bundle	3.6.1.R36x_...	org.eclipse.osgi
	Eclipse.org - Equinox	p2 query language	2.0.0.v2010...	org.eclipse.equinox.p2.ql
	Eclipse.org - Equinox	Provisioning Garbage Coll...	1.0.100.v20...	org.eclipse.equinox.p2.garb...
	Eclipse.org - Equinox	Simple Configurator	1.0.200.v20...	org.eclipse.equinox.simpleco...
	Eclipse.org - Equinox	Simple Configurator Manip...	2.0.0.v2010...	org.eclipse.equinox.simpleco...
	Eclipse.org - Equinox	Update site repository ad...	1.0.201.R36...	org.eclipse.equinox.p2.upda...
	Eclipse.org - Equinox	Windows Data Protection ...	1.0.200.v20...	org.eclipse.equinox.security...
	google.com	Google Collections	0.8.0.v2010...	com.google.collect
	IBM Corporation	International Components...	4.2.1.v2010...	com.ibm.icu
	JCraft, Inc.	JSch	0.1.41.v200...	com.jcraft.jsch
	Mort Bay Consulting	Jetty Server	6.1.23.v201...	org.mortbay.jetty.server
	Mort Bay Consulting	Jetty Utilities	6.1.23.v201...	org.mortbay.jetty.util
	providerName	pluginName	1.0.0.v2008...	org.apache.commons.di
	www.example.org	AUTOSAR_EAST_ADL Edit...	1.0.0	Mentor_EAST-ADL.edit
	www.example.org	AUTOSAR_EAST_ADL Editor	1.0.0	Mentor_EAST-ADL.editor
	www.example.org	AUTOSAR_EAST_ADL Model	1.0.0	Mentor_EAST-ADL
	www.example.org	AUTOSAR_EAST_ADL Tests	1.0.0	Mentor_EAST-ADL.tests
	www.example.org	EAST-ADL_Root Edit Supp...	1.0.0	EAST-ADL_ROOT.edit
	www.example.org	EAST-ADL_Root Editor	1.0.0	EAST-ADL_ROOT.editor
	www.example.org	EAST-ADL_Root Model	1.0.0	EAST-ADL_ROOT
	www.example.org	EAST-ADL_Root Tests	1.0.0	EAST-ADL_ROOT.tests
	www.example.org	Pannotation Model	1.1.2.v2010...	org.eclipse.emf.teneo.annot...

Selected Object: platform:/resource/FirstModel/Mydemonstration1.adlt

Figure B2.3

First step, after the new instance of the Eclipse is opened, to create a new EAST-ADL Model

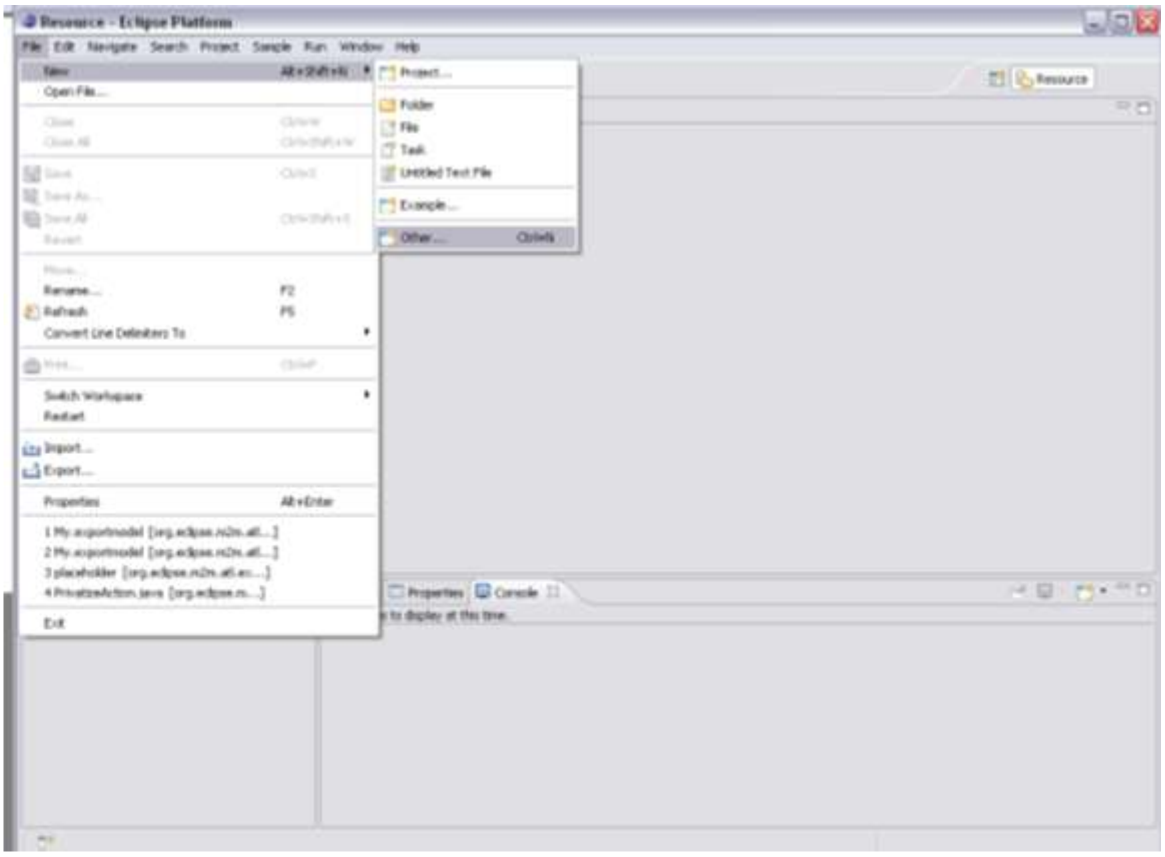


Figure B2.4

There are different models that can be created. These models are either example models that can be found in Eclipse as default or the models whose Editor source files are available in the main instance of Eclipse that is running.

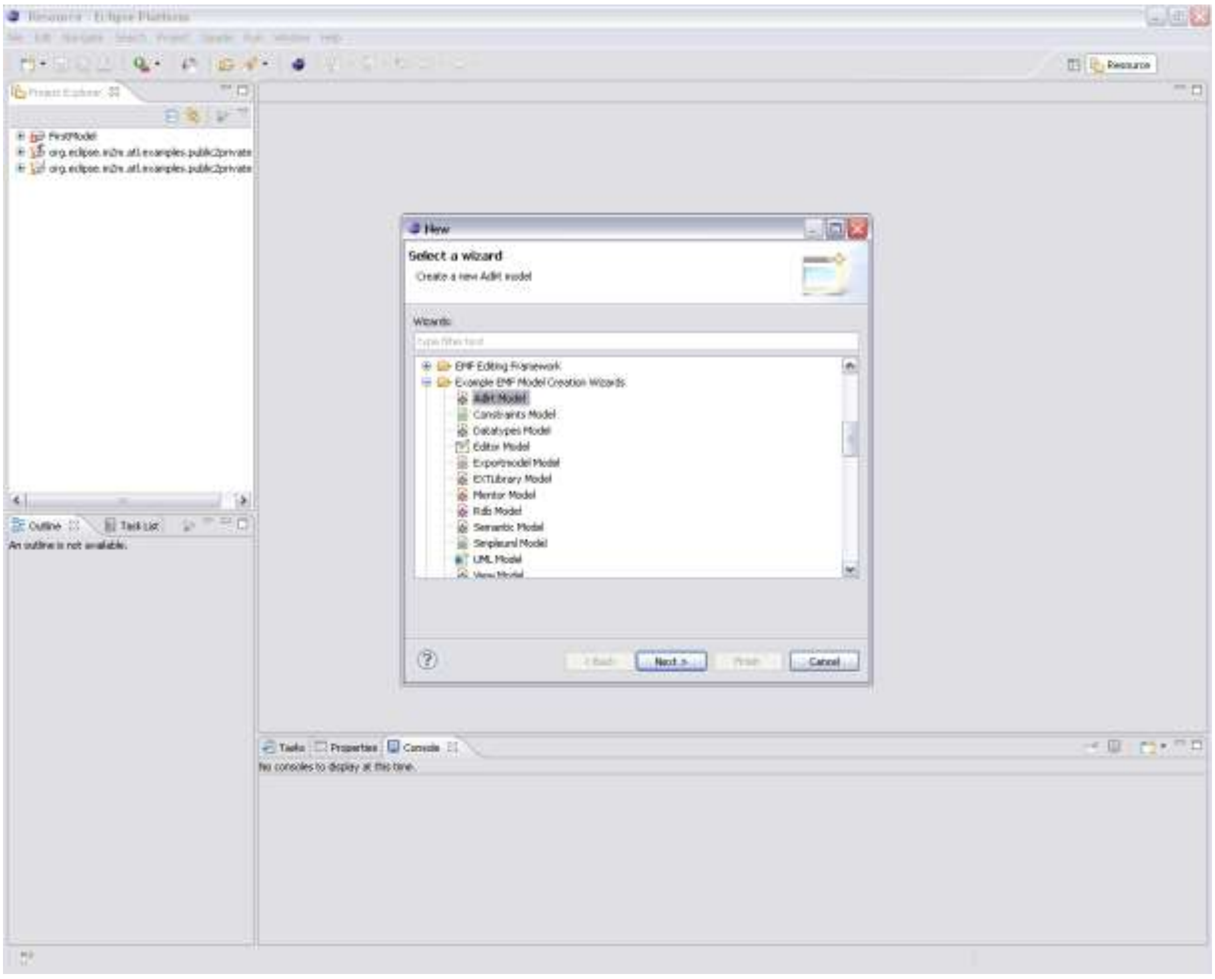
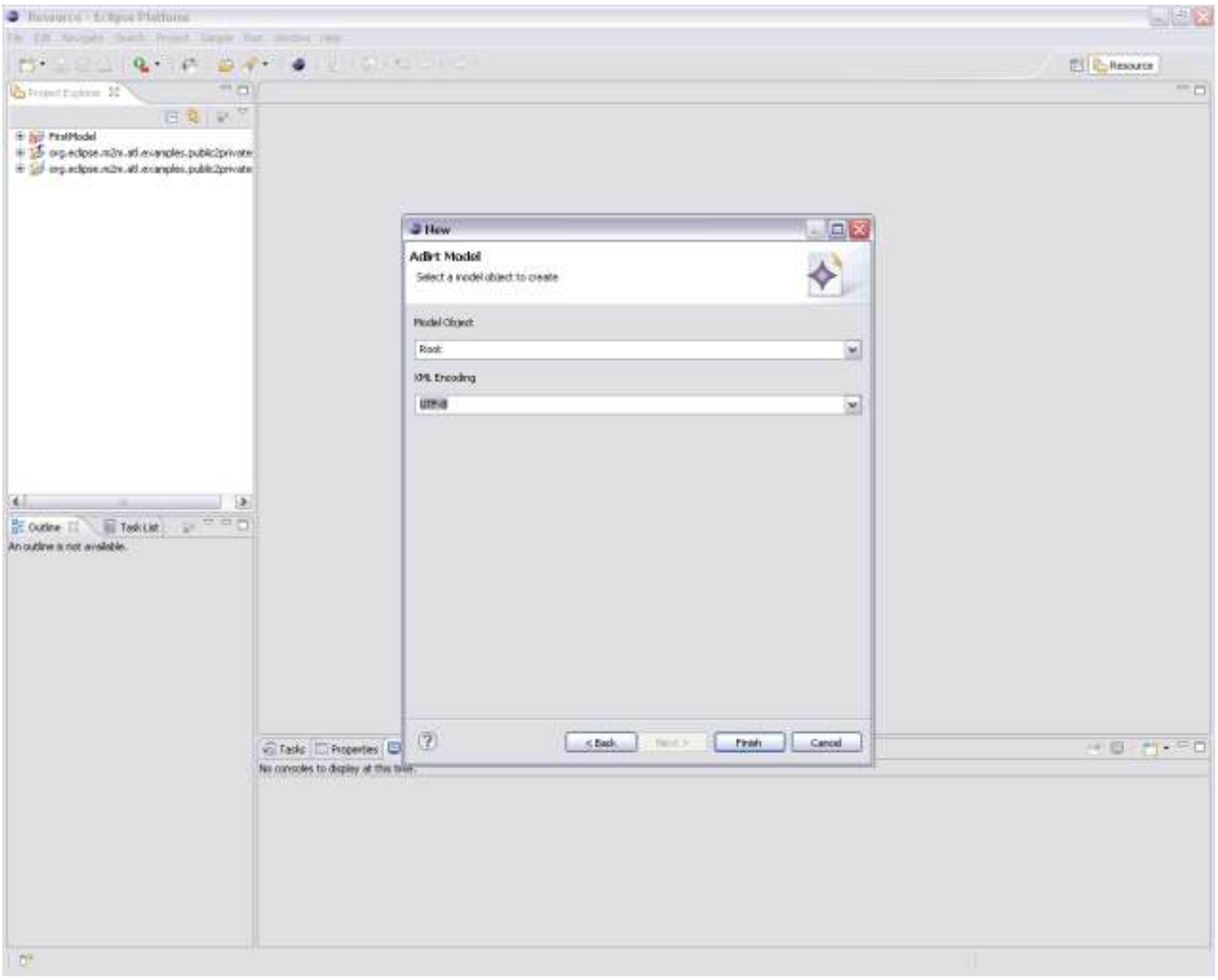


Figure B2.5

The Figure shows the 3rd step of creating an empty East-ADL model. This step lets the user choose the encoding type of the model. It also shows which kind of model is wanted to be created. The reason of this option is because an XSD can include several different architecture description languages which results as different choices in the Editor side. In this version of the Editor, we only have 1 kind of model which is adlrt as the XSD with AUTOSAR and EAST-ADL elements is merged under a common element and this XSD is used to generate an Editor.


Resource - Eclipse Platform
Project Explorer
FirstModel
org.eclipse.m2m.atl.examples.public2private
org.eclipse.m2m.atl.examples.public2private
Outline
Task List
An outline is not available.
Resource
Tasks
Properties
No consoles to display at this time.
New
< Back
Next >
Finish
Cancel

Adlrt Model

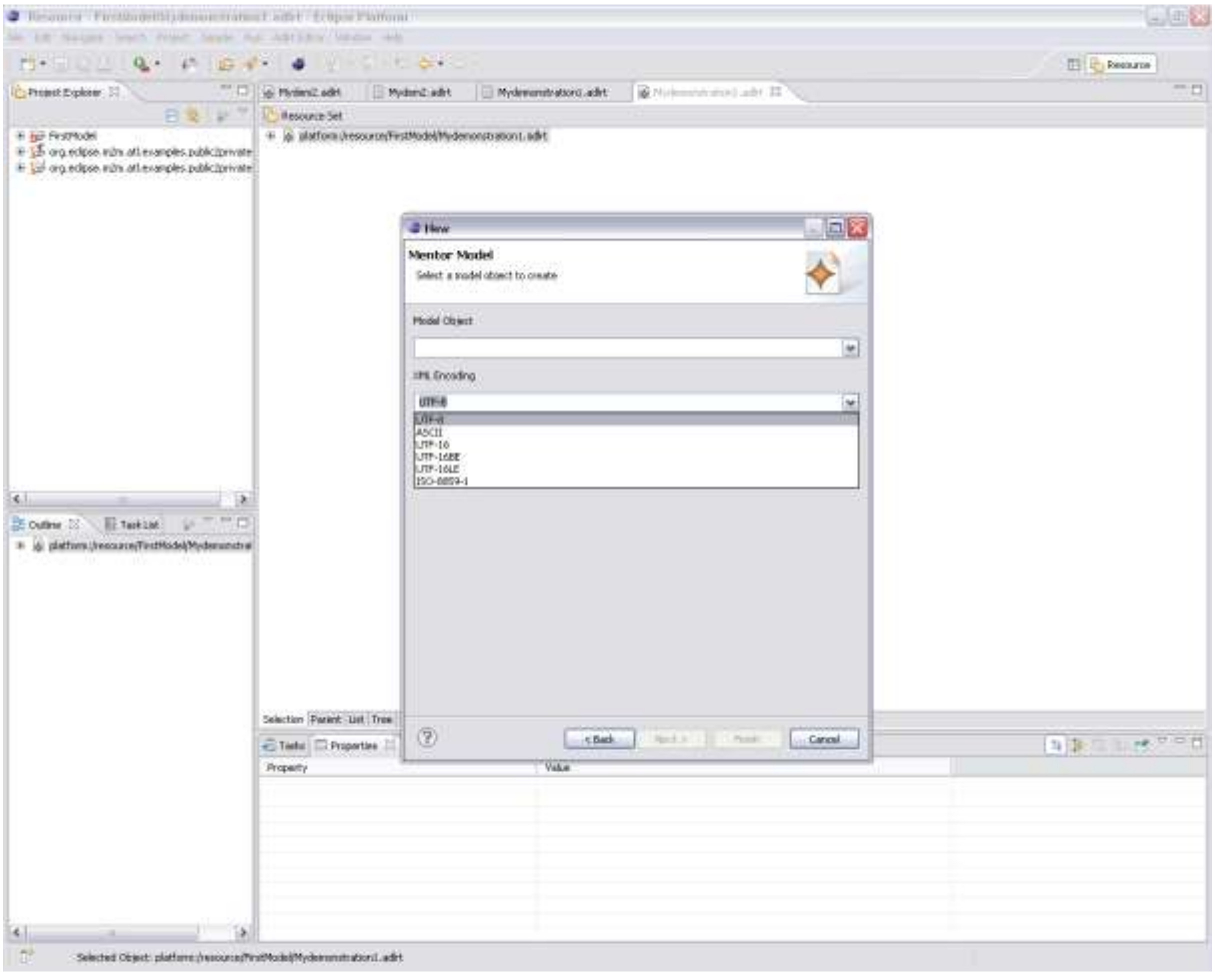
Select a model object to create

Model Object: Root

XML Encoding: UTF-8

Figure B2.6

This Figure shows the possible encodeing types that can be selected.



UTF-8

UTF-8
ASCII
UTF-16
UTF-16BE
UTF-16LE
ISO-8859-1

Selected Object: platform:/resource/FirstModel/Mydemonstration1.adrt

Figure B2.7

This Figure shows an empty EAST-ADL model. As AUTOSAR and EAST-ADL elements are merged under Root Type elements as can be seen from the Figure, an AUTOSAR or EAST-ADL model can be formed under this element. The root elements of these models are AUTOSAR and EAXML.

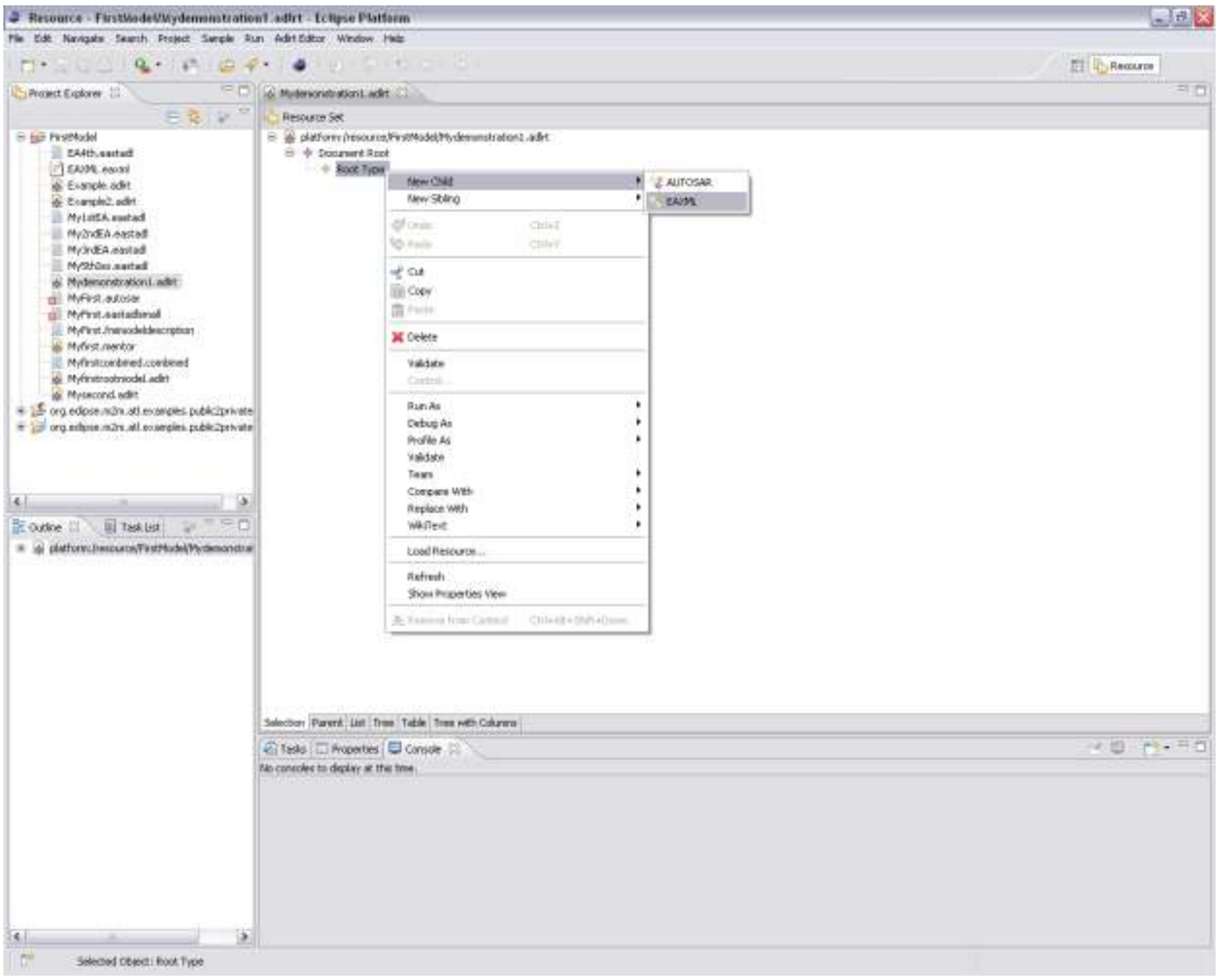


Figure B2.8

In the figure below, an AUTOSAR model which has some elements is shown. Additional elements is tried to be added to the model under the Elements element of it. A part of the list of the elements that can be added to this element can be seen from the Figure. The way that one can add elements or relationships is directly conforming to AUTOSAR metamodel, so one can say that the tool guides you in a strict way during the design process.

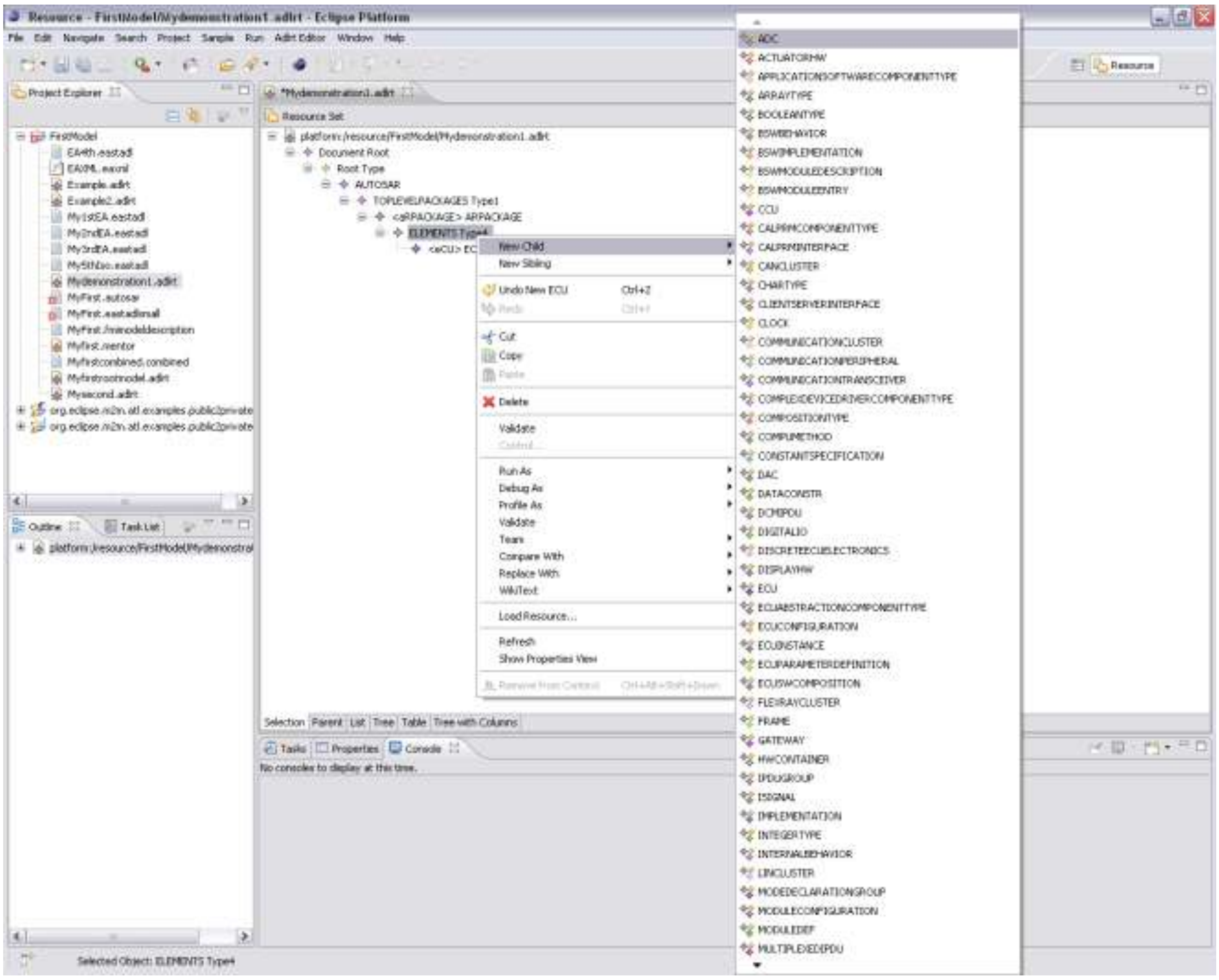


Figure B2.9

In the figure below, an EAST-ADL model which has some elements is shown. Additional elements is tried to be added to the model under the Elements element of it. A part of the list of the elements that can be added to this element can be seen from the Figure.

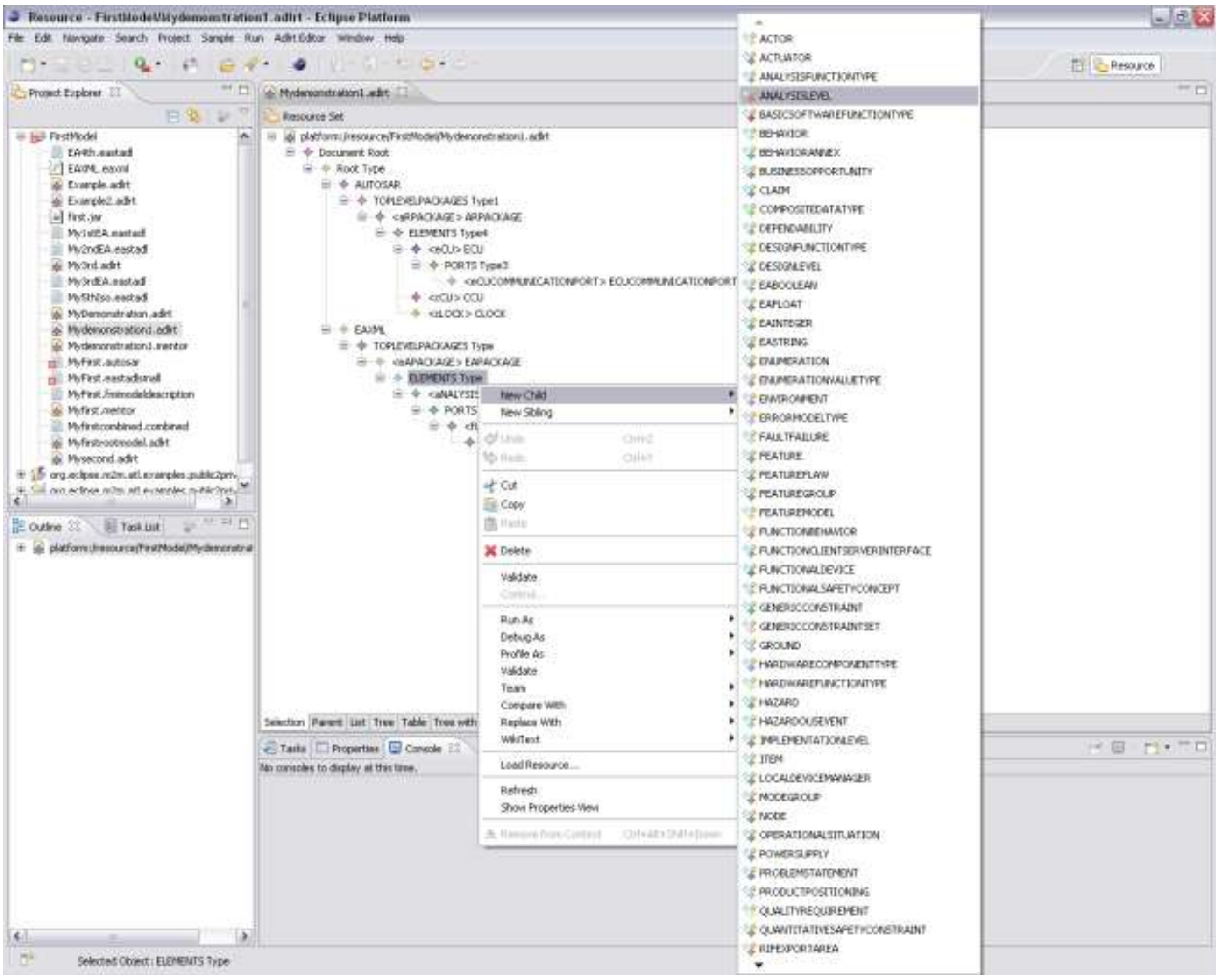
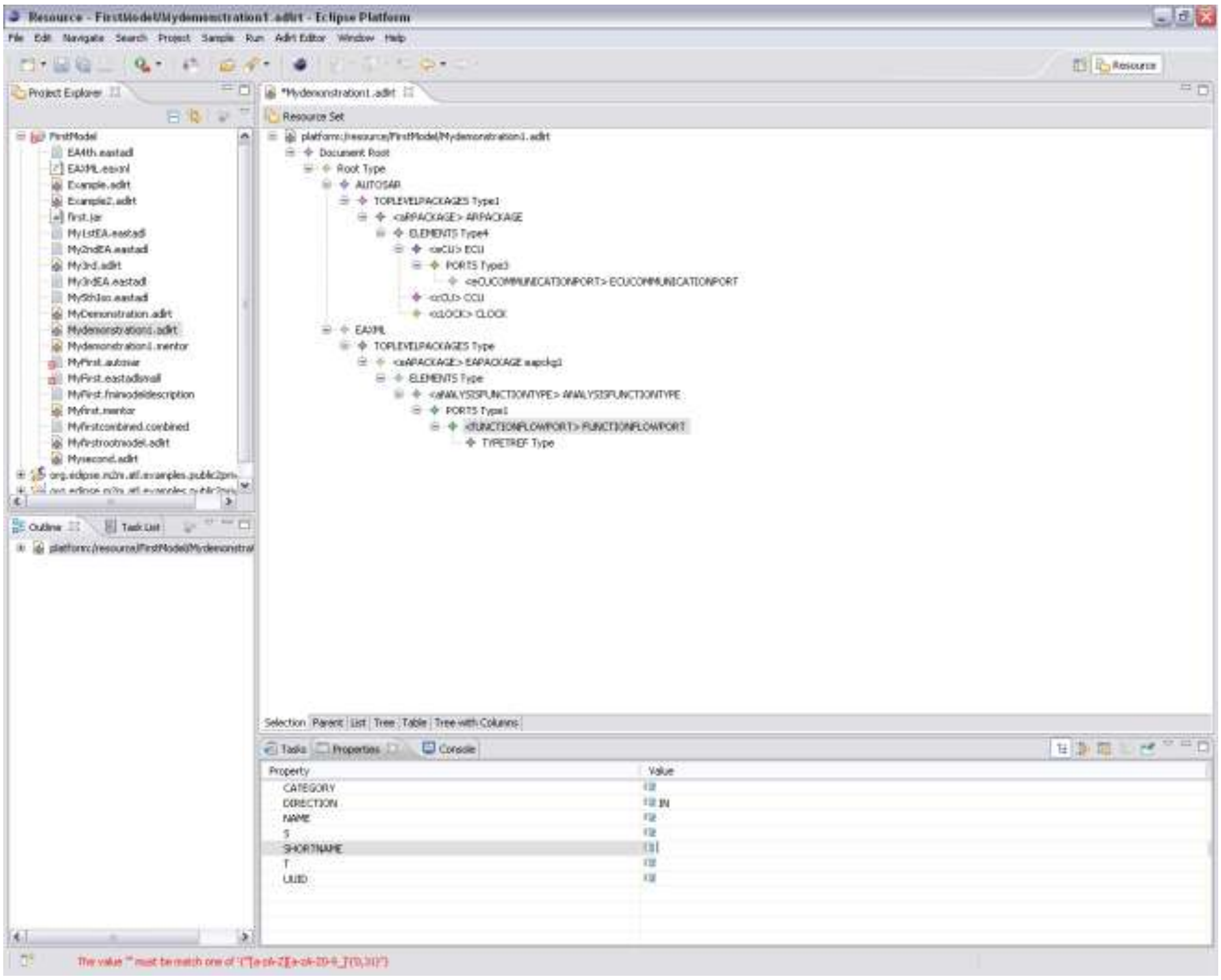


Figure B2.10

As can be seen in the Figure, the attriburtes of the FunctionFlowPort's element is shown. ShortName attribute is selected to be entered a value. Editor directs the user to enter the value of the attribute in a specific format. So the editor makes sure the model is valid in some sense.


Resource - FirstModel/Mydemonstration1.adlrt - Eclipse Platform
File Edit Navigate Search Project Sample Run Adlrt Editor Window Help
Resource
Project Explorer
*Mydemonstration1.adlrt
Resource Set
Outline
Task List
Selection Parent List Tree Table Tree with Columns
Tasks
Properties
Console

```
FirstModel
    EA4th.eastadl
    EAXML.eaxml
    Example.adlrt
    Example2.adlrt
    first.jar
    My1stEA.eastadl
    My2ndEA.eastadl
    My3rd.adlrt
    My3rdEA.eastadl
    My5th1oo.eastadl
    MyDemonstration.adlrt
    Mydemonstration1.adlrt
    Mydemonstration1.mentor
    MyFirst.autosar
    MyFirst.eastadlsmall
    MyFirst.fmmodeldescription
    Myfirst.mentor
    Myfirstcombined.combined
    Myfirstrootmodel.adlrt
    Mysecond.adlrt
org.eclipse.m2m.atl.examples.public2pri
org.eclipse.m2m.atl.examples.public2pri

platform:/resource/FirstModel/Mydemonstral
```

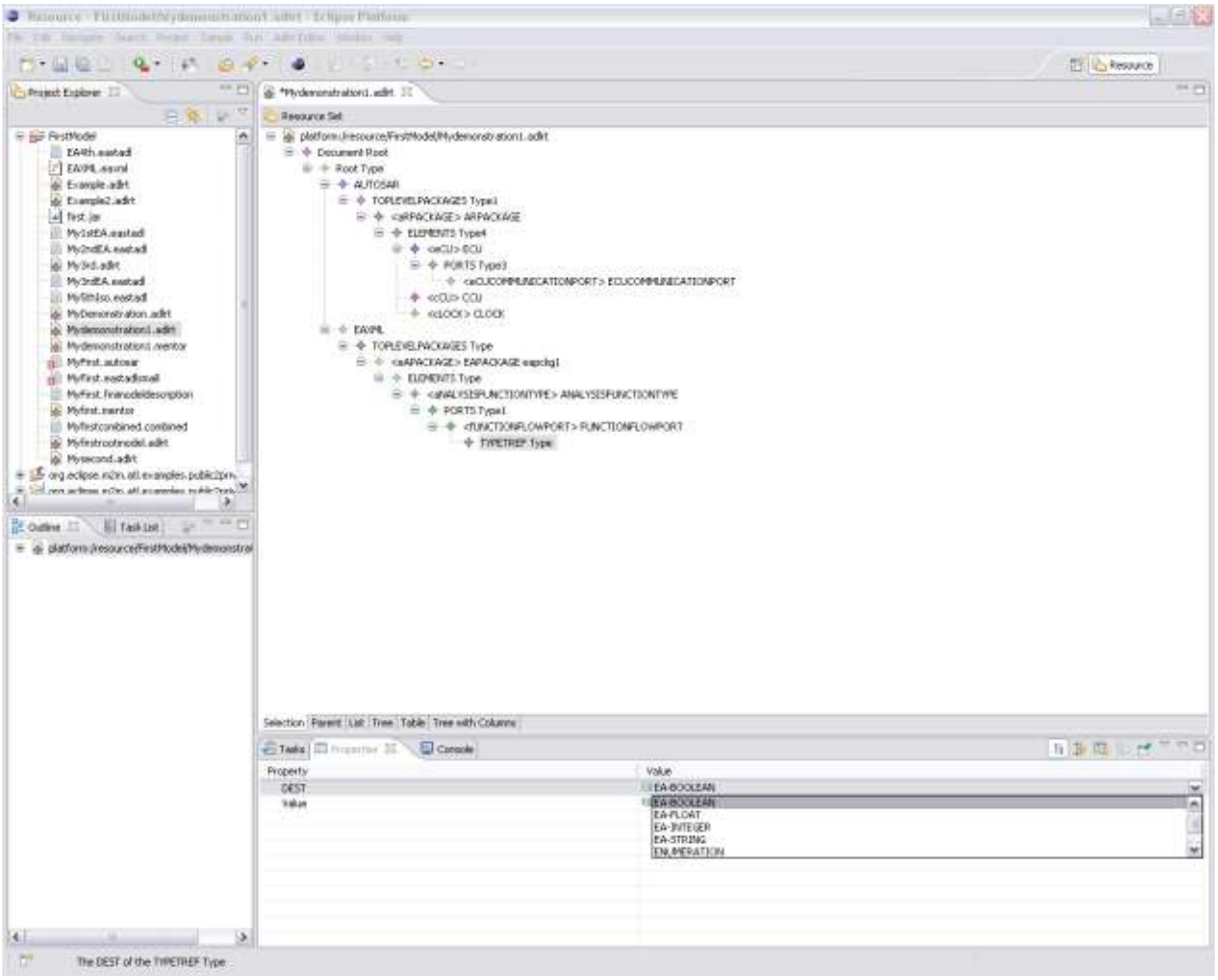
```
platform:/resource/FirstModel/Mydemonstration1.adlrt
  Document Root
    Root Type
      AUTOSAR
        TOPLEVELPACKAGES Type1
          <aRPACKAGE> ARPACKAGE
            ELEMENTS Type4
              <eCU> ECU
                PORTS Type3
                  <eCUCOMMUNICATIONPORT> ECUCOMMUNICATIONPORT
              <cCU> CCU
              <cLOCK> CLOCK
      EAXML
        TOPLEVELPACKAGES Type
          <eAPACKAGE> EAPACKAGE eapckg1
            ELEMENTS Type
              <aNALYSISFUNCTIONTYPE> ANALYSISFUNCTIONTYPE
                PORTS Type1
                  <fUNCTIONFLOWPORT> FUNCTIONFLOWPORT
                    TYPETREF Type
```

Property	Value
CATEGORY	
DIRECTION	IN
NAME	
S	
SHORTNAME	
T	
UUID	

```
The value '' must be match one of '{"[a-zA-Z][a-zA-Z0-9_]{0,31}"}'
```

Figure B2.11

Below figure shows when the user is trying to enter the type of a FunctionFlowPort. As can be seen the type can be chosen from a menu, it is not entered directly to a white box. This menu ensure that a valid type is selected.



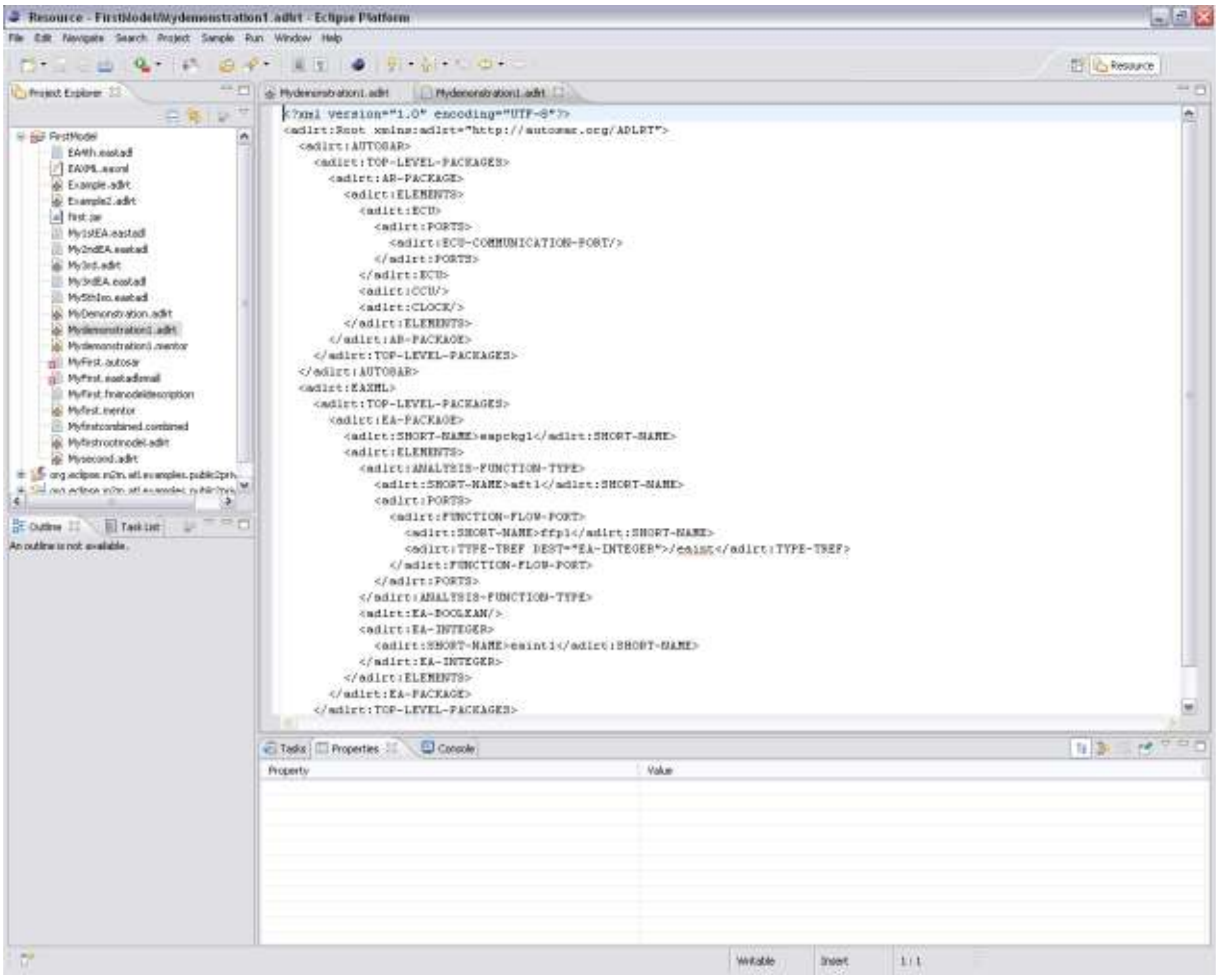
```
platform:/resource/FirstModel/Mydemonstration1.adlt
  Document Root
    Root Type
      AUTOSAR
        TOPLEVELPACKAGES Type1
          <aRPACKAGE> ARPACKAGE
            ELEMENTS Type4
              <eCU> ECU
              PORTS Type3
                <eCUCOMMUNICATIONPORT> ECUCOMMUNICATIONPORT
              <cCU> CCU
              <cLOCK> CLOCK
      EAXML
        TOPLEVELPACKAGES Type
          <eAPACKAGE> EAPACKAGE eapckg1
            ELEMENTS Type
              <aNALYSISFUNCTIONTYPE> ANALYSISFUNCTIONTYPE
                PORTS Type1
                  <fUNCTIONFLOWPORT> FUNCTIONFLOWPORT
                    TYPETREF Type
```

Property	Value
DEST	EA-BOOLEAN
Value	EA-BOOLEAN EA-FLOAT EA-INTEGER EA-STRING ENUMERATION

The DEST of the TYPETREF Type

Figure B2.12

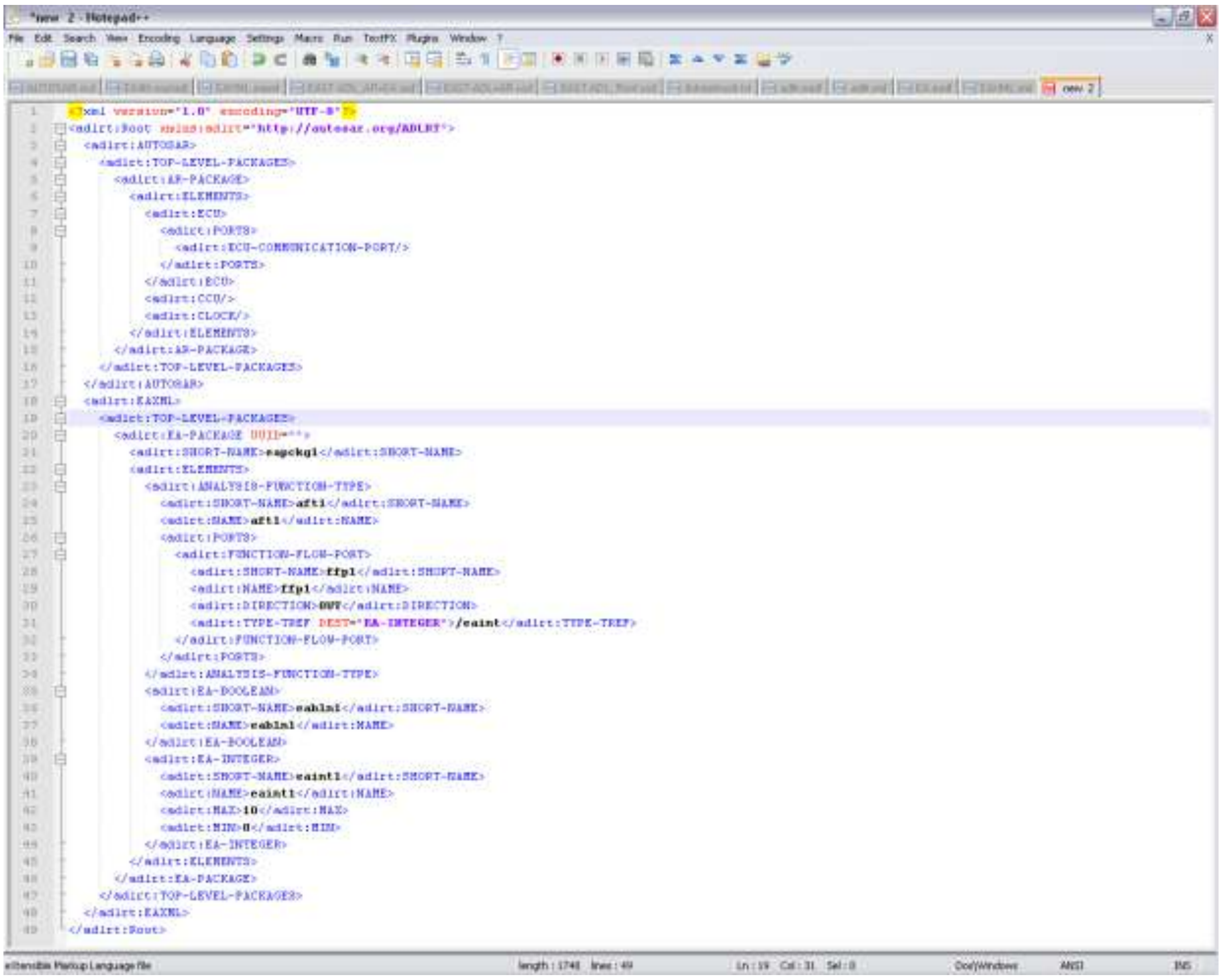
The figure below shows the notepad view of the model formed so far.



```
<?xml version="1.0" encoding="UTF-8"?>
<adlrt:Root xmlns:adlrt="http://autosar.org/ADLRT">
  <adlrt:AUTOSAR>
    <adlrt:TOP-LEVEL-PACKAGES>
      <adlrt:AR-PACKAGE>
        <adlrt:ELEMENTS>
          <adlrt:ECU>
            <adlrt:PORTS>
              <adlrt:ECU-COMMUNICATION-PORT/>
            </adlrt:PORTS>
          </adlrt:ECU>
          <adlrt:CCU/>
          <adlrt:CLOCK/>
        </adlrt:ELEMENTS>
      </adlrt:AR-PACKAGE>
    </adlrt:TOP-LEVEL-PACKAGES>
  </adlrt:AUTOSAR>
  <adlrt:EAXML>
    <adlrt:TOP-LEVEL-PACKAGES>
      <adlrt:EA-PACKAGE>
        <adlrt:SHORT-NAME>eapckg1</adlrt:SHORT-NAME>
        <adlrt:ELEMENTS>
          <adlrt:ANALYSIS-FUNCTION-TYPE>
            <adlrt:SHORT-NAME>aft1</adlrt:SHORT-NAME>
            <adlrt:PORTS>
              <adlrt:FUNCTION-FLOW-PORT>
                <adlrt:SHORT-NAME>ffp1</adlrt:SHORT-NAME>
                <adlrt:TYPE-TREF DEST="EA-INTEGER">/eaint</adlrt:TYPE-TREF>
              </adlrt:FUNCTION-FLOW-PORT>
            </adlrt:PORTS>
          </adlrt:ANALYSIS-FUNCTION-TYPE>
          <adlrt:EA-BOOLEAN/>
          <adlrt:EA-INTEGER>
            <adlrt:SHORT-NAME>eaint1</adlrt:SHORT-NAME>
          </adlrt:EA-INTEGER>
        </adlrt:ELEMENTS>
      </adlrt:EA-PACKAGE>
    </adlrt:TOP-LEVEL-PACKAGES>
```

Figure B2.13

The figure below shows another notepad view of the model formed so far.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<adlrt:Root xmlns:adlrt="http://autosar.org/ADLRT">
  <adlrt:AUTOSAR>
    <adlrt:TOP-LEVEL-PACKAGES>
      <adlrt:AR-PACKAGE>
        <adlrt:ELEMENTS>
          <adlrt:ECU>
            <adlrt:PORTS>
              <adlrt:ECU-COMMUNICATION-PORT/>
            </adlrt:PORTS>
          </adlrt:ECU>
          <adlrt:CCU/>
          <adlrt:CLOCK/>
        </adlrt:ELEMENTS>
      </adlrt:AR-PACKAGE>
    </adlrt:TOP-LEVEL-PACKAGES>
  </adlrt:AUTOSAR>
  <adlrt:EAXML>
    <adlrt:TOP-LEVEL-PACKAGES>
      <adlrt:EA-PACKAGE UUID="">
        <adlrt:SHORT-NAME>eapckg1</adlrt:SHORT-NAME>
        <adlrt:ELEMENTS>
          <adlrt:ANALYSIS-FUNCTION-TYPE>
            <adlrt:SHORT-NAME>aft1</adlrt:SHORT-NAME>
            <adlrt:NAME>aft1</adlrt:NAME>
            <adlrt:PORTS>
              <adlrt:FUNCTION-FLOW-PORT>
                <adlrt:SHORT-NAME>ffp1</adlrt:SHORT-NAME>
                <adlrt:NAME>ffp1</adlrt:NAME>
                <adlrt:DIRECTION>OUT</adlrt:DIRECTION>
                <adlrt:TYPE-TREF DEST="EA-INTEGER">/eaint</adlrt:TYPE-TREF>
              </adlrt:FUNCTION-FLOW-PORT>
            </adlrt:PORTS>
          </adlrt:ANALYSIS-FUNCTION-TYPE>
          <adlrt:EA-BOOLEAN>
            <adlrt:SHORT-NAME>eablnl</adlrt:SHORT-NAME>
            <adlrt:NAME>eablnl</adlrt:NAME>
          </adlrt:EA-BOOLEAN>
          <adlrt:EA-INTEGER>
            <adlrt:SHORT-NAME>eaint1</adlrt:SHORT-NAME>
            <adlrt:NAME>eaint1</adlrt:NAME>
            <adlrt:MAX>10</adlrt:MAX>
            <adlrt:MIN>0</adlrt:MIN>
          </adlrt:EA-INTEGER>
        </adlrt:ELEMENTS>
      </adlrt:EA-PACKAGE>
    </adlrt:TOP-LEVEL-PACKAGES>
  </adlrt:EAXML>
</adlrt:Root>
```

*new 2 - Notepad++
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
eXtensible Markup Language file
length : 1748 lines : 49
Ln : 19 Col : 31 Sel : 0
Dos\Windows
ANSI
INS

Figure B2.14

The Editor also has a validation functionality. This service lets the user to make sure that the model conforms to the MetaModel and so it is a valid one.

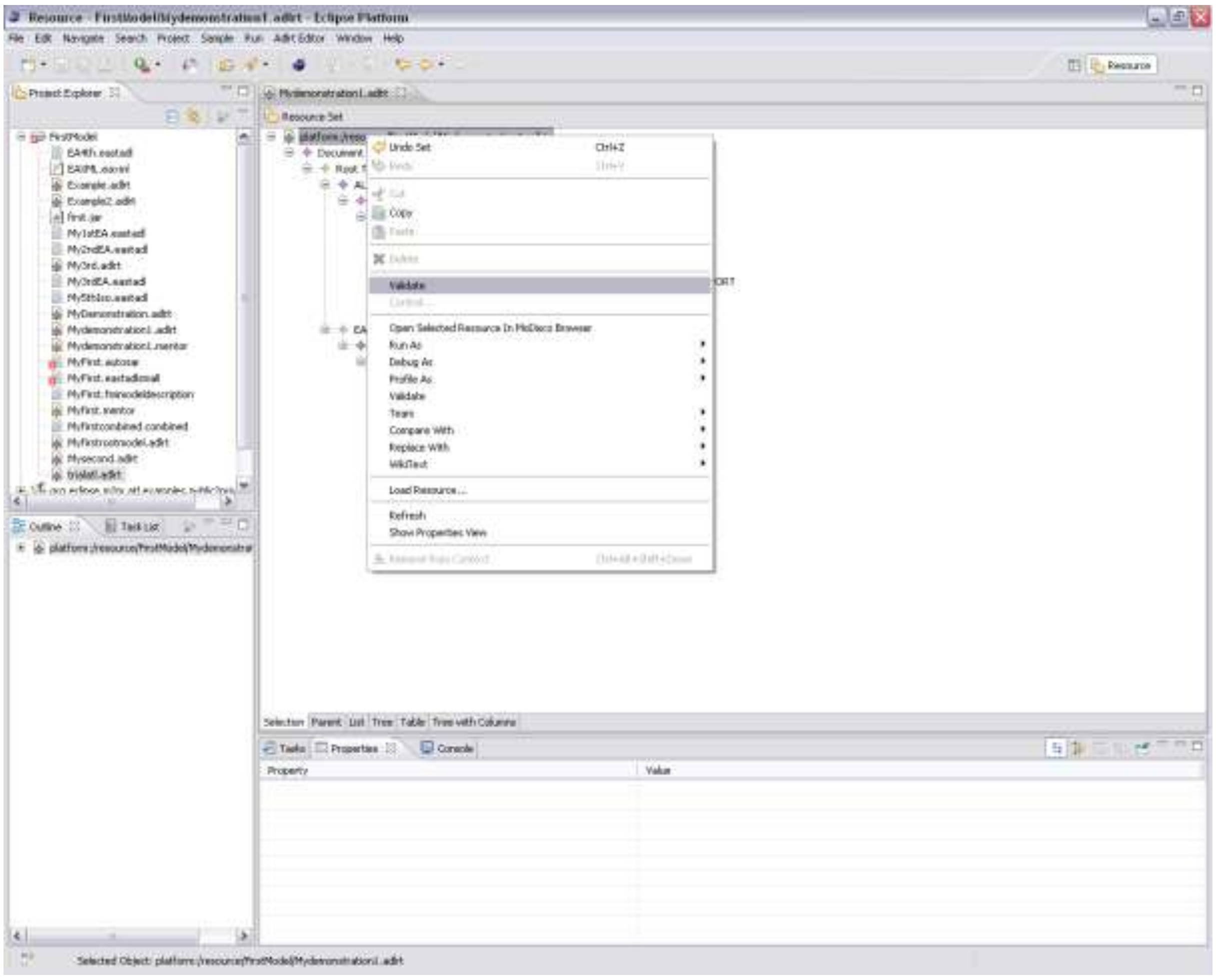
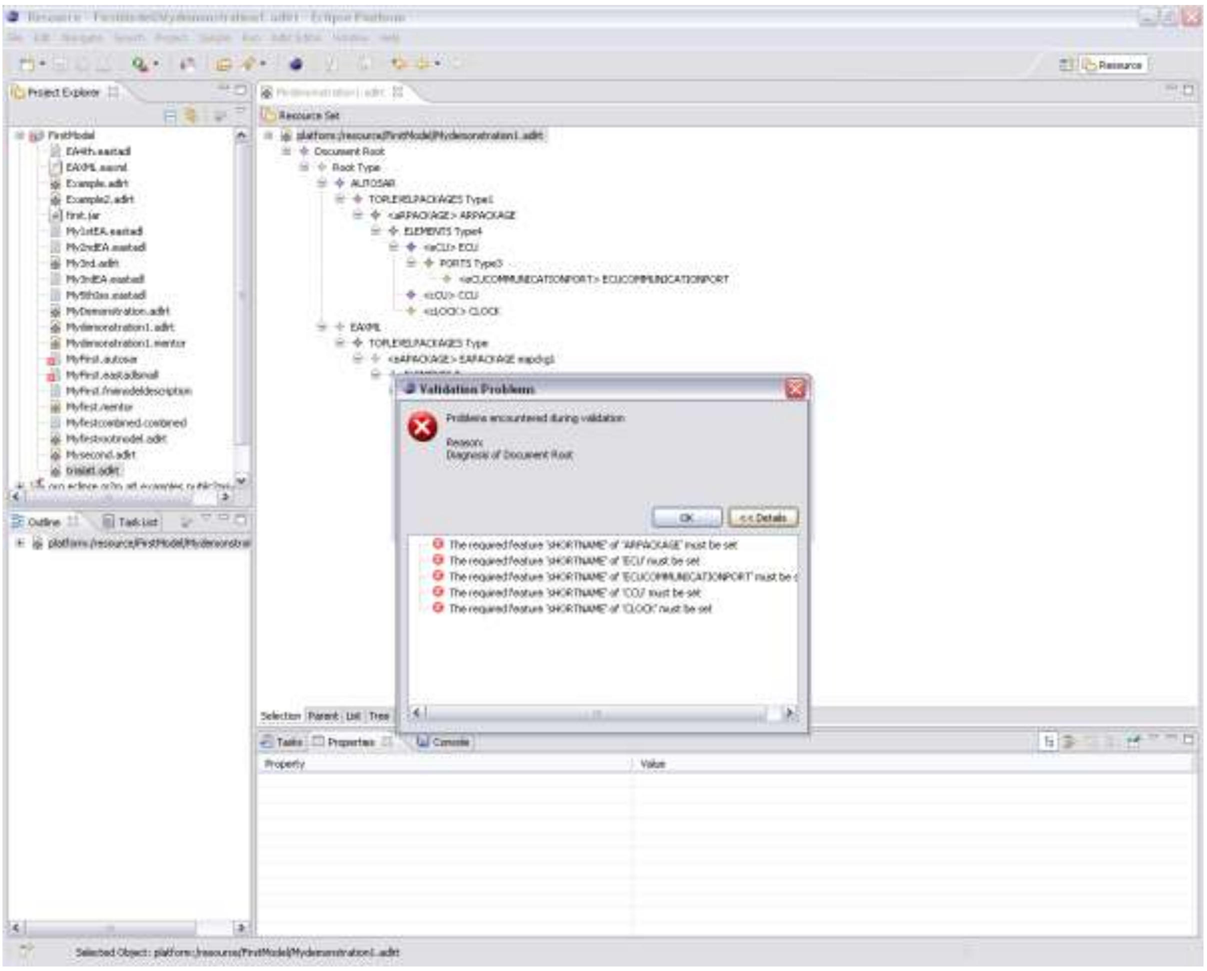

Resource - FirstModel/Mydemonstration1.adlrt - Eclipse Platform
File Edit Navigate Search Project Sample Run Adlrt Editor Window Help
Project Explorer
Mydemonstration1.adlrt
Resource Set
Undo Set Ctrl+Z
Redo Ctrl+Y
Cut
Copy
Paste
Delete
Validate
Control...
Open Selected Resource In MoDisco Browser
Run As
Debug As
Profile As
Validate
Team
Compare With
Replace With
WikiText
Load Resource...
Refresh
Show Properties View
Outline
Task List
Selection Parent List Tree Table Tree with Columns
Tasks Properties Console
Property
Value
Selected Object: platform:/resource/FirstModel/Mydemonstration1.adlrt

Figure B2.15

The validator has found some error as shown in the Figure. As seen it says that it is needed to have sHORTNAME property of the listed elements to have a valid model.



Validation Problems

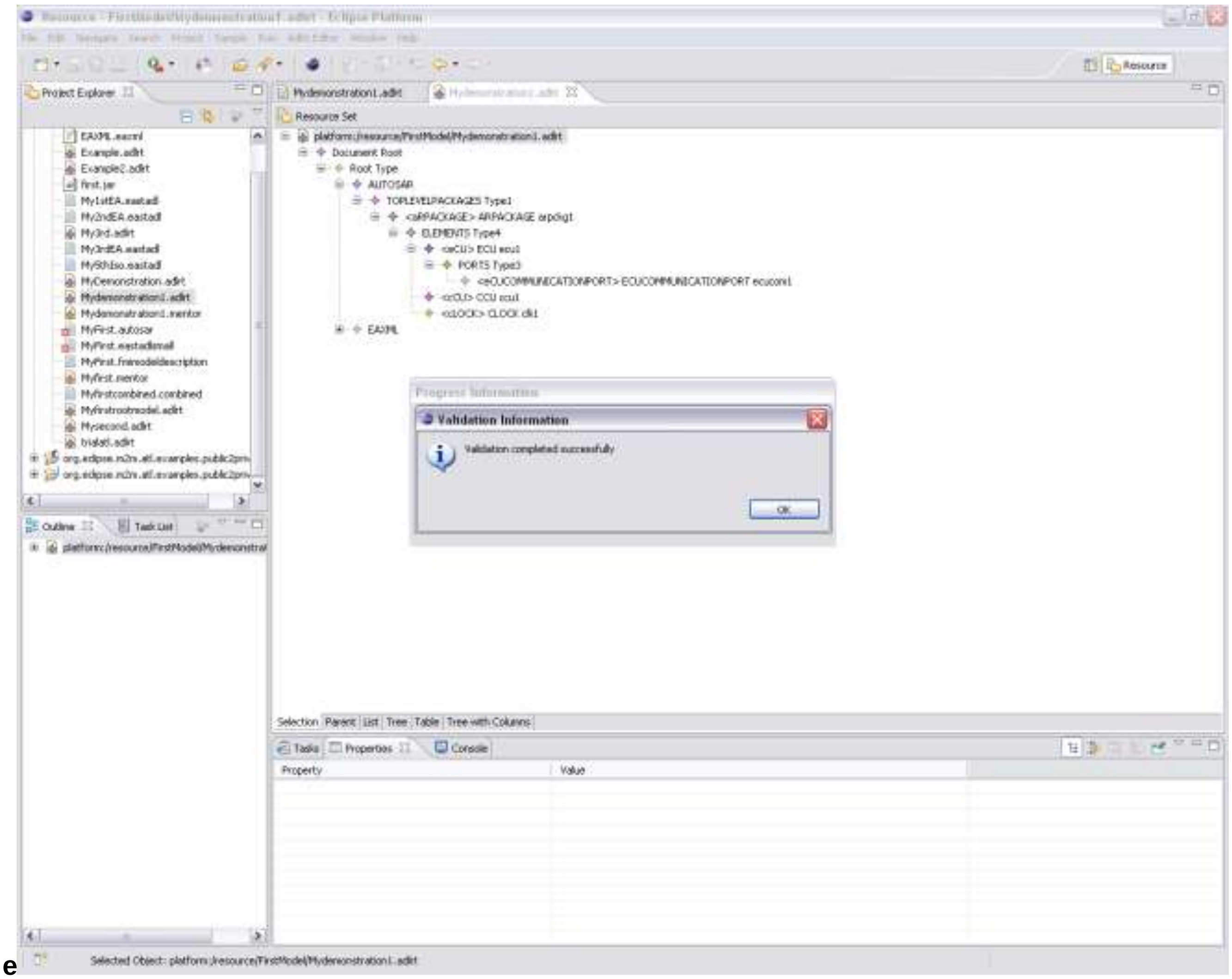
Problems encountered during validation

Reason:
Diagnosis of Document Root

The required feature 'SHORTNAME' of 'ARPACKAGE' must be set
The required feature 'SHORTNAME' of 'ECU' must be set
The required feature 'SHORTNAME' of 'ECUCOMMUNICATIONPORT' must be s
The required feature 'SHORTNAME' of 'CCU' must be set
The required feature 'SHORTNAME' of 'CLOCK' must be set

Figure B2.16

After fixing the errors in the model, Eclipse verifies the model as expected. This validation is not perfect but helps very much still. For instance it does not do type checking.



```
EAXML.eaxml
Example.adlrt
Example2.adlrt
first.jar
My1stEA.eastadl
My2ndEA.eastadl
My3rd.adlrt
My3rdEA.eastadl
My5thIso.eastadl
MyDemonstration.adlrt
Mydemonstration1.adlrt
Mydemonstration1.mentor
MyFirst.autosar
MyFirst.eastadlsmall
MyFirst.fmmodeldescription
Myfirst.mentor
Myfirstcombined.combined
Myfirstrootmodel.adlrt
Mysecond.adlrt
trialatl.adlrt
org.eclipse.m2m.atl.examples.public2priv
org.eclipse.m2m.atl.examples.public2priv
```

```
platform:/resource/FirstModel/Mydemonstration1.adlrt
  Document Root
    Root Type
      AUTOSAR
        TOPLEVELPACKAGES Type1
          <aRPACKAGE> ARPACKAGE arpdigt
            ELEMENTS Type4
              <eCU> ECU ecu1
                PORTS Type3
                  <eCUCOMMUNICATIONPORT> ECUCOMMUNICATIONPORT ecucom1
              <cCU> CCU ccu1
              <cLOCK> CLOCK clk1
      EAXML
```

Validation completed successfully

Selected Object: platform:/resource/FirstModel/Mydemonstration1.adlrt

e

Figure B2.17

The Model which has both AUTOSAR and EAST-ADL elements which is obtained using the Editor

```
<?xml version="1.0" encoding="UTF-8"?>
<adlrt:Root xmlns:adlrt="http://autosar.org/ADLRT">
 <adlrt:AUTOSAR>
  <adlrt:TOP-LEVEL-PACKAGES>
   <adlrt:AR-PACKAGE>
    <adlrt:SHORT-NAME>arpckg1</adlrt:SHORT-NAME>
    <adlrt:ELEMENTS>
     <adlrt:ECU>
      <adlrt:SHORT-NAME>ecu1</adlrt:SHORT-NAME>
      <adlrt:PORTS>
       <adlrt:ECU-COMMUNICATION-PORT>
        <adlrt:SHORT-NAME>ecucom1</adlrt:SHORT-NAME>
       </adlrt:ECU-COMMUNICATION-PORT>
      </adlrt:PORTS>
     </adlrt:ECU>
     <adlrt:CCU>
      <adlrt:SHORT-NAME>ccu1</adlrt:SHORT-NAME>
     </adlrt:CCU>
     <adlrt:CLOCK>
      <adlrt:SHORT-NAME>clk1</adlrt:SHORT-NAME>
     </adlrt:CLOCK>
    </adlrt:ELEMENTS>
   </adlrt:AR-PACKAGE>
  </adlrt:TOP-LEVEL-PACKAGES>
 </adlrt:AUTOSAR>
 <adlrt:EAXML>
  <adlrt:TOP-LEVEL-PACKAGES>
   <adlrt:EA-PACKAGE UUID="">
```

```
      <adlrt:SHORT-NAME>eapckg1</adlrt:SHORT-NAME>
      <adlrt:ELEMENTS>
        <adlrt:ANALYSIS-FUNCTION-TYPE>
          <adlrt:SHORT-NAME>aft1</adlrt:SHORT-NAME>
          <adlrt:NAME>aft1</adlrt:NAME>
          <adlrt:PORTS>
            <adlrt:FUNCTION-FLOW-PORT>
              <adlrt:SHORT-NAME>ffp1</adlrt:SHORT-NAME>
              <adlrt:NAME>ffp1</adlrt:NAME>
              <adlrt:DIRECTION>OUT</adlrt:DIRECTION>
              <adlrt:TYPE-TREF DEST="EA-INTEGER">/eaint</adlrt:TYPE-TREF>
            </adlrt:FUNCTION-FLOW-PORT>
          </adlrt:PORTS>
        </adlrt:ANALYSIS-FUNCTION-TYPE>
        <adlrt:EA-BOOLEAN>
          <adlrt:SHORT-NAME>eabln1</adlrt:SHORT-NAME>
          <adlrt:NAME>eabln1</adlrt:NAME>
        </adlrt:EA-BOOLEAN>
        <adlrt:EA-INTEGER>
          <adlrt:SHORT-NAME>eaint1</adlrt:SHORT-NAME>
          <adlrt:NAME>eaint1</adlrt:NAME>
          <adlrt:MAX>10</adlrt:MAX>
          <adlrt:MIN>0</adlrt:MIN>
        </adlrt:EA-INTEGER>
      </adlrt:ELEMENTS>
    </adlrt:EA-PACKAGE>
  </adlrt:TOP-LEVEL-PACKAGES>
 </adlrt:EAXML>
</adlrt:Root>
```

C. Appendix C

This section includes a step by step demonstration of the process taken for the generation of the FMI and EAST-ADL MetaModels, EAST-ADL Editor, FMU2EA Plug-in and FMU2EA Executable

1. Generating the MetaModels

The Figure below shows the Fmi XSD.

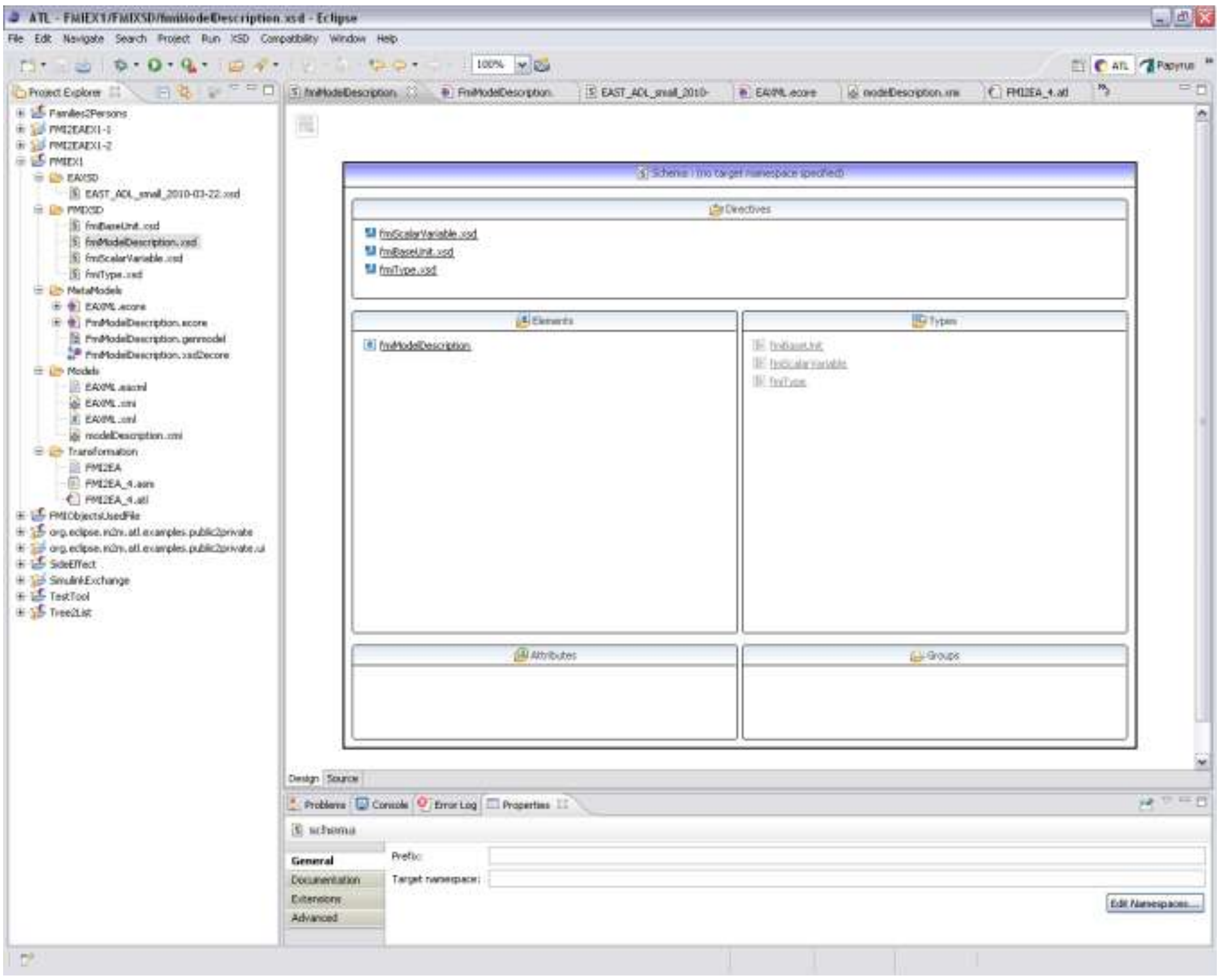


Figure C1.1

The First step to create the FMI Metamodel is shown in the figure.

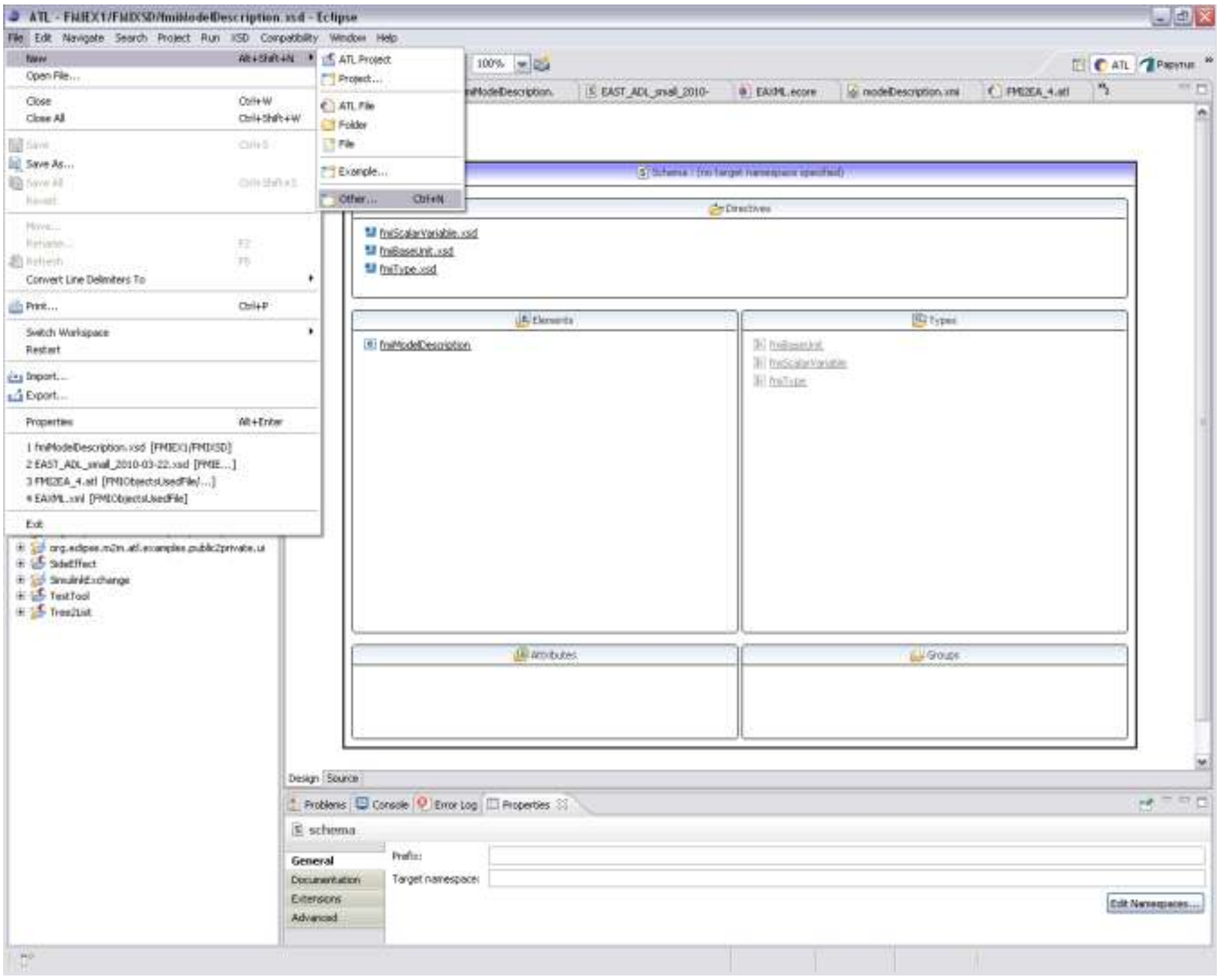
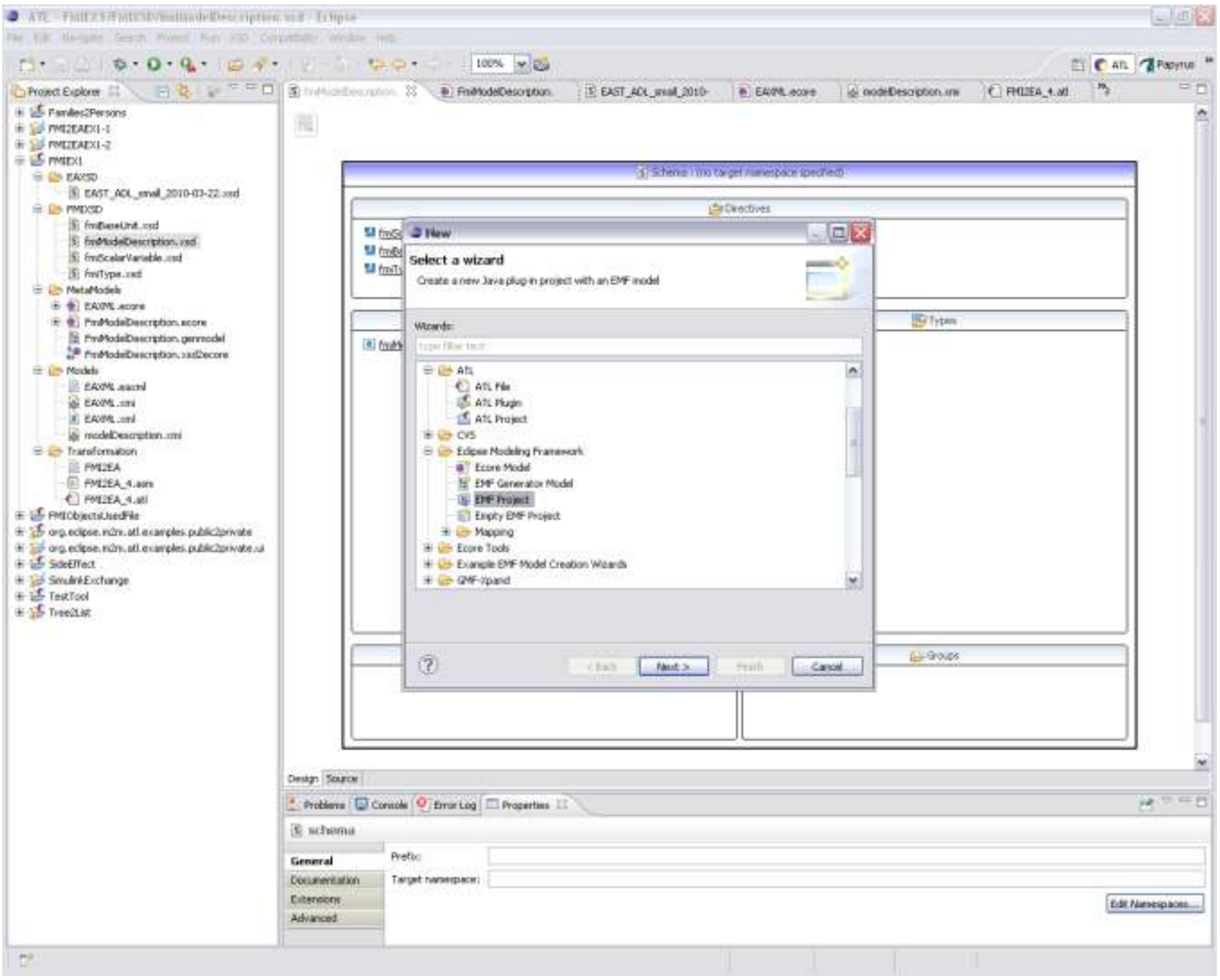


Figure C1.2

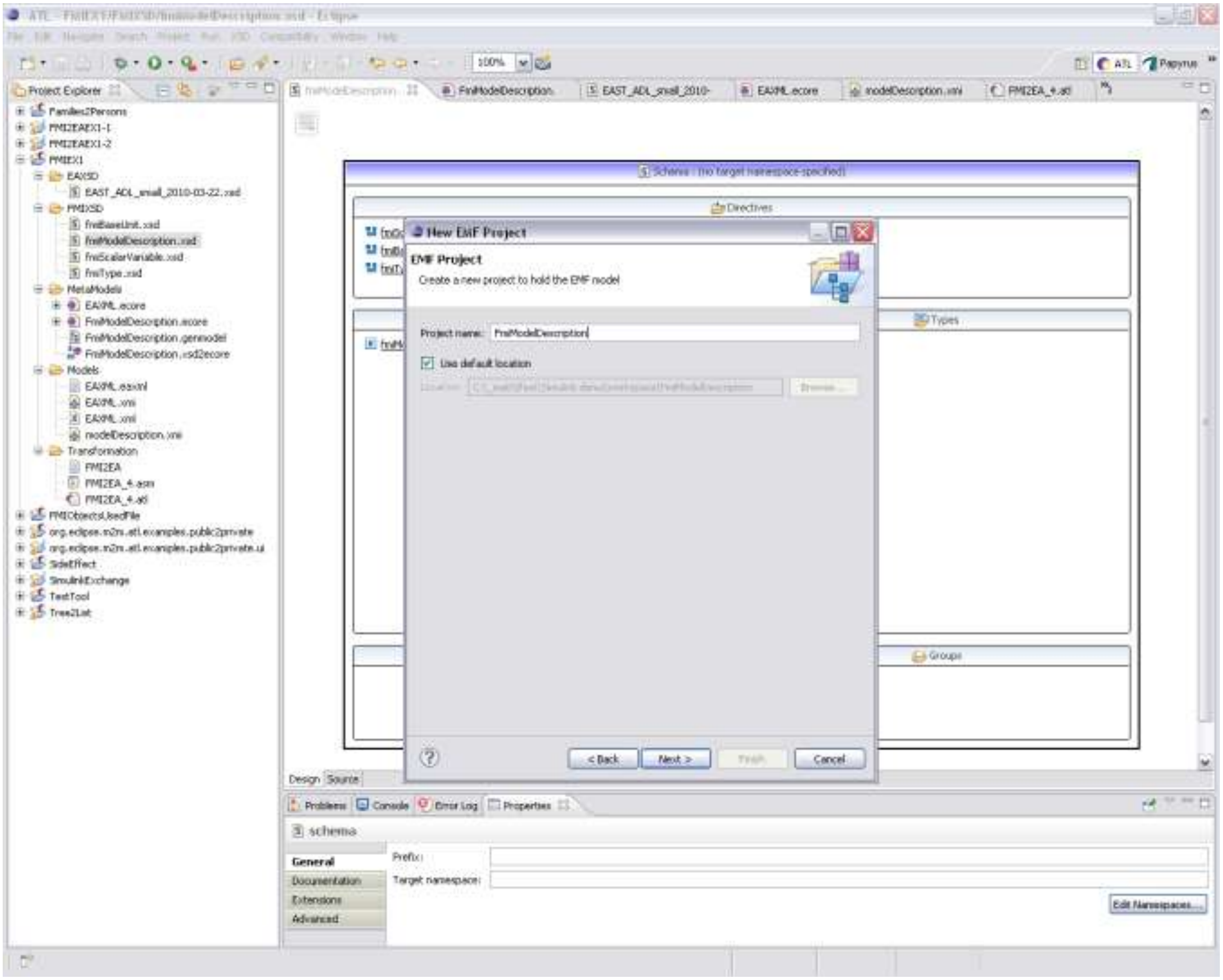
An EMF Project is created for obtaining the MetaModel.


Select a wizard

Create a new Java plug-in project with an EMF model

Figure C1.3

Name of the project is given as FmiModelDescription.



EMF Project

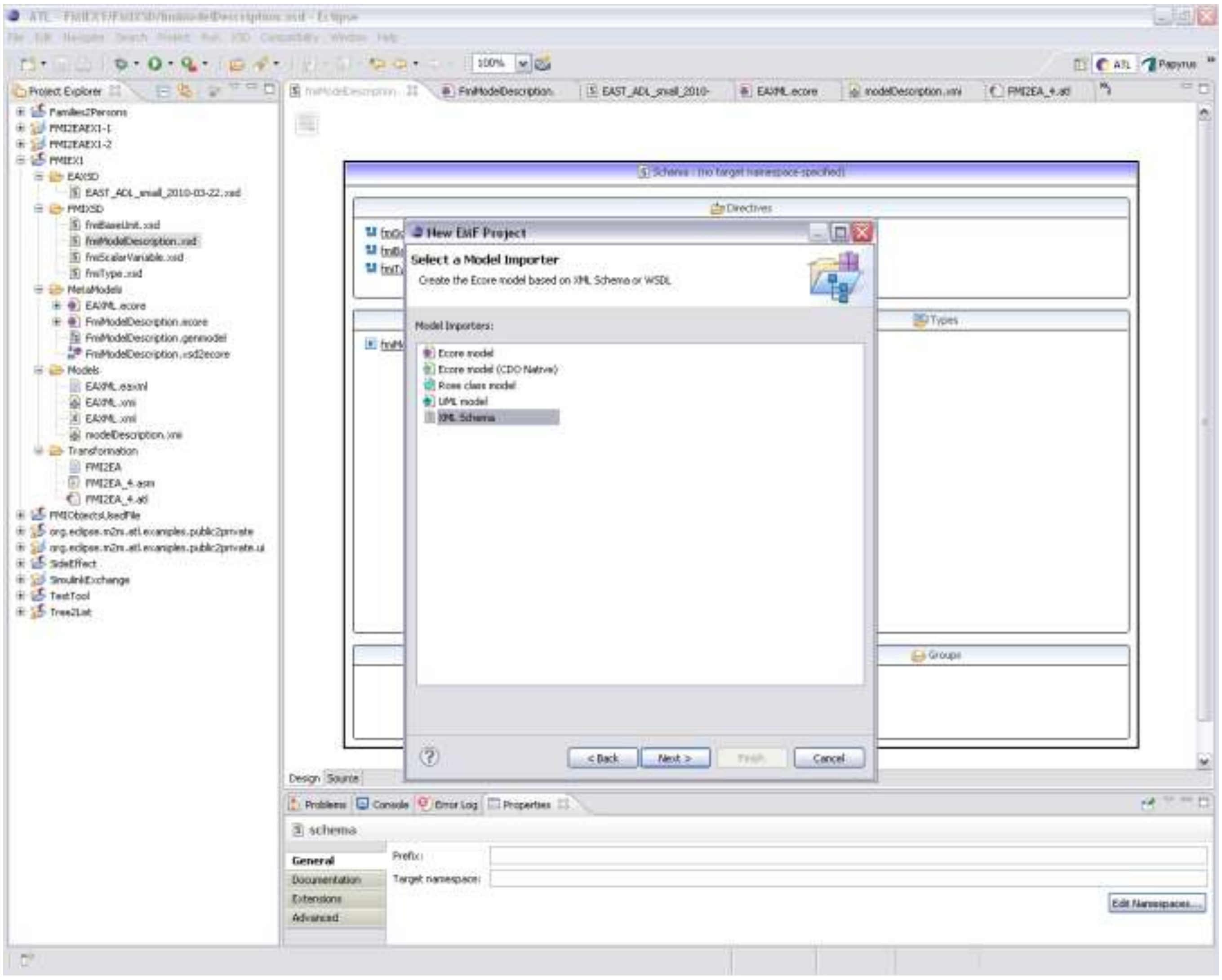
Create a new project to hold the EMF model

Project name: FmiModelDescription

[✓] Use default location

Figure C1.4

XML Schema(XSD) is used as a source for creating the MetaModel as we have the FMI XSD in hand.



New EMF Project

Select a Model Importer

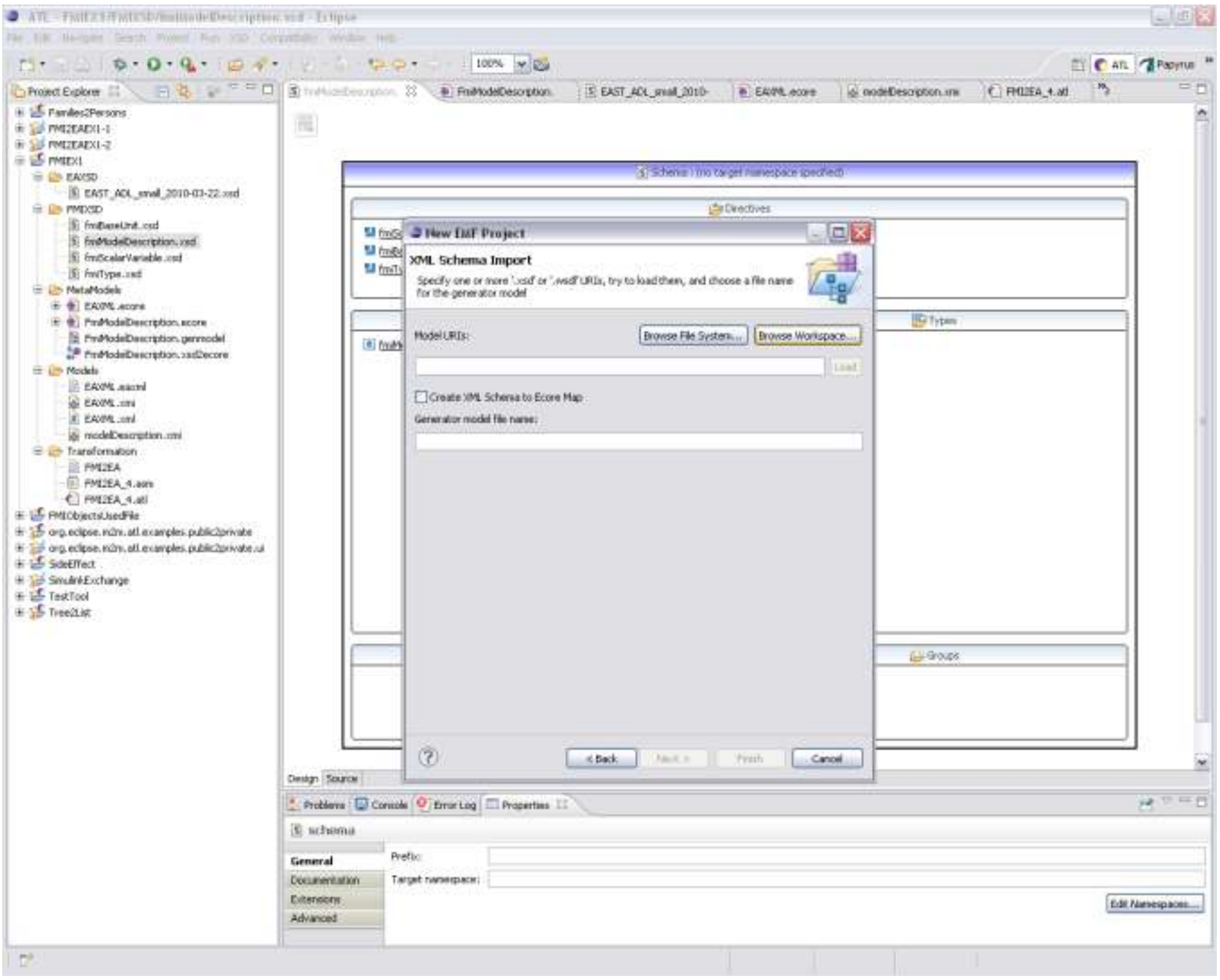
Create the Ecore model based on XML Schema or WSDL

Model Importers:

- Ecore model
- Ecore model (CDO Native)
- Rose class model
- UML model
- XML Schema

Figure C1.5

The next step is to select the FMI XSD from the file directory which is done from the menu for XML Schema Import.



Specify one or more '.xsd' or '.wsdl' URIs, try to load them, and choose a file name for the generator model

Figure C1.6

As can be seen from the figure, fmiModelDescription.XSD is chosen. There are 3 more XSDs also in the Figure which are fmiScalarVariable.XSD, fmiType.XSD, fmiBaseUnit.XSD. These are a part of the FMI schema but they are in separate files, fmiModelDescripton gives reference to these files inside. So forming an Ecore MetaModel from the fmiModelDescription.XSD file encloses all the other three.

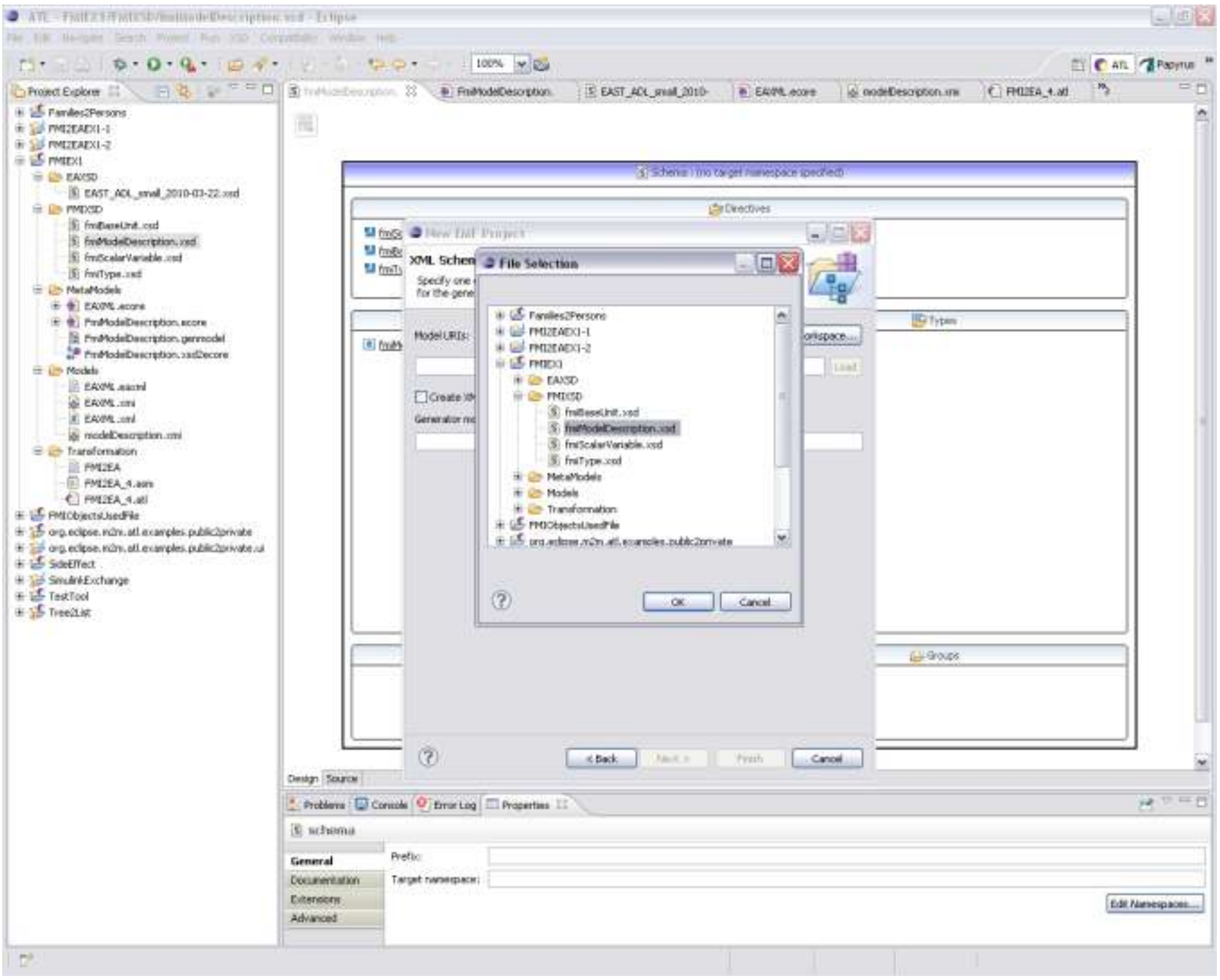
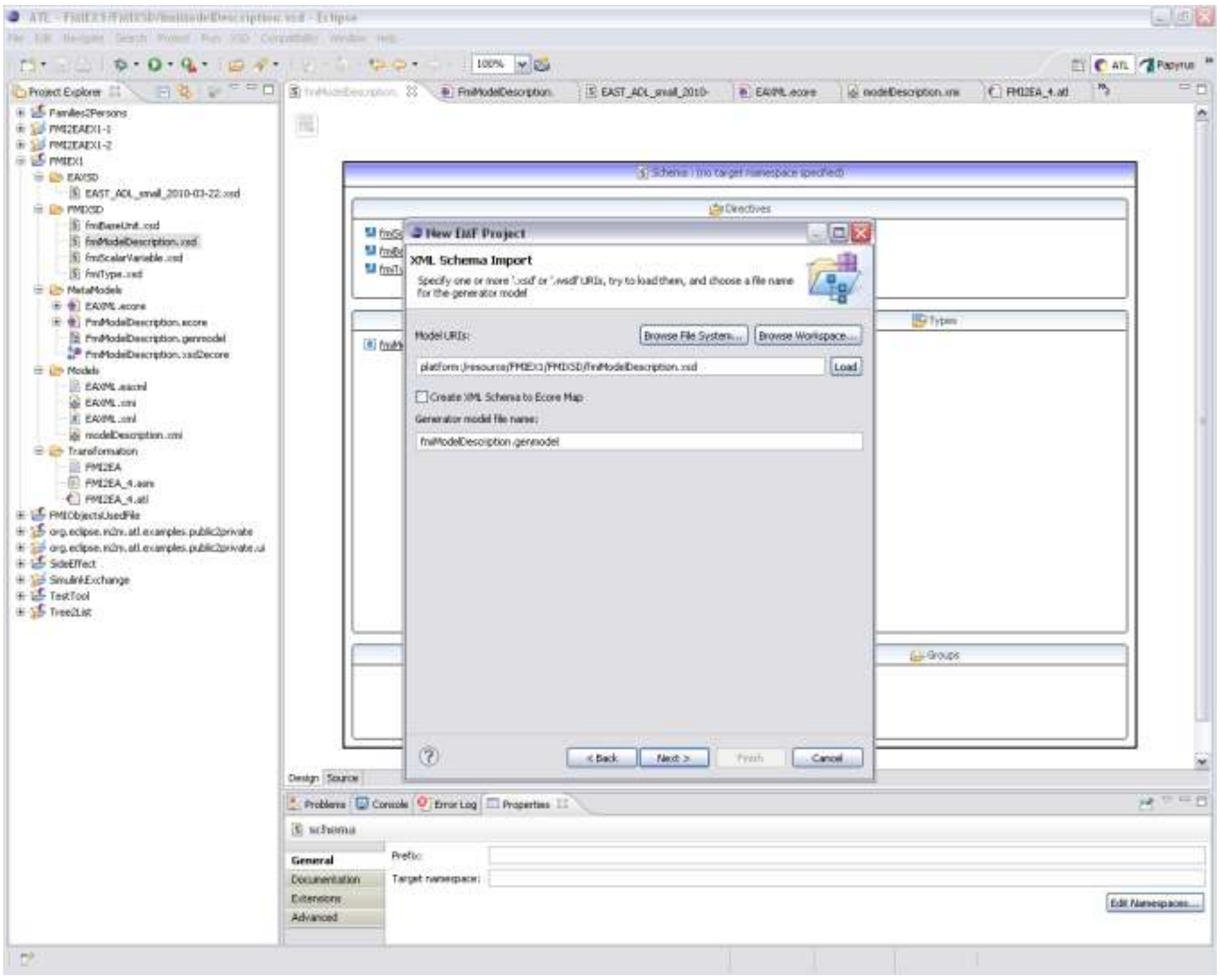


Figure C1.7

fmiModelDescription.XSD is selected. The generated models name can be changed from the menu below.


New EMF Project

XML Schema Import

Specify one or more '.xsd' or '.wsdl' URIs, try to load them, and choose a file name for the generator model

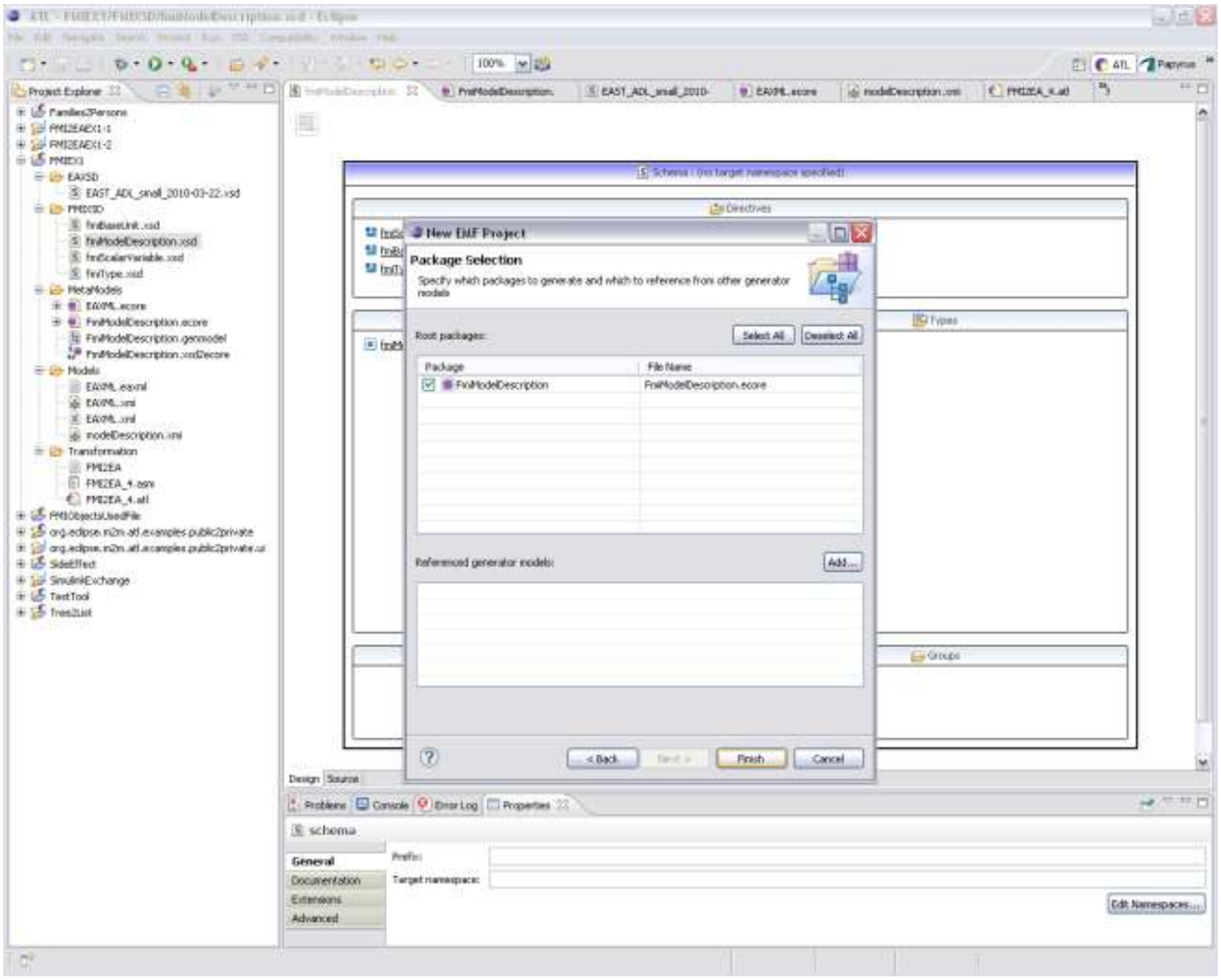
Model URIs: platform:/resource/FMIEX1/FMIXSD/fmiModelDescription.xsd

☐ Create XML Schema to Ecore Map

Generator model file name: fmiModelDescription.genmodel

Figure C1.8

The final step for generating the Ecore MetaModel. One can select the packages that is desired to be generated. Additionally, one can also give references to other previously generated MetaModels to include them in the generation process.



New EMF Project

Package Selection

Specify which packages to generate and which to reference from other generator models

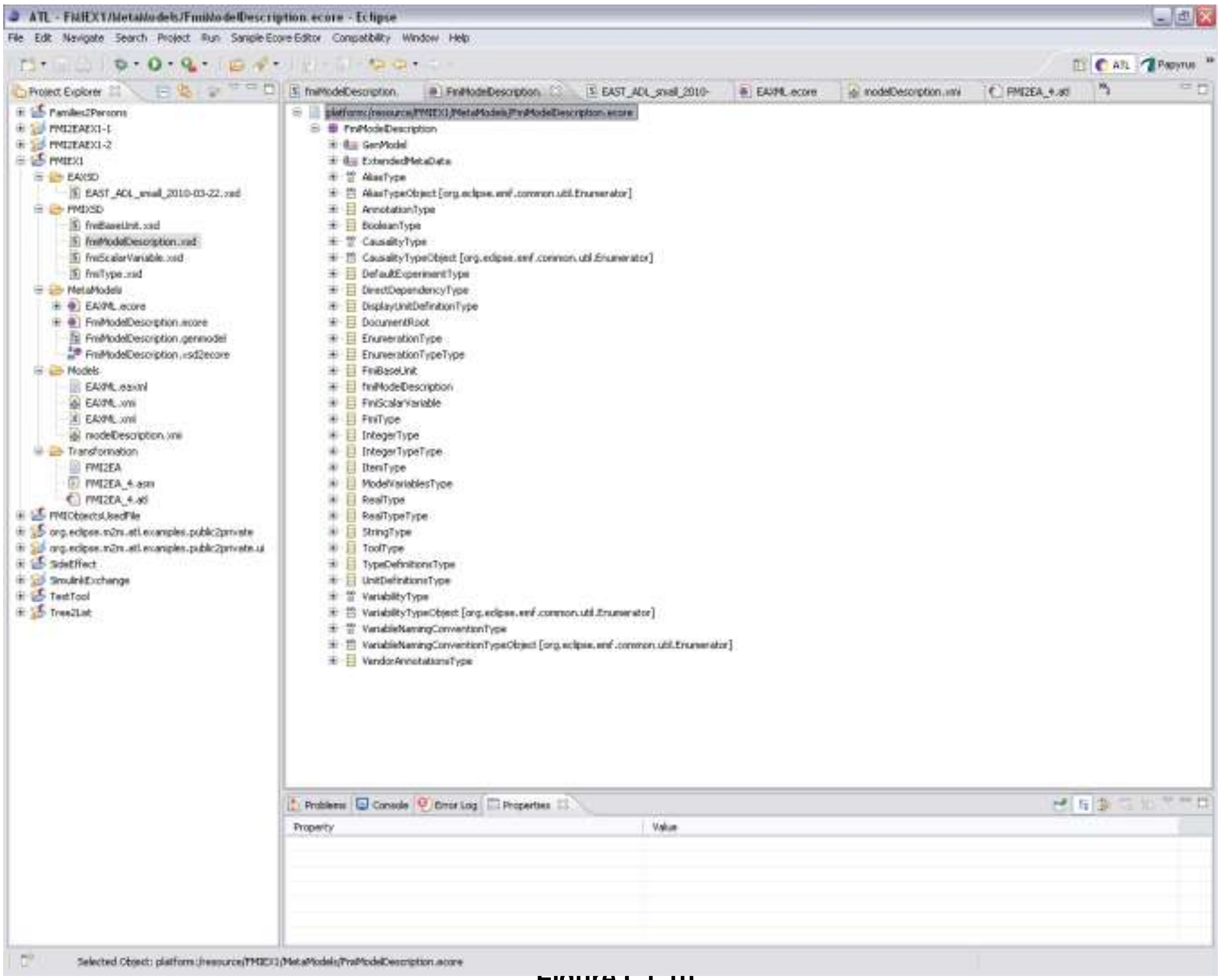
Root packages:

Package	File Name
✓ FmiModelDescription	FmiModelDescription.ecore

Referenced generator models:

Figure C1.9

The obtained Fmi Ecore MetaModel is as seen in the below Figure.



```
Project Explorer
+ Families2Persons
+ FMI2EAEX1-1
+ FMI2EAEX1-2
- FMIEX1
  - EAXSD
      EAST_ADL_small_2010-03-22.xsd
  - FMIXSD
      fmiBaseUnit.xsd
      fmiModelDescription.xsd
      fmiScalarVariable.xsd
      fmiType.xsd
  - MetaModels
    + EAXML.ecore
    + FmiModelDescription.ecore
      FmiModelDescription.genmodel
      FmiModelDescription.xsd2ecore
  - Models
      EAXML.eaxml
      EAXML.xmi
      EAXML.xml
      modelDescription.xmi
  - Transformation
      FMI2EA
      FMI2EA_4.asm
      FMI2EA_4.atl
+ FMIObjectsUsedFile
+ org.eclipse.m2m.atl.examples.public2private
+ org.eclipse.m2m.atl.examples.public2private.ui
+ SideEffect
+ SimulinkExchange
+ TestTool
+ Tree2List
```

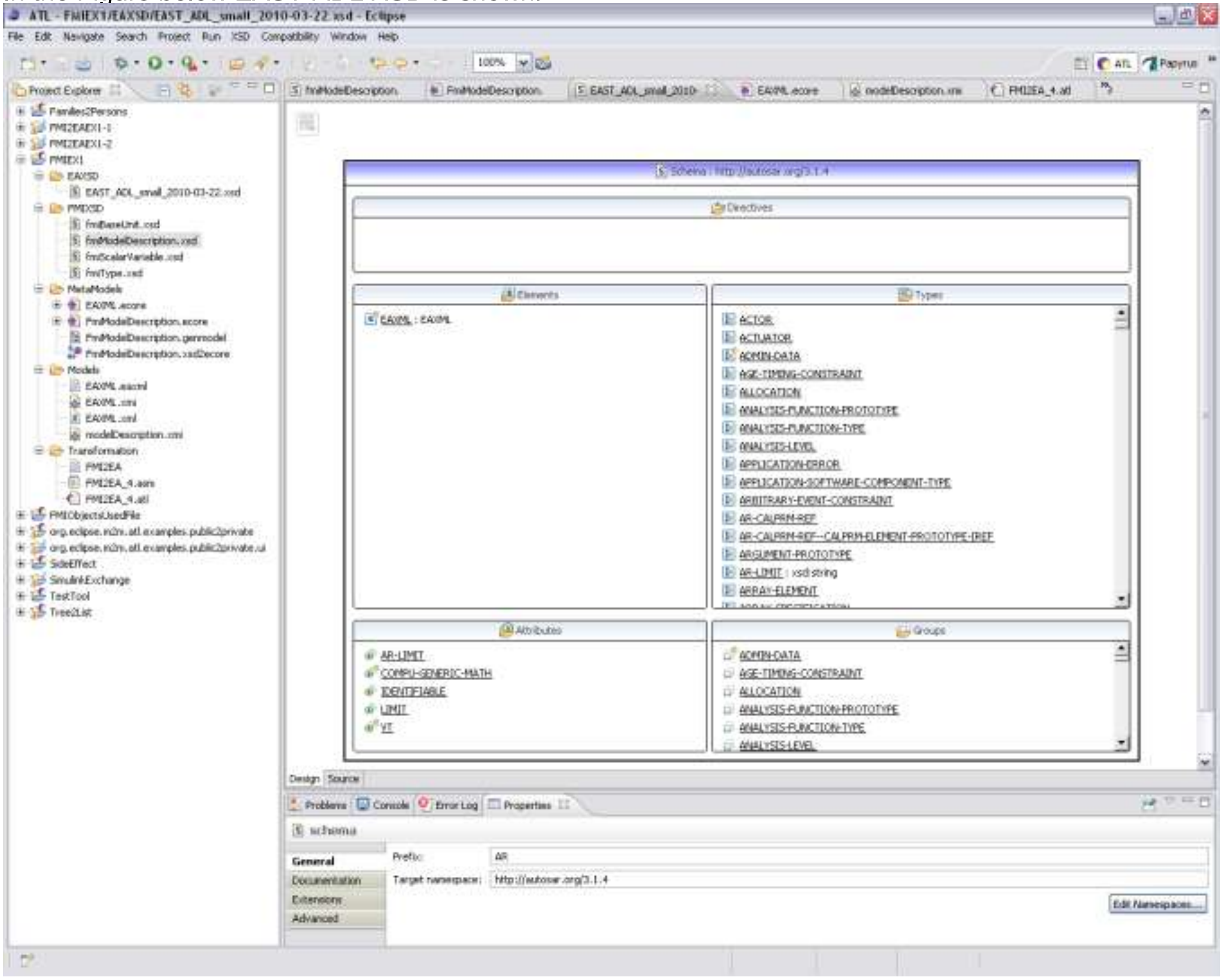
```
- platform:/resource/FMIEX1/MetaModels/FmiModelDescription.ecore
  - FmiModelDescription
    + GenModel
    + ExtendedMetaData
    + AliasType
    + AliasTypeObject [org.eclipse.emf.common.util.Enumerator]
    + AnnotationType
    + BooleanType
    + CausalityType
    + CausalityTypeObject [org.eclipse.emf.common.util.Enumerator]
    + DefaultExperimentType
    + DirectDependencyType
    + DisplayUnitDefinitionType
    + DocumentRoot
    + EnumerationType
    + EnumerationTypeType
    + FmiBaseUnit
    + fmiModelDescription
    + FmiScalarVariable
    + FmiType
    + IntegerType
    + IntegerTypeType
    + ItemType
    + ModelVariablesType
    + RealType
    + RealTypeType
    + StringType
    + ToolType
    + TypeDefinitionsType
    + UnitDefinitionsType
    + VariabilityType
    + VariabilityTypeObject [org.eclipse.emf.common.util.Enumerator]
    + VariableNamingConventionType
    + VariableNamingConventionTypeObject [org.eclipse.emf.common.util.Enumerator]
    + VendorAnnotationsType
```

Selected Object: platform:/resource/FMIEX1/MetaModels/FmiModelDescription.ecore

ATL - FMIEX1/MetaModels/FmiModelDescription.ecore - Eclipse
File Edit Navigate Search Project Run Sample Ecore Editor Compatibility Window Help
ATL
Papyrus
fmiModelDescription
FmiModelDescription
EAST_ADL_small_2010-
EAXML.ecore
modelDescription.xmi
FMI2EA_4.atl
Problems Console Error Log Properties
Property
Value

Figure C1.10

In the Figure below EAST-ADL XSD is shown.



Elements:
EAXML : EAXML

Types:
ACTOR
ACTUATOR
ADMIN-DATA
AGE-TIMING-CONSTRAINT
ALLOCATION
ANALYSIS-FUNCTION-PROTOTYPE
ANALYSIS-FUNCTION-TYPE
ANALYSIS-LEVEL
APPLICATION-ERROR
APPLICATION-SOFTWARE-COMPONENT-TYPE
ARBITRARY-EVENT-CONSTRAINT
AR-CALPRM-REF
AR-CALPRM-REF--CALPRM-ELEMENT-PROTOTYPE-IREF
ARGUMENT-PROTOTYPE
AR-LIMIT : xsd:string
ARRAY-ELEMENT

Attributes:
AR-LIMIT
COMPU-GENERIC-MATH
IDENTIFIABLE
LIMIT
VT

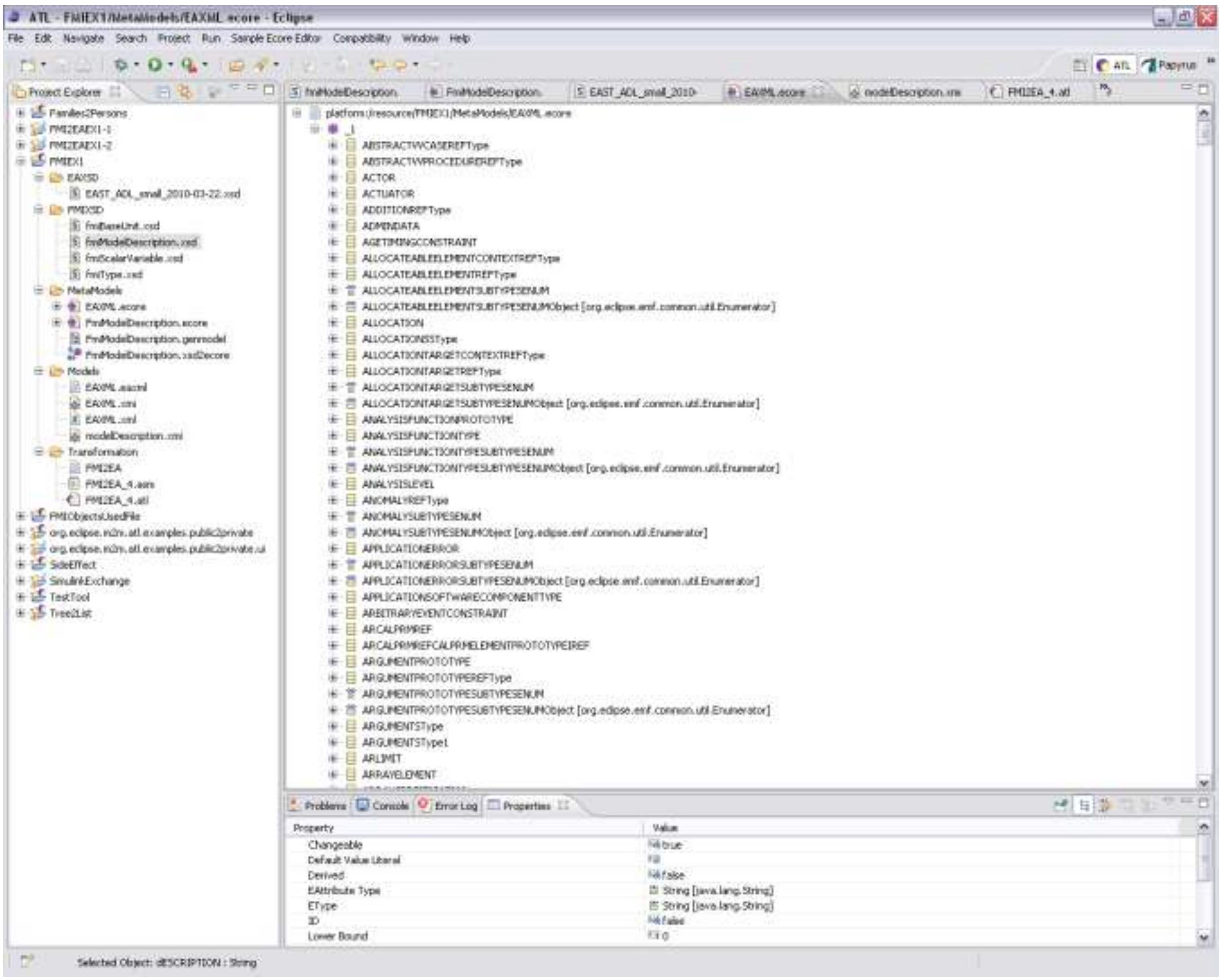
Groups:
ADMIN-DATA
AGE-TIMING-CONSTRAINT
ALLOCATION
ANALYSIS-FUNCTION-PROTOTYPE
ANALYSIS-FUNCTION-TYPE
ANALYSIS-LEVEL

schema

Prefix: AR
Target namespace: http://autosar.org/3.1.4

Figure C1.11

Applying the same steps above that is used for obtaining the FMI Ecore MetaModel to the EAST-ADL XSD results in EAST-ADL Ecore MetaModel as can be seen from the below Figure.



```
Families2Persons
FMI2EAEX1-1
FMI2EAEX1-2
FMIEX1
  EAXSD
    EAST_ADL_small_2010-03-22.xsd
  FMIXSD
    fmiBaseUnit.xsd
    fmiModelDescription.xsd
    fmiScalarVariable.xsd
    fmiType.xsd
  MetaModels
    EAXML.ecore
    FmiModelDescription.ecore
    FmiModelDescription.genmodel
    FmiModelDescription.xsd2ecore
  Models
    EAXML.eacml
    EAXML.xmi
    EAXML.xml
    modelDescription.xmi
  Transformation
    FMI2EA
    FMI2EA_4.asm
    FMI2EA_4.atl
FMIObjectsUsedFile
org.eclipse.m2m.atl.examples.public2private
org.eclipse.m2m.atl.examples.public2private.ui
SideEffect
SimulinkExchange
TestTool
Tree2List
```

```
platform:/resource/FMIEX1/MetaModels/EAXML.ecore
  _1
    ABSTRACTVVCASEREFType
    ABSTRACTVVPROCEDUREREFType
    ACTOR
    ACTUATOR
    ADDITIONREFType
    ADMENDATA
    AGETIMINGCONSTRAINT
    ALLOCATEABLEELEMENTCONTEXTREFType
    ALLOCATEABLEELEMENTREFType
    ALLOCATEABLEELEMENTSUBTYPESENUM
    ALLOCATEABLEELEMENTSUBTYPESENUMObject [org.eclipse.emf.common.util.Enumerator]
    ALLOCATION
    ALLOCATIONSSType
    ALLOCATIONTARGETCONTEXTREFType
    ALLOCATIONTARGETREFType
    ALLOCATIONTARGETSUBTYPESENUM
    ALLOCATIONTARGETSUBTYPESENUMObject [org.eclipse.emf.common.util.Enumerator]
    ANALYSISFUNCTIONPROTOTYPE
    ANALYSISFUNCTIONTYPE
    ANALYSISFUNCTIONTYPESUBTYPESENUM
    ANALYSISFUNCTIONTYPESUBTYPESENUMObject [org.eclipse.emf.common.util.Enumerator]
    ANALYSISLEVEL
    ANOMALYREFType
    ANOMALYSUBTYPESENUM
    ANOMALYSUBTYPESENUMObject [org.eclipse.emf.common.util.Enumerator]
    APPLICATIONERROR
    APPLICATIONERRORSUBTYPESENUM
    APPLICATIONERRORSUBTYPESENUMObject [org.eclipse.emf.common.util.Enumerator]
    APPLICATIONSOFTWARECOMPONENTTYPE
    ARBITRARYEVENTCONSTRAINT
    ARCALPRIMREF
    ARCALPRIMREFCALPRMELEMENTPROTOTYPEIREF
    ARGUMENTPROTOTYPE
    ARGUMENTPROTOTYPEREFType
    ARGUMENTPROTOTYPESUBTYPESENUM
    ARGUMENTPROTOTYPESUBTYPESENUMObject [org.eclipse.emf.common.util.Enumerator]
    ARGUMENTSType
    ARGUMENTSType1
    ARLIMIT
    ARRAYELEMENT
```

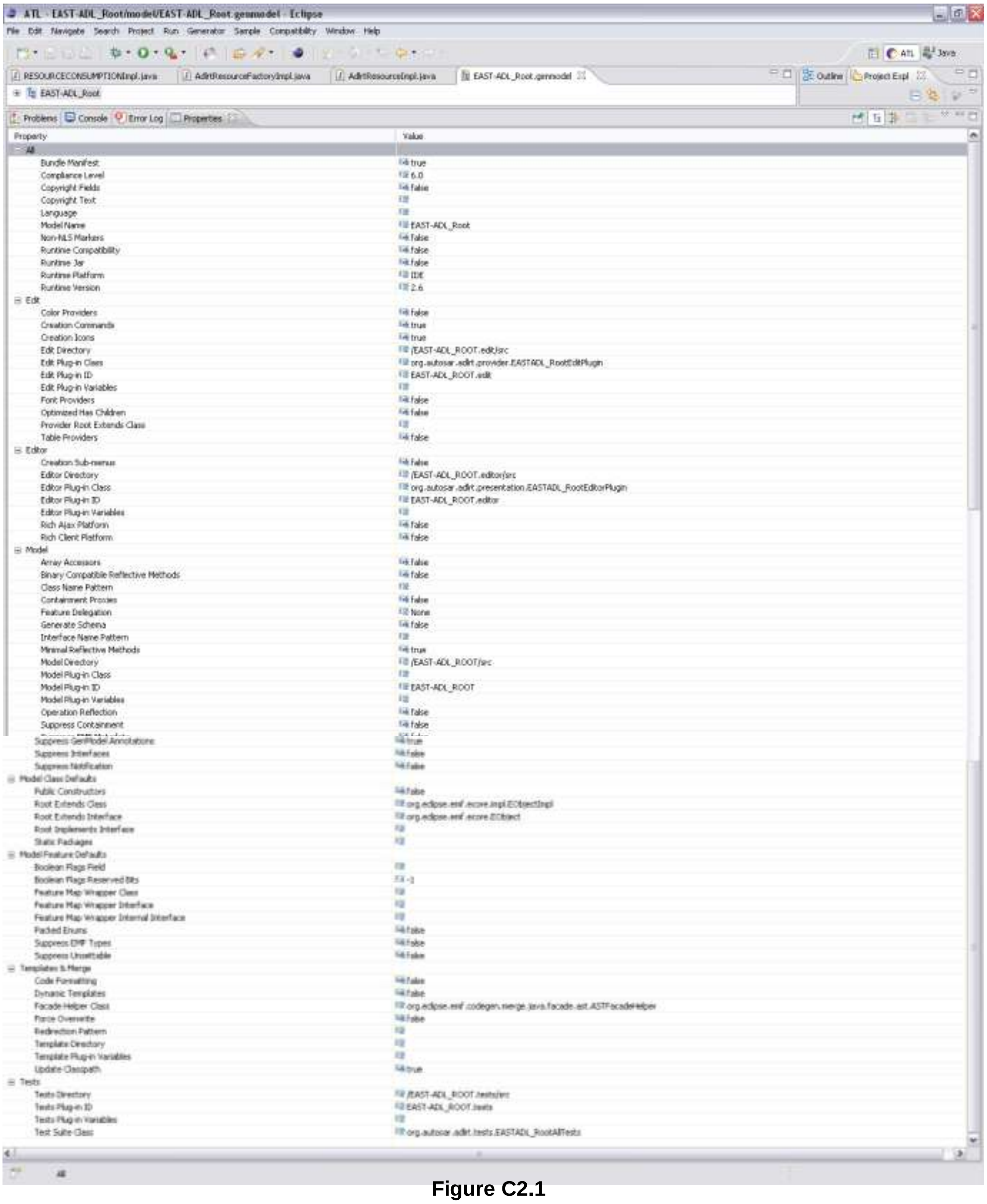
Property	Value
Changeable	true
Default Value Literal	
Derived	false
EAttribute Type	String [java.lang.String]
EType	String [java.lang.String]
ID	false
Lower Bound	0

Selected Object: dESCRIPTION : String

Figure C1.12

2. Obtaining the Editor

Figure C2.1 shows the options used to generate the Editor. Changes in the options can result in different Editors.

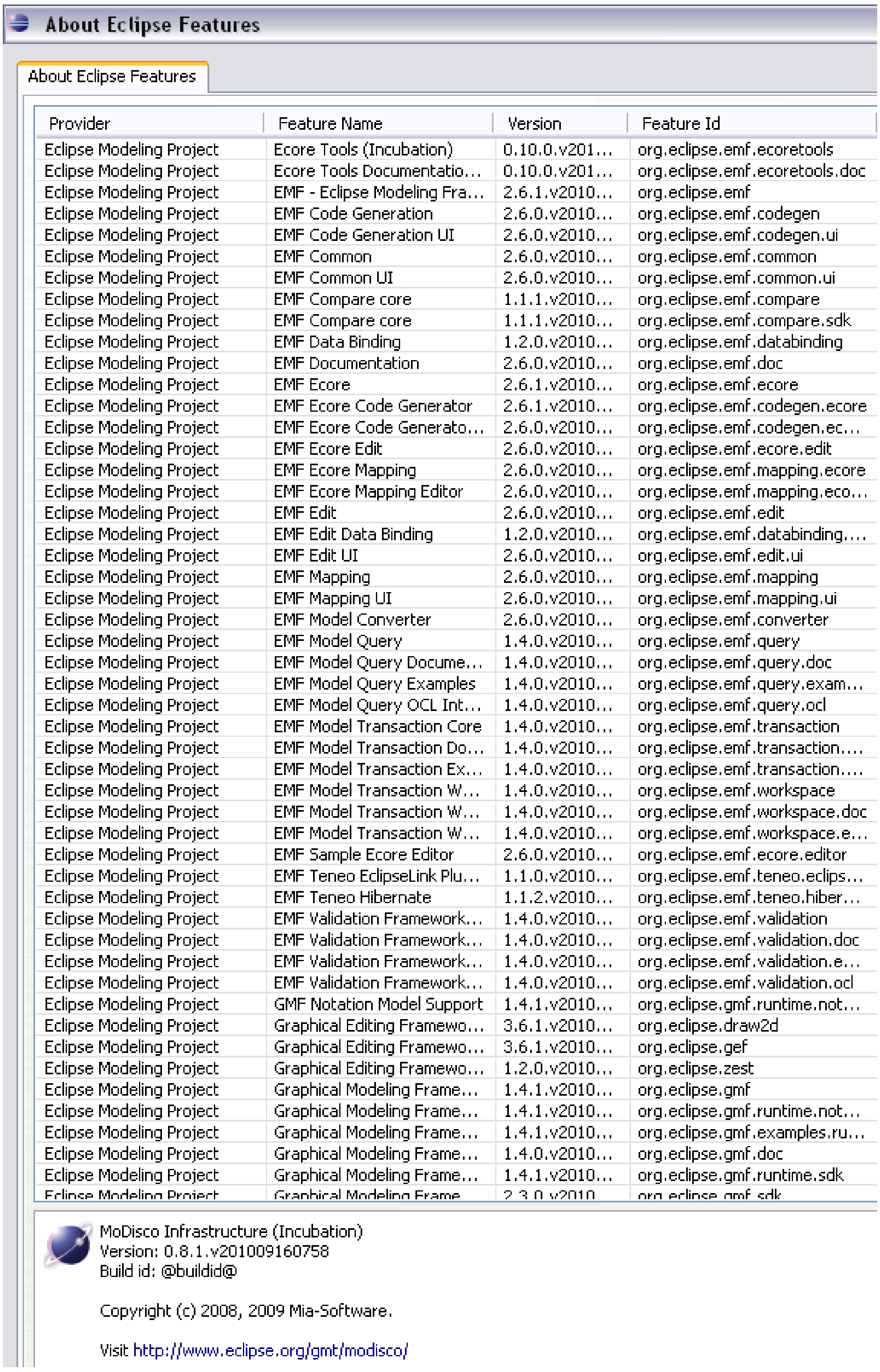


ATL - EAST-ADL_Root/model/EAST-ADL_Root.genmodel - Eclipse

Property	Value
All	
Bundle Manifest	true
Compliance Level	6.0
Copyright Fields	false
Copyright Text	
Language	
Model Name	EAST-ADL_Root
Non-NLS Markers	false
Runtime Compatibility	false
Runtime Jar	false
Runtime Platform	IDE
Runtime Version	2.6
Edit	
Color Providers	false
Creation Commands	true
Creation Icons	true
Edit Directory	/EAST-ADL_ROOT.edit/src
Edit Plug-in Class	org.autosar.adlt.provider.EASTADL_RootEditPlugin
Edit Plug-in ID	EAST-ADL_ROOT.edit
Edit Plug-in Variables	
Font Providers	false
Optimized Has Children	false
Provider Root Extends Class	
Table Providers	false
Editor	
Creation Sub-menus	false
Editor Directory	/EAST-ADL_ROOT.editor/src
Editor Plug-in Class	org.autosar.adlt.presentation.EASTADL_RootEditorPlugin
Editor Plug-in ID	EAST-ADL_ROOT.editor
Editor Plug-in Variables	
Rich Ajax Platform	false
Rich Client Platform	false
Model	
Array Accessors	false
Binary Compatible Reflective Methods	false
Class Name Pattern	
Containment Proxies	false
Feature Delegation	None
Generate Schema	false
Interface Name Pattern	
Minimal Reflective Methods	true
Model Directory	/EAST-ADL_ROOT/src
Model Plug-in Class	
Model Plug-in ID	EAST-ADL_ROOT
Model Plug-in Variables	
Operation Reflection	false
Suppress Containment	false
Suppress EMF Metadata	false
Suppress GenModel Annotations	true
Suppress Interfaces	false
Suppress Notification	false
Model Class Defaults	
Public Constructors	false
Root Extends Class	org.eclipse.emf.ecore.impl.EObjectImpl
Root Extends Interface	org.eclipse.emf.ecore.EObject
Root Implements Interface	
Static Packages	
Model Feature Defaults	
Boolean Flags Field	
Boolean Flags Reserved Bits	-1
Feature Map Wrapper Class	
Feature Map Wrapper Interface	
Feature Map Wrapper Internal Interface	
Packed Enums	false
Suppress EMF Types	false
Suppress Unsettable	false
Templates & Merge	
Code Formatting	false
Dynamic Templates	false
Facade Helper Class	org.eclipse.emf.codegen.merge.java.facade.ast.ASTFacadeHelper
Force Overwrite	false
Redirection Pattern	
Template Directory	
Template Plug-in Variables	
Update Classpath	true
Tests	
Tests Directory	/EAST-ADL_ROOT.tests/src
Tests Plug-in ID	EAST-ADL_ROOT.tests
Tests Plug-in Variables	
Test Suite Class	org.autosar.adlt.tests.EASTADL_RootAllTests

Figure C2.1

The below figure shows the versions of the Eclipse Features installed in the Eclipse Helios platform which is used for obtaining the Editor.



About Eclipse Features

About Eclipse Features

Provider	Feature Name	Version	Feature Id
Eclipse Modeling Project	Ecore Tools (Incubation)	0.10.0.v201...	org.eclipse.emf.ecoretools
Eclipse Modeling Project	Ecore Tools Documentatio...	0.10.0.v201...	org.eclipse.emf.ecoretools.doc
Eclipse Modeling Project	EMF - Eclipse Modeling Fra...	2.6.1.v2010...	org.eclipse.emf
Eclipse Modeling Project	EMF Code Generation	2.6.0.v2010...	org.eclipse.emf.codegen
Eclipse Modeling Project	EMF Code Generation UI	2.6.0.v2010...	org.eclipse.emf.codegen.ui
Eclipse Modeling Project	EMF Common	2.6.0.v2010...	org.eclipse.emf.common
Eclipse Modeling Project	EMF Common UI	2.6.0.v2010...	org.eclipse.emf.common.ui
Eclipse Modeling Project	EMF Compare core	1.1.1.v2010...	org.eclipse.emf.compare
Eclipse Modeling Project	EMF Compare core	1.1.1.v2010...	org.eclipse.emf.compare.sdk
Eclipse Modeling Project	EMF Data Binding	1.2.0.v2010...	org.eclipse.emf.databinding
Eclipse Modeling Project	EMF Documentation	2.6.0.v2010...	org.eclipse.emf.doc
Eclipse Modeling Project	EMF Ecore	2.6.1.v2010...	org.eclipse.emf.ecore
Eclipse Modeling Project	EMF Ecore Code Generator	2.6.1.v2010...	org.eclipse.emf.codegen.ecore
Eclipse Modeling Project	EMF Ecore Code Generato...	2.6.0.v2010...	org.eclipse.emf.codegen.ec...
Eclipse Modeling Project	EMF Ecore Edit	2.6.0.v2010...	org.eclipse.emf.ecore.edit
Eclipse Modeling Project	EMF Ecore Mapping	2.6.0.v2010...	org.eclipse.emf.mapping.ecore
Eclipse Modeling Project	EMF Ecore Mapping Editor	2.6.0.v2010...	org.eclipse.emf.mapping.eco...
Eclipse Modeling Project	EMF Edit	2.6.0.v2010...	org.eclipse.emf.edit
Eclipse Modeling Project	EMF Edit Data Binding	1.2.0.v2010...	org.eclipse.emf.databinding....
Eclipse Modeling Project	EMF Edit UI	2.6.0.v2010...	org.eclipse.emf.edit.ui
Eclipse Modeling Project	EMF Mapping	2.6.0.v2010...	org.eclipse.emf.mapping
Eclipse Modeling Project	EMF Mapping UI	2.6.0.v2010...	org.eclipse.emf.mapping.ui
Eclipse Modeling Project	EMF Model Converter	2.6.0.v2010...	org.eclipse.emf.converter
Eclipse Modeling Project	EMF Model Query	1.4.0.v2010...	org.eclipse.emf.query
Eclipse Modeling Project	EMF Model Query Docume...	1.4.0.v2010...	org.eclipse.emf.query.doc
Eclipse Modeling Project	EMF Model Query Examples	1.4.0.v2010...	org.eclipse.emf.query.exam...
Eclipse Modeling Project	EMF Model Query OCL Int...	1.4.0.v2010...	org.eclipse.emf.query.ocl
Eclipse Modeling Project	EMF Model Transaction Core	1.4.0.v2010...	org.eclipse.emf.transaction
Eclipse Modeling Project	EMF Model Transaction Do...	1.4.0.v2010...	org.eclipse.emf.transaction....
Eclipse Modeling Project	EMF Model Transaction Ex...	1.4.0.v2010...	org.eclipse.emf.transaction....
Eclipse Modeling Project	EMF Model Transaction W...	1.4.0.v2010...	org.eclipse.emf.workspace
Eclipse Modeling Project	EMF Model Transaction W...	1.4.0.v2010...	org.eclipse.emf.workspace.doc
Eclipse Modeling Project	EMF Model Transaction W...	1.4.0.v2010...	org.eclipse.emf.workspace.e...
Eclipse Modeling Project	EMF Sample Ecore Editor	2.6.0.v2010...	org.eclipse.emf.ecore.editor
Eclipse Modeling Project	EMF Teneo EclipseLink Plu...	1.1.0.v2010...	org.eclipse.emf.teneo.eclips...
Eclipse Modeling Project	EMF Teneo Hibernate	1.1.2.v2010...	org.eclipse.emf.teneo.hiber...
Eclipse Modeling Project	EMF Validation Framework...	1.4.0.v2010...	org.eclipse.emf.validation
Eclipse Modeling Project	EMF Validation Framework...	1.4.0.v2010...	org.eclipse.emf.validation.doc
Eclipse Modeling Project	EMF Validation Framework...	1.4.0.v2010...	org.eclipse.emf.validation.e...
Eclipse Modeling Project	EMF Validation Framework...	1.4.0.v2010...	org.eclipse.emf.validation.ocl
Eclipse Modeling Project	GMF Notation Model Support	1.4.1.v2010...	org.eclipse.gmf.runtime.not...
Eclipse Modeling Project	Graphical Editing Framewo...	3.6.1.v2010...	org.eclipse.draw2d
Eclipse Modeling Project	Graphical Editing Framewo...	3.6.1.v2010...	org.eclipse.gef
Eclipse Modeling Project	Graphical Editing Framewo...	1.2.0.v2010...	org.eclipse.zest
Eclipse Modeling Project	Graphical Modeling Frame...	1.4.1.v2010...	org.eclipse.gmf
Eclipse Modeling Project	Graphical Modeling Frame...	1.4.1.v2010...	org.eclipse.gmf.runtime.not...
Eclipse Modeling Project	Graphical Modeling Frame...	1.4.1.v2010...	org.eclipse.gmf.examples.ru...
Eclipse Modeling Project	Graphical Modeling Frame...	1.4.0.v2010...	org.eclipse.gmf.doc
Eclipse Modeling Project	Graphical Modeling Frame...	1.4.1.v2010...	org.eclipse.gmf.runtime.sdk
Eclipse Modeling Project	Graphical Modeling Frame	2.3.0.v2010	org.eclipse.gmf.sdk

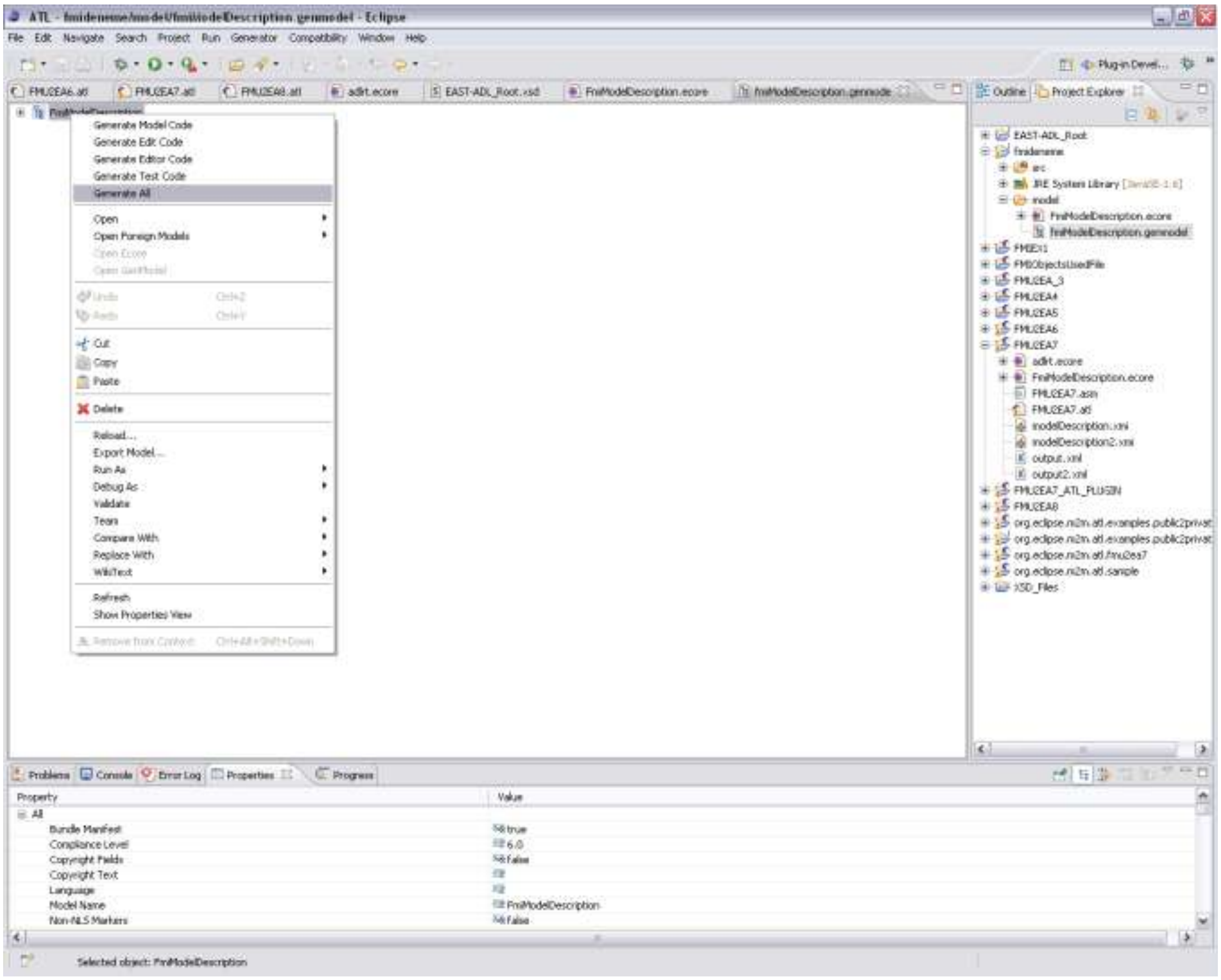
MoDisco Infrastructure (Incubation)
Version: 0.8.1.v201009160758
Build id: @buildid@

Copyright (c) 2008, 2009 Mia-Software.

Visit http://www.eclipse.org/gmt/modisco/

Figure C2.2

While generating the Ecore MetaModel of the FMI XSD, a .genmodel file is created too. This file is the main file which lets the user to obtain the Editor's source files. As seen in Figure, when one gives the generate all command to Eclipse, EMF generates the Editor source files of the corresponding .genmodel. One can run the Editor from these source files. Exactly the same procedure has been applied to EAST-ADL.genmodel for obtaining the EAST-ADL editor. Some modifications have been done to the source files of the EAST-ADL Editor source files in order to obtain the desired Editor.



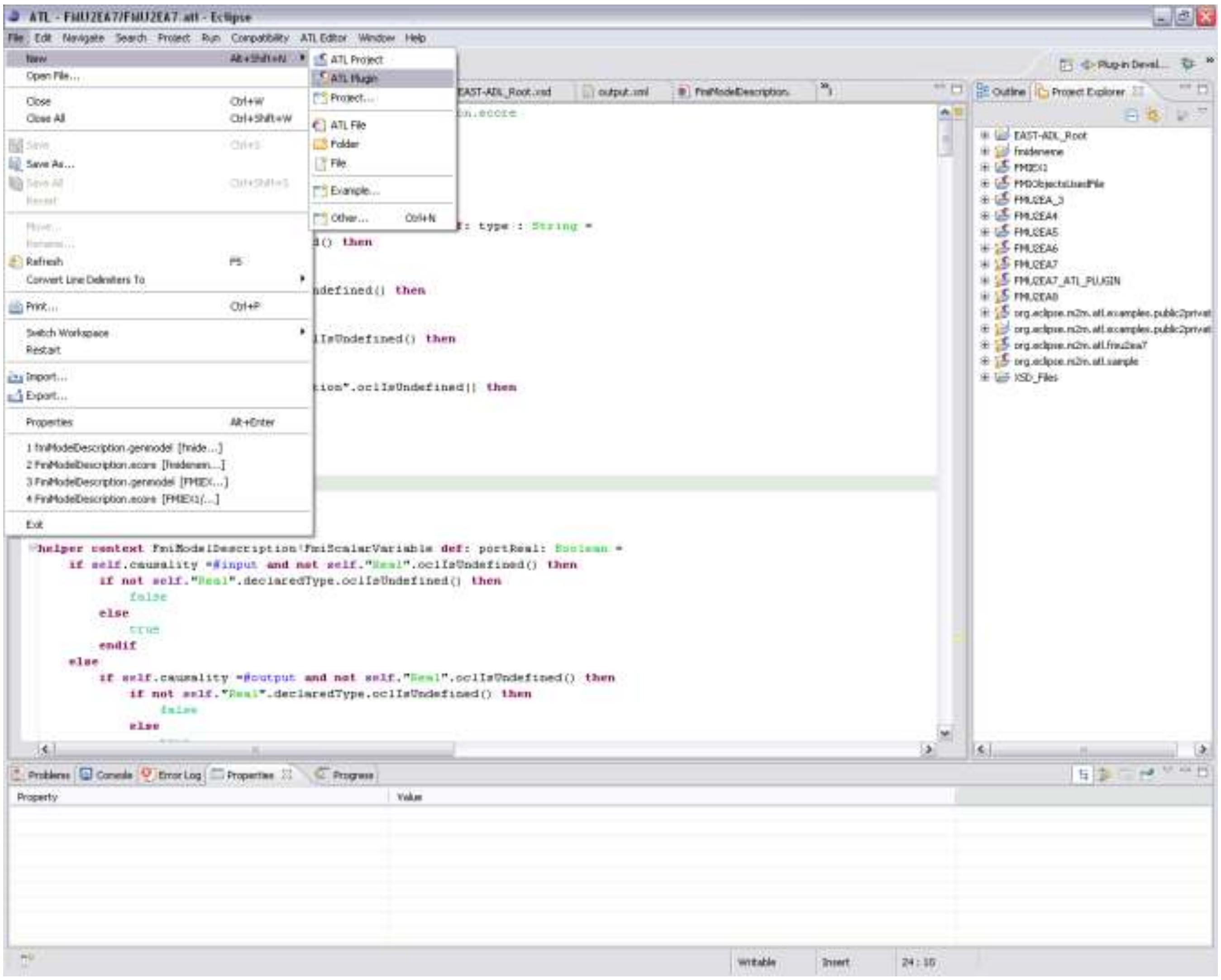
Property	Value
All	
Bundle Manifest	true
Compliance Level	6.0
Copyright Fields	false
Copyright Text	
Language	
Model Name	FmiModelDescription
Non-NLS Markers	false

Selected object: FmiModelDescription

Figure C2.3

3. Obtaining the Plug-in

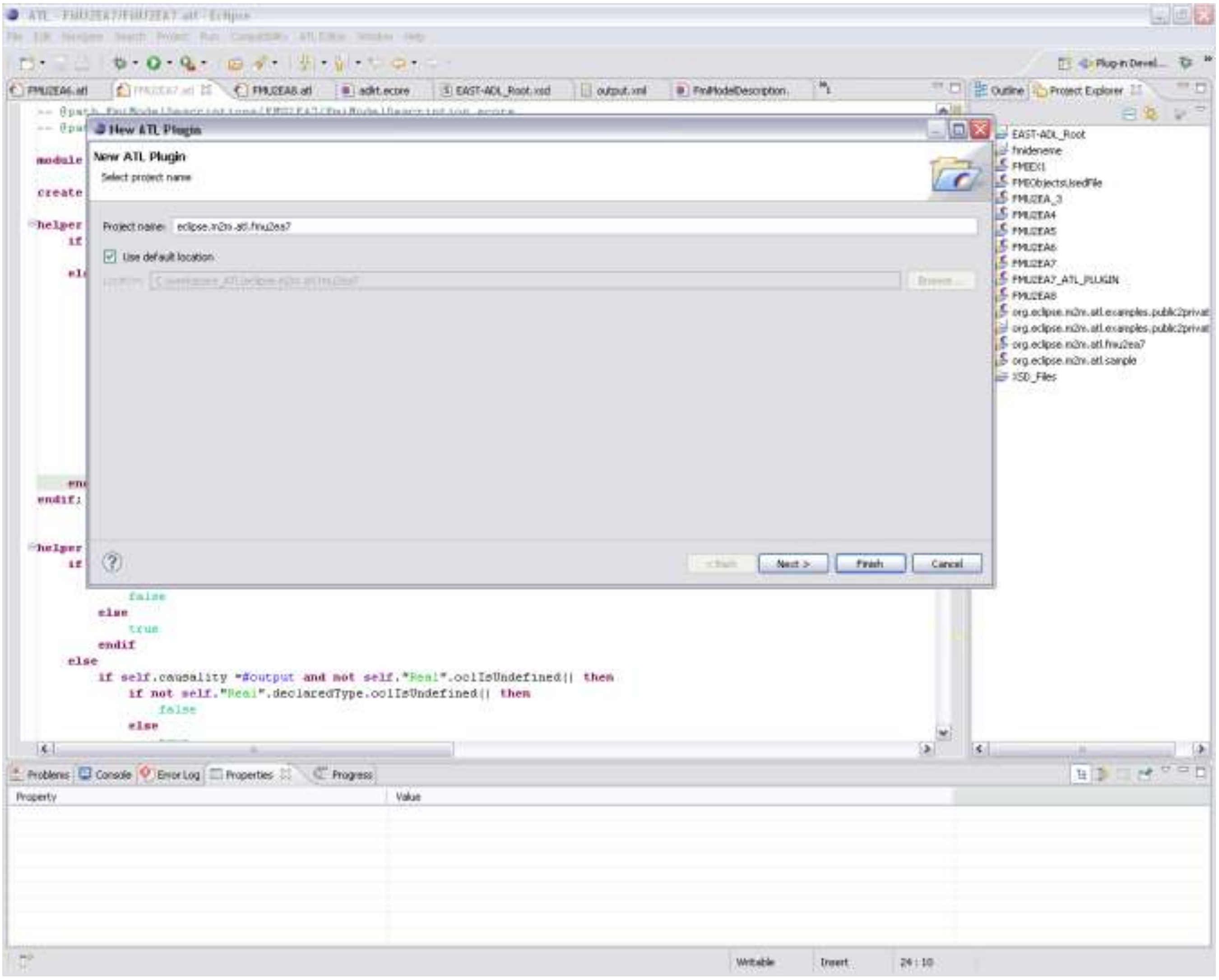
First step to create an ATL Plugin.



```
helper context FmiModelDescription!FmiScalarVariable def: portReal: Boolean =
    if self.causality =#input and not self."Real".oclIsUndefined() then
        if not self."Real".declaredType.oclIsUndefined() then
            false
        else
            true
        endif
    else
        if self.causality =#output and not self."Real".oclIsUndefined() then
            if not self."Real".declaredType.oclIsUndefined() then
                false
            else
```

Figure C3.1

When the new ATL plugin menu opens, one can specify its name as can be seen below.



New ATL Plugin

Select project name

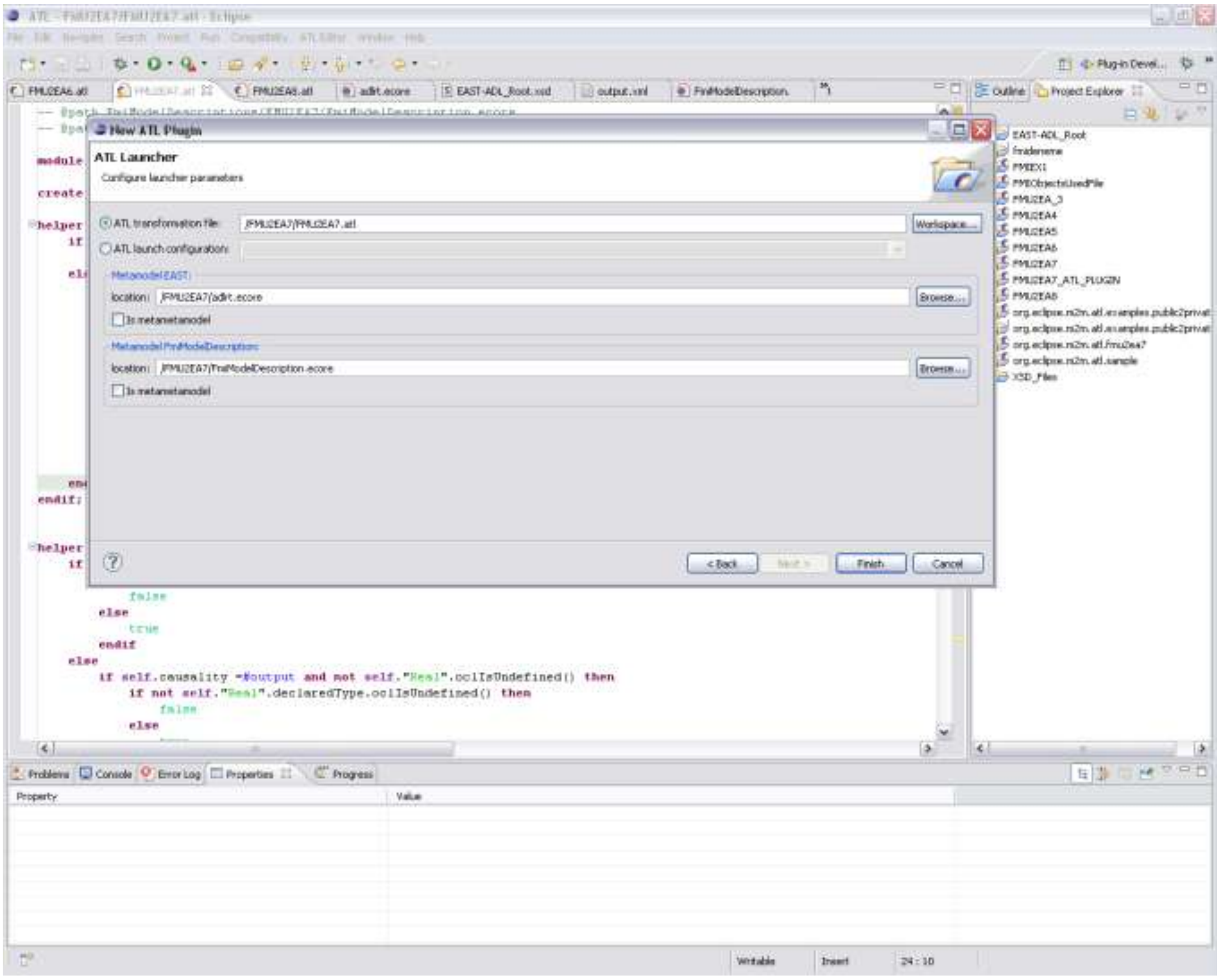
Project name: eclipse.m2m.atl.fmu2ea7

[✓] Use default location

```
            false
        else
            true
        endif
    else
        if self.causality ="#output and not self."Real".oclIsUndefined() then
            if not self."Real".declaredType.oclIsUndefined() then
                false
            else
```

Figure C3.2

Finally, one can specify the ATL file in interest to create a plug-in. When finished, plug-in source files have been generated by EMF. Additional modifications have taken place to these files for obtaining the current plug-in.


New ATL Plugin

ATL Launcher

Configure launcher parameters

(●) ATL transformation file: /FMU2EA7/FMU2EA7.atl

( ) ATL launch configuration:

Metamodel EAST:

location: /FMU2EA7/adlt.ecore

[ ] Is metametamodel

Metamodel FmiModelDescription:

location: /FMU2EA7/FmiModelDescription.ecore

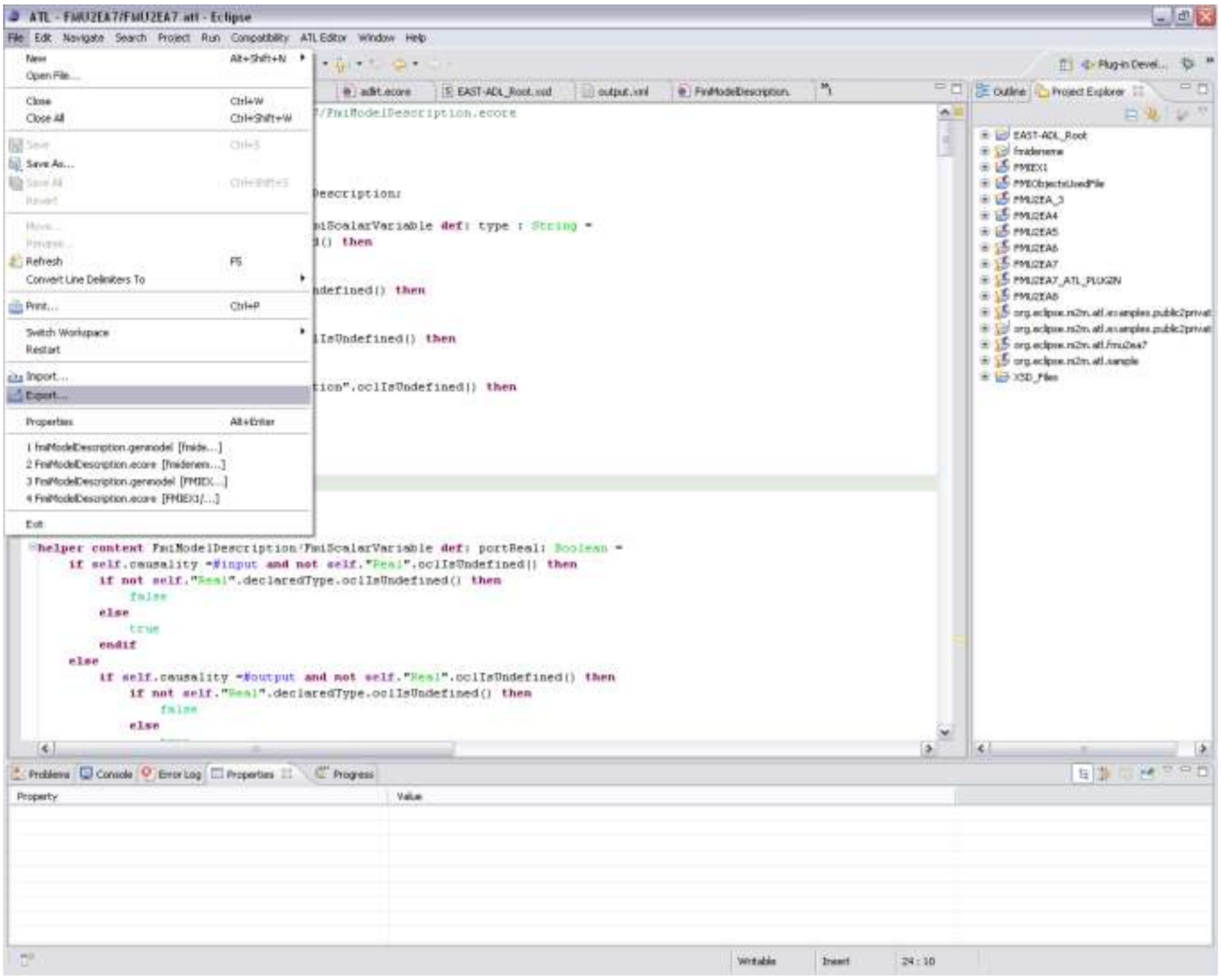
[ ] Is metametamodel

```
        false
    else
        true
    endif
else
    if self.causality =#output and not self."Real".oclIsUndefined() then
        if not self."Real".declaredType.oclIsUndefined() then
            false
        else
```

Figure C3.3

4. Obtaining the Executable

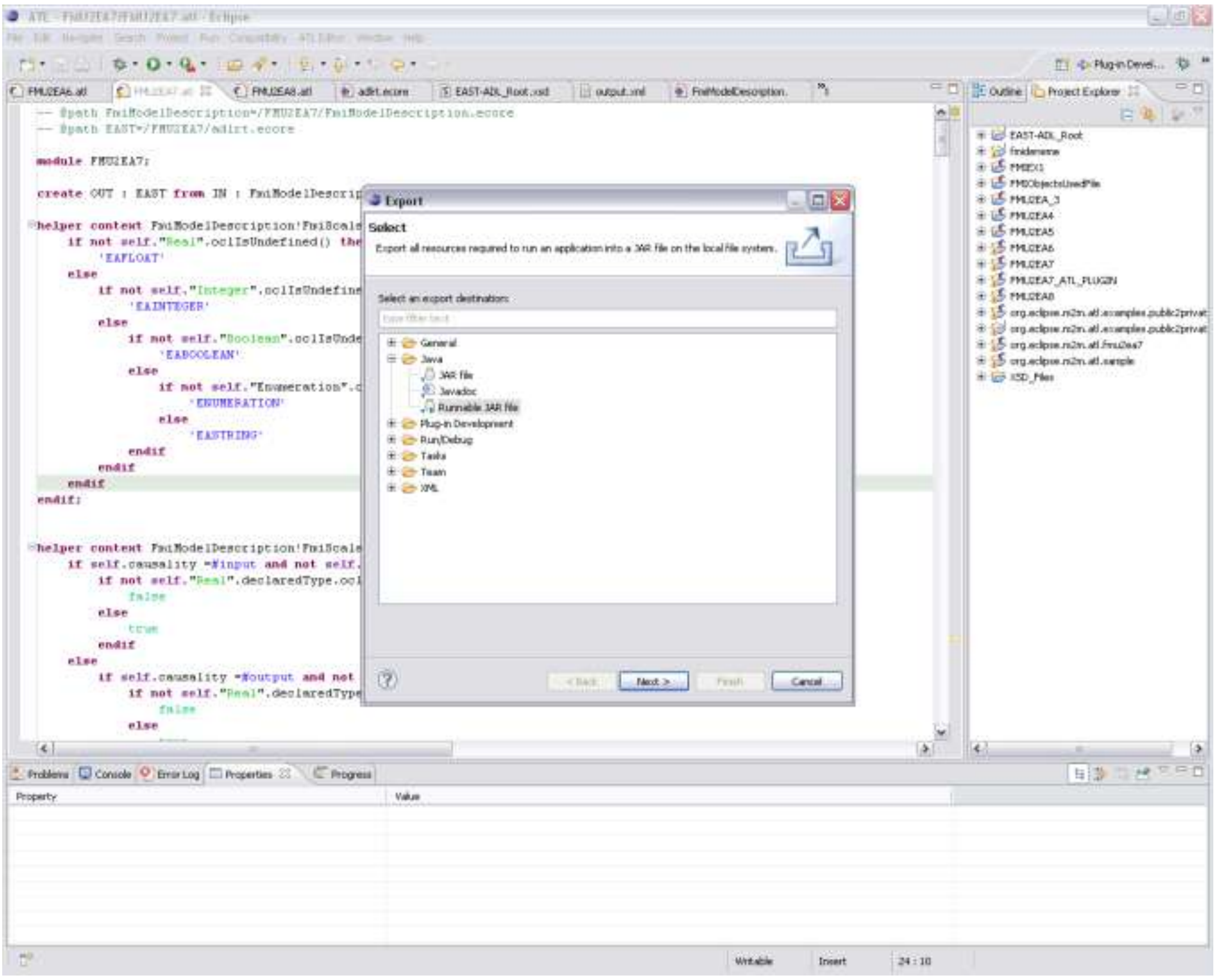
After creating the plug-in one can obtain an executable of the plug-in. The first step to do this is shown in the below figure.



```
/FmiModelDescription.ecore

Description:

iScalarVariable def: type : String =
d() then

ndefined() then

lIsUndefined() then

tion".oclIsUndefined()) then

helper context FmiModelDescription!FmiScalarVariable def: portReal: Boolean =
    if self.causality =#input and not self."Real".oclIsUndefined() then
        if not self."Real".declaredType.oclIsUndefined() then
            false
        else
            true
        endif
    else
        if self.causality =#output and not self."Real".oclIsUndefined() then
            if not self."Real".declaredType.oclIsUndefined() then
                false
            else
```

Figure C4.1

The second step is to select between some choices. For our executable, runnable Jar file is chosen.



```
-- @path FmiModelDescription=/FMU2EA7/FmiModelDescription.ecore
-- @path EAST=/FMU2EA7/adlrt.ecore

module FMU2EA7;

create OUT : EAST from IN : FmiModelDescrip

helper context FmiModelDescription!FmiScala
    if not self."Real".oclIsUndefined() the
        'EAFLOAT'
    else
        if not self."Integer".oclIsUndefine
            'EAINTEGER'
        else
            if not self."Boolean".oclIsUnde
                'EABOOLEAN'
            else
                if not self."Enumeration".c
                    'ENUMERATION'
                else
                    'EASTRING'
            endif
        endif
    endif
endif;


helper context FmiModelDescription!FmiScala
    if self.causality =#input and not self.
        if not self."Real".declaredType.ocl
            false
        else
            true
        endif
    else
        if self.causality =#output and not
            if not self."Real".declaredType
                false
            else
```

Export

Select

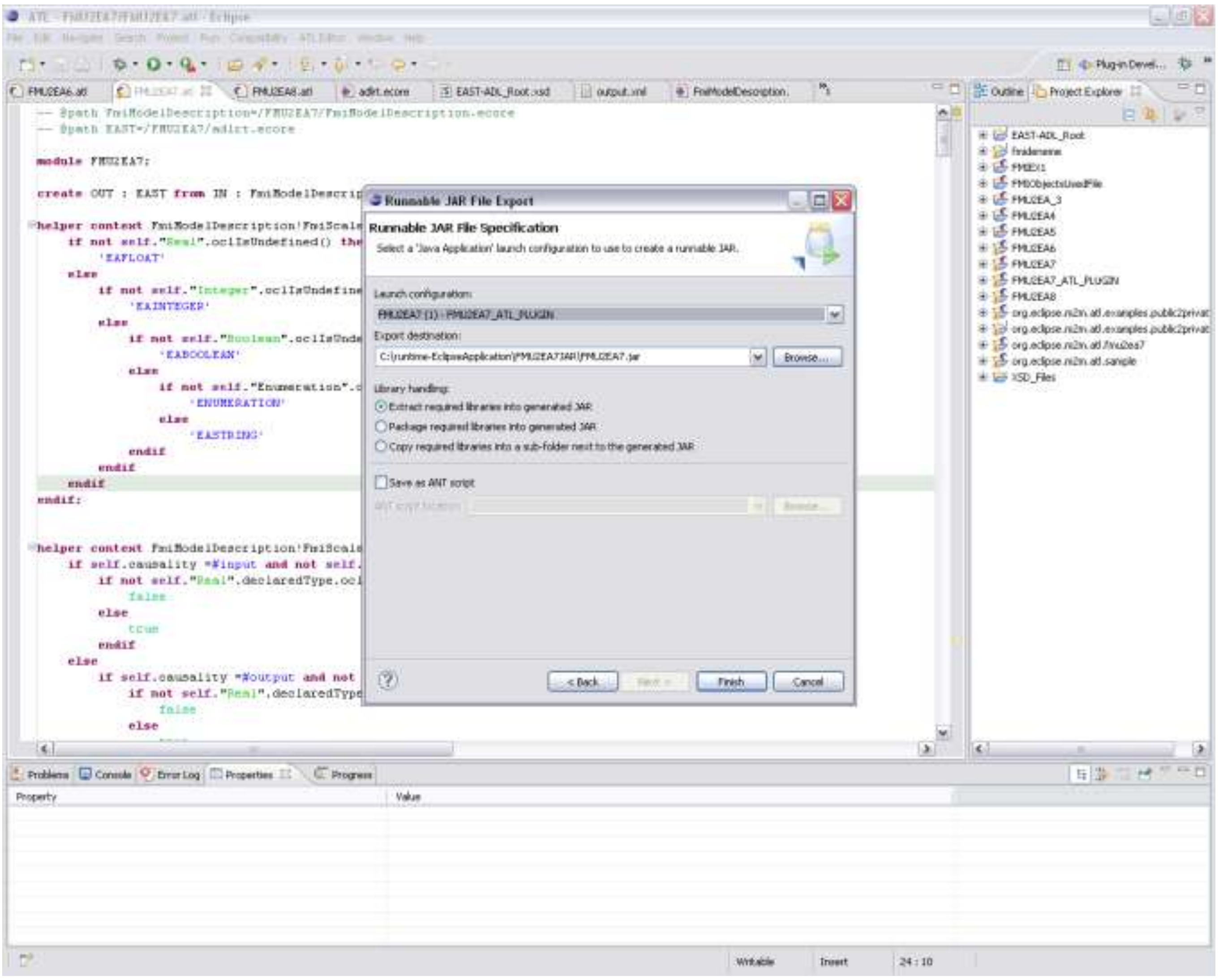
Export all resources required to run an application into a JAR file on the local file system.

Select an export destination:

- General
- Java
  - JAR file
  - Javadoc
  - Runnable JAR file
- Plug-in Development
- Run/Debug
- Tasks
- Team
- XML

Figure C4.2

Final step in obtaining the executable is shown as below. First, the plug-in obtained for the ATL transformation is chosen. Then the desired location of where to put the generated executable is specified. Then extracting the required libraries into the Jar file is chosen.



```
-- @path FmiModelDescription=/FMU2EA7/FmiModelDescription.ecore
-- @path EAST=/FMU2EA7/adlrt.ecore

module FMU2EA7;

create OUT : EAST from IN : FmiModelDescrip

helper context FmiModelDescription!FmiScala
    if not self."Real".oclIsUndefined() the
        'EAFLOAT'
    else
        if not self."Integer".oclIsUndefine
            'EAINTEGER'
        else
            if not self."Boolean".oclIsUnde
                'EABOOLEAN'
            else
                if not self."Enumeration".c
                    'ENUMERATION'
                else
                    'EASTRING'
            endif
        endif
    endif
endif;


helper context FmiModelDescription!FmiScala
    if self.causality =#input and not self.
        if not self."Real".declaredType.ocl
            false
        else
            true
        endif
    else
        if self.causality =#output and not
            if not self."Real".declaredType
                false
            else
```

Runnable JAR File Export

Runnable JAR File Specification

Select a 'Java Application' launch configuration to use to create a runnable JAR.

Launch configuration: FMU2EA7 (1) - FMU2EA7_ATL_PLUGIN

Export destination: C:\runtime-EclipseApplication\FMU2EA7JAR\FMU2EA7.jar

Library handling:
- [✓ Extract required libraries into generated JAR]
- Package required libraries into generated JAR
- Copy required libraries into a sub-folder next to the generated JAR

Save as ANT script

Figure C4.3

D. Appendix D

1. Analysis of the current problems of the KTH tool

FROM MATLAB 2 ECLIPSE

Case 1/ If we have a model(a .mdl file) that constitutes of only subsystems and "no connections" we can save them as a .simulink file at Matlab and then we can open this .simulink file in eclipse with no problem and we can translate it into a .uml file also. The conversion of the .mdl file to .uml file in this case looks right.

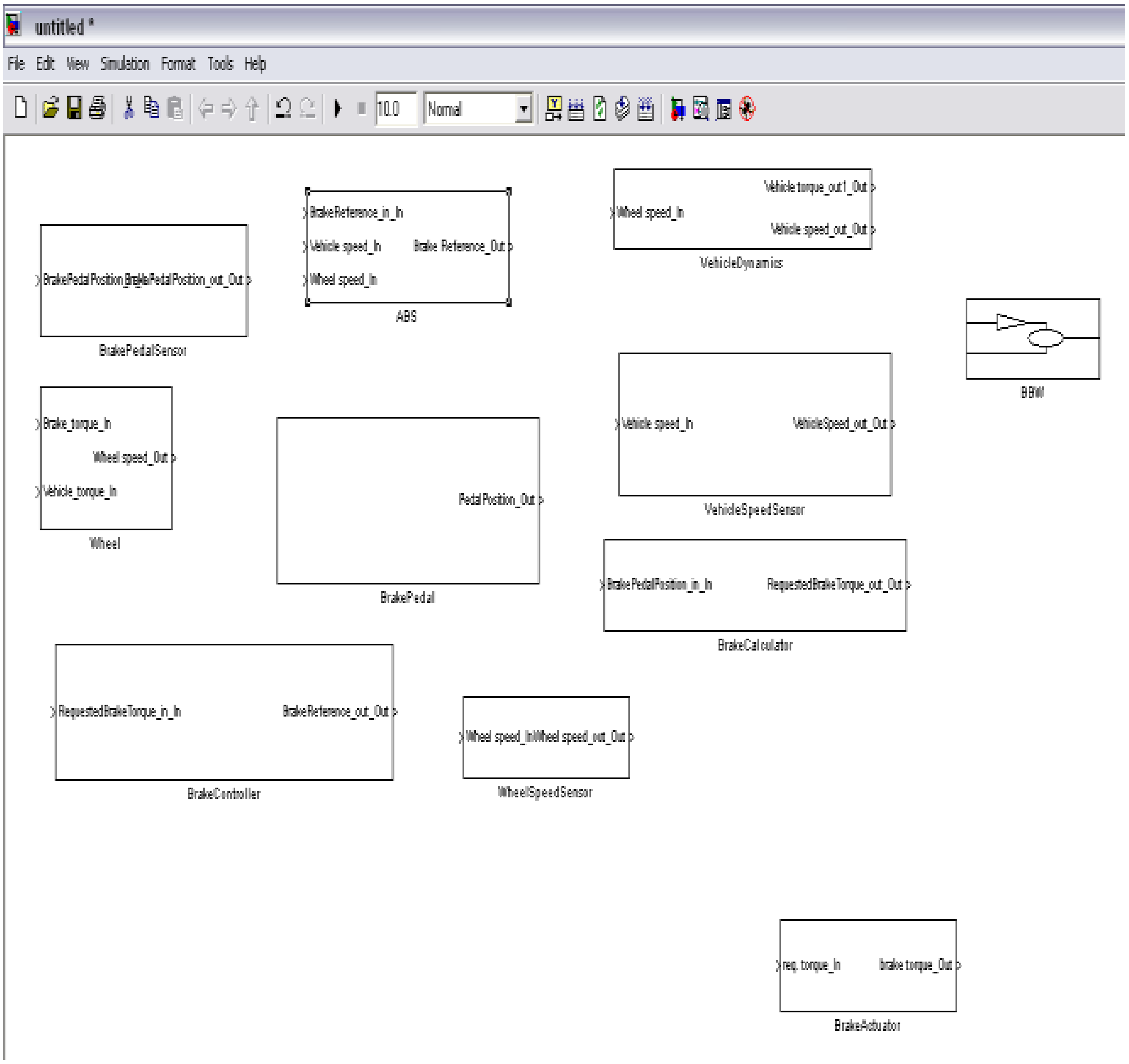


Figure D1.1/ View of the .mdl file created in Simulink. Notice that there are several subsystems in the system and none of the subsystems in the system are connected to each other.

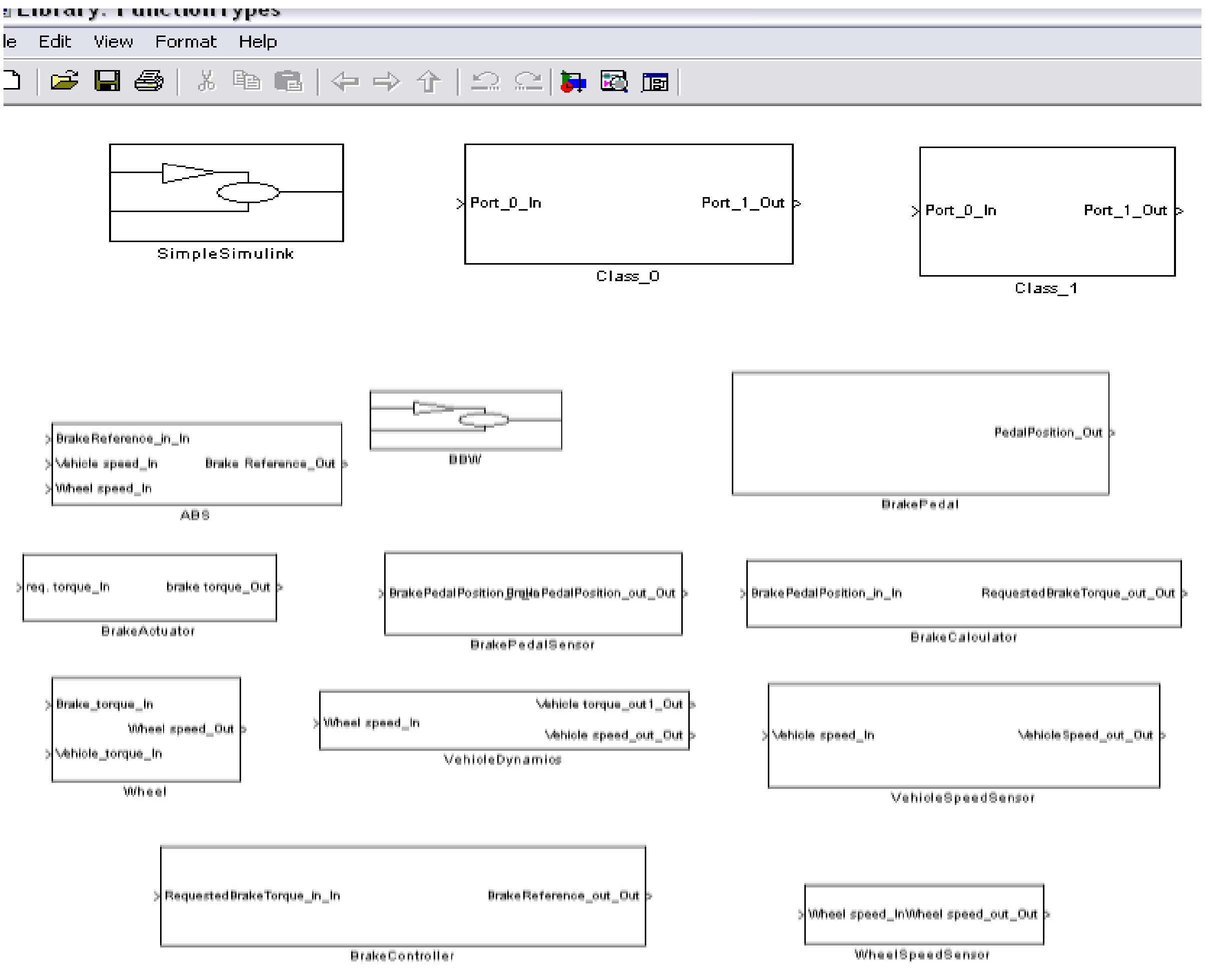
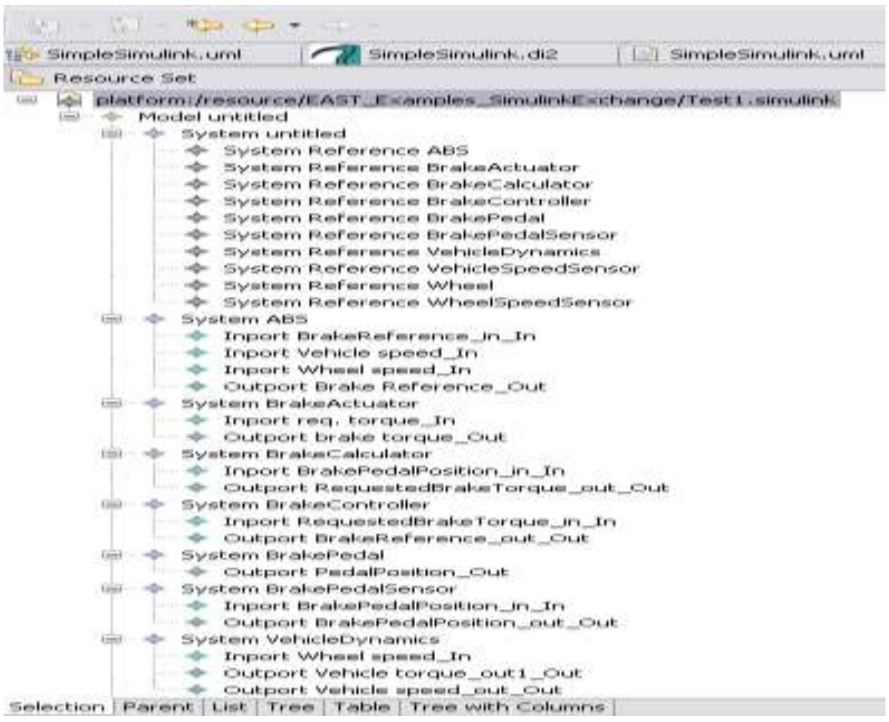
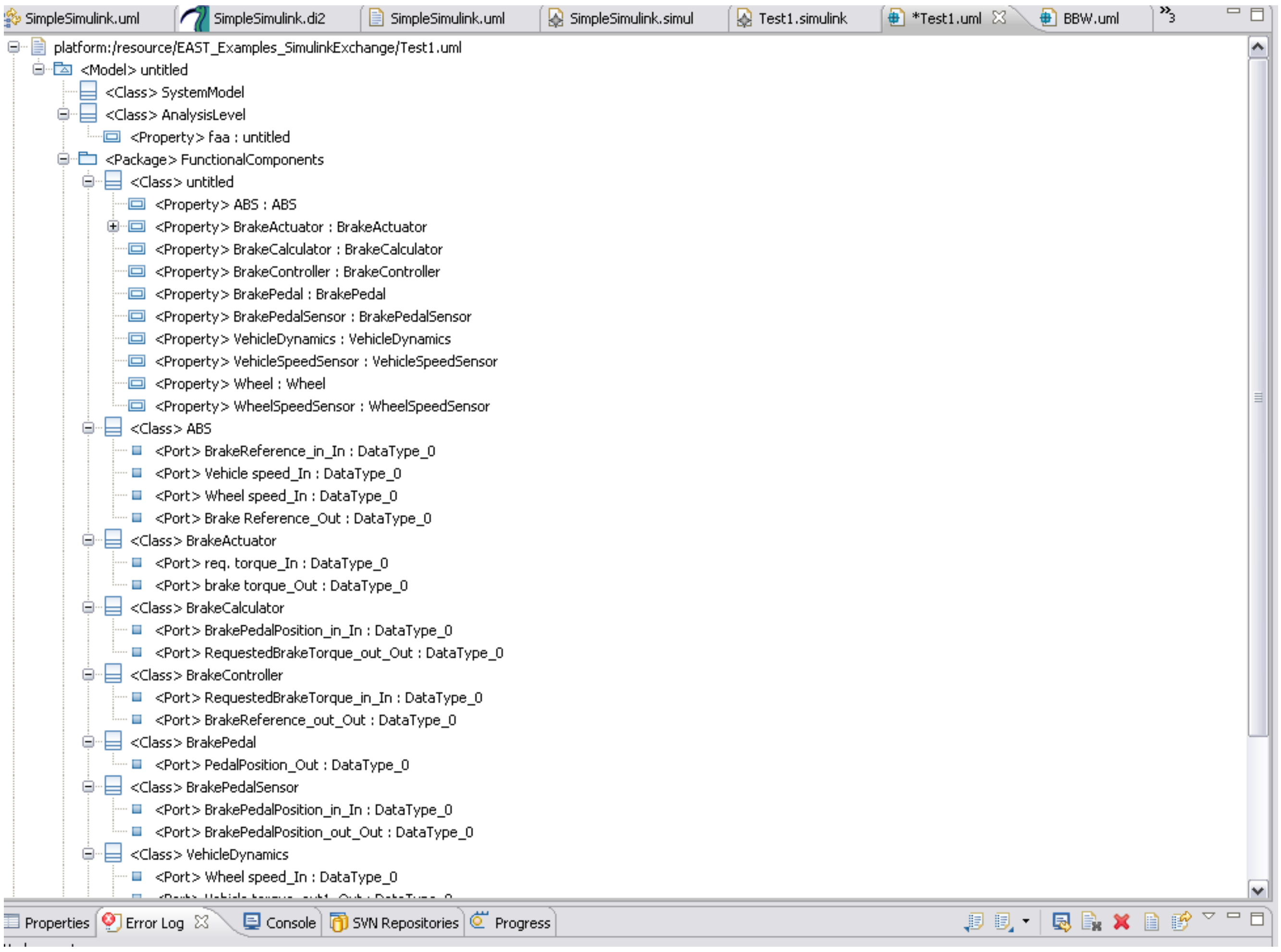


Figure D1.2/ View of the FunctionTypes.mdl file which all the subsystem blocks have been added.


SimpleSimulink.uml
SimpleSimulink.di2
SimpleSimulink.uml
Selection | Parent | List | Tree | Table | Tree with Columns

```
Resource Set
  platform:/resource/EAST_Examples_SimulinkExchange/Test1.simulink
    Model untitled
      System untitled
        System Reference ABS
        System Reference BrakeActuator
        System Reference BrakeCalculator
        System Reference BrakeController
        System Reference BrakePedal
        System Reference BrakePedalSensor
        System Reference VehicleDynamics
        System Reference VehicleSpeedSensor
        System Reference Wheel
        System Reference WheelSpeedSensor
      System ABS
        Inport BrakeReference_in_In
        Inport Vehicle speed_In
        Inport Wheel speed_In
        Outport Brake Reference_Out
      System BrakeActuator
        Inport req. torque_In
        Outport brake torque_Out
      System BrakeCalculator
        Inport BrakePedalPosition_in_In
        Outport RequestedBrakeTorque_out_Out
      System BrakeController
        Inport RequestedBrakeTorque_in_In
        Outport BrakeReference_out_Out
      System BrakePedal
        Outport PedalPosition_Out
      System BrakePedalSensor
        Inport BrakePedalPosition_in_In
        Outport BrakePedalPosition_out_Out
      System VehicleDynamics
        Inport Wheel speed_In
        Outport Vehicle torque_out1_Out
        Outport Vehicle speed_out_Out
```

Figure D1.3/ View of the .simulink file(which is created by Matlab) that is opened in Eclipse. Everything is in order.



```
platform:/resource/EAST_Examples_SimulinkExchange/Test1.uml
  <Model> untitled
    <Class> SystemModel
    <Class> AnalysisLevel
      <Property> faa : untitled
    <Package> FunctionalComponents
      <Class> untitled
        <Property> ABS : ABS
        <Property> BrakeActuator : BrakeActuator
        <Property> BrakeCalculator : BrakeCalculator
        <Property> BrakeController : BrakeController
        <Property> BrakePedal : BrakePedal
        <Property> BrakePedalSensor : BrakePedalSensor
        <Property> VehicleDynamics : VehicleDynamics
        <Property> VehicleSpeedSensor : VehicleSpeedSensor
        <Property> Wheel : Wheel
        <Property> WheelSpeedSensor : WheelSpeedSensor
      <Class> ABS
        <Port> BrakeReference_in_In : DataType_0
        <Port> Vehicle speed_In : DataType_0
        <Port> Wheel speed_In : DataType_0
        <Port> Brake Reference_Out : DataType_0
      <Class> BrakeActuator
        <Port> req. torque_In : DataType_0
        <Port> brake torque_Out : DataType_0
      <Class> BrakeCalculator
        <Port> BrakePedalPosition_in_In : DataType_0
        <Port> RequestedBrakeTorque_out_Out : DataType_0
      <Class> BrakeController
        <Port> RequestedBrakeTorque_in_In : DataType_0
        <Port> BrakeReference_out_Out : DataType_0
      <Class> BrakePedal
        <Port> PedalPosition_Out : DataType_0
      <Class> BrakePedalSensor
        <Port> BrakePedalPosition_in_In : DataType_0
        <Port> BrakePedalPosition_out_Out : DataType_0
      <Class> VehicleDynamics
        <Port> Wheel speed_In : DataType_0
```

Figure D1.4/ View of the .uml file that is obtained by converting the .simulink file in hand. Everything is OK which means(and also will mean the same in the below figures): The structure of the system and the building blocks looks alright and complete. We have the right classifications of types and prototypes and we have all the types and prototypes that makes up the system(If there are also ports and connections in the system, all of them is expected to be in hand and correctly classified for the system to be called OK).

Case 2/ If we have a model(a .mdl file) that constitutes of "subsystems that have connections" we can save them as a .simulink file at Matlab but we can not open this .simulink file in eclipse directly. The compiler complains about a variable "connections" that is in the current .simulink file created by Matlab. When we have a look of the simulink file with the help of text editor we see there is a definiton of every connection with that variable by giving a value to that. And when we erase these variables with their values, the compiler stops giving warnings.Now we can translate the .simulink file into a .uml file. And the conversion of the .mdl file to .uml file in this case looks right. We can see also all the connections which seems right.(The modified .simulink file(it was obtained by erasing the "connections" variables in the original .simulink file that is created by Matlab) of course does not work in Matlab now. Matlab does not recognize this file and gives errors if you try to open this modified .simulink file as we erased some of the variables in the original .simulink file that Matlab created in first hand.)

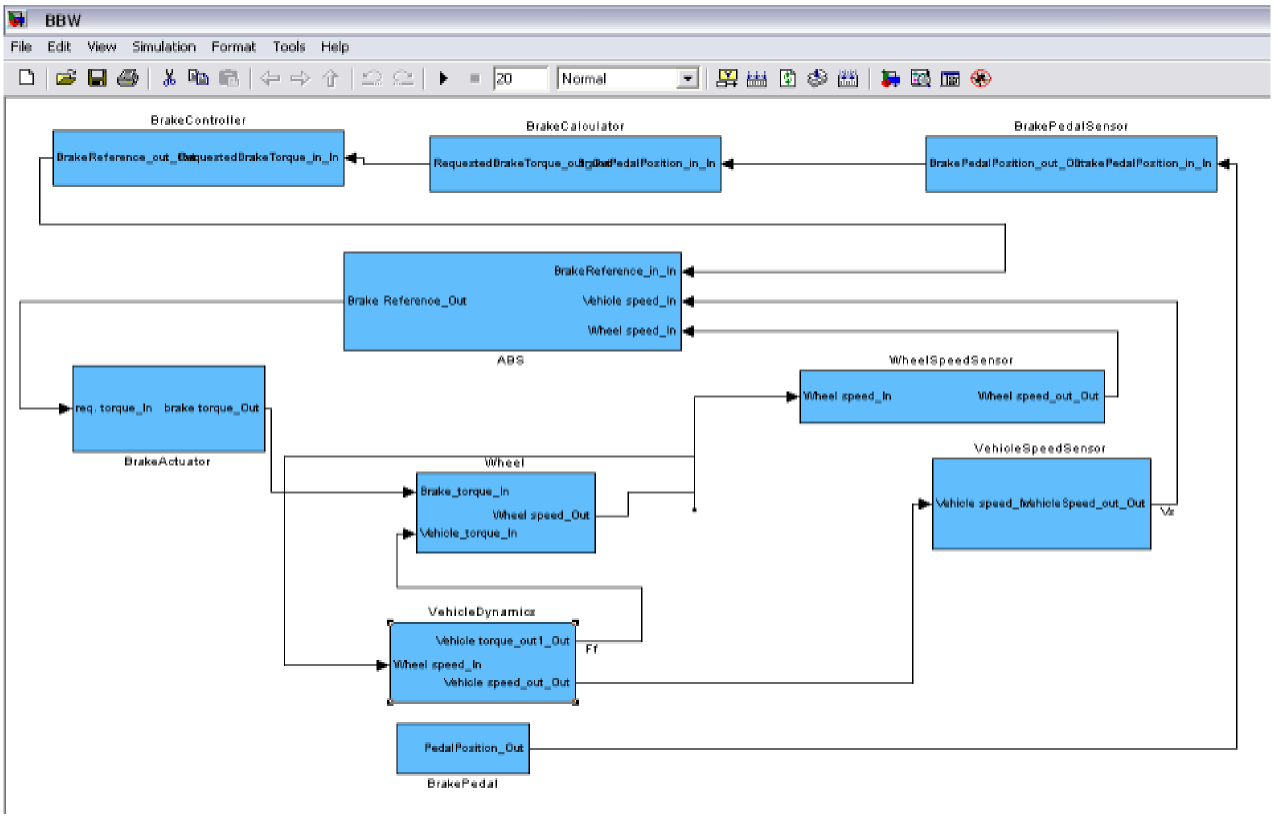


Figure D1.5/ A system in Simulink/Matlab. Note that the system has connections between the blocks.

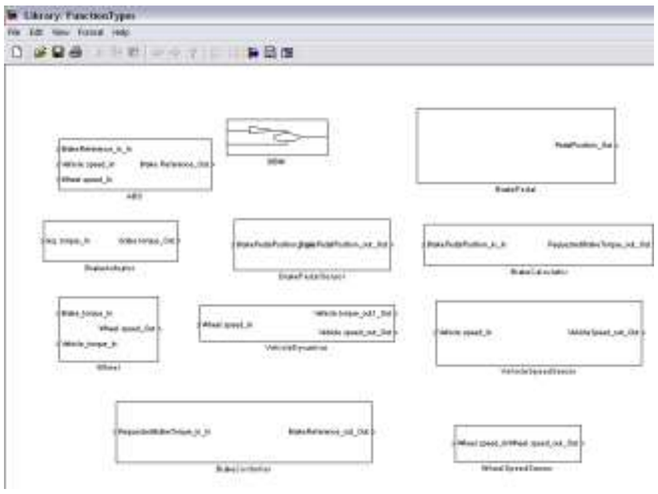
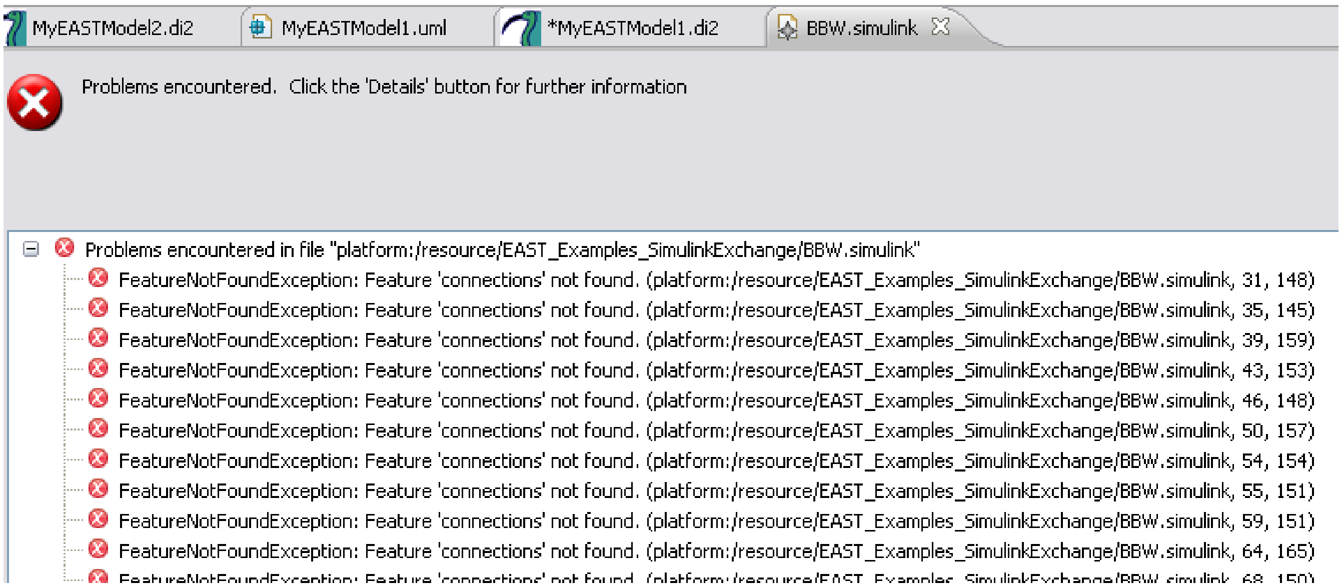


Figure D1.6/ Function Types of the system

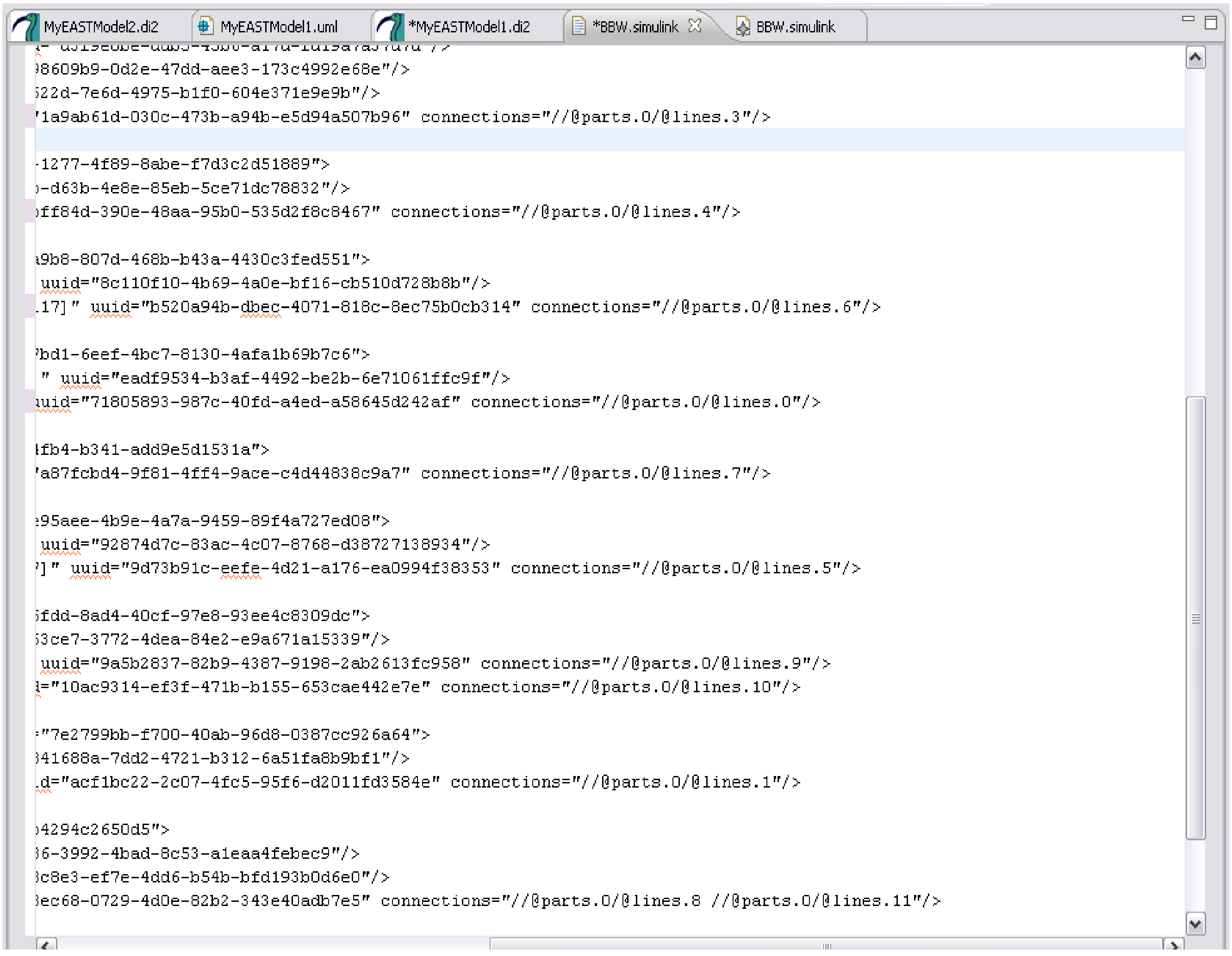


MyEASTModel2.di2 | MyEASTModel1.uml | *MyEASTModel1.di2 | BBW.simulink

Problems encountered. Click the 'Details' button for further information

```
Problems encountered in file "platform:/resource/EAST_Examples_SimulinkExchange/BBW.simulink"
    FeatureNotFoundException: Feature 'connections' not found. (platform:/resource/EAST_Examples_SimulinkExchange/BBW.simulink, 31, 148)
    FeatureNotFoundException: Feature 'connections' not found. (platform:/resource/EAST_Examples_SimulinkExchange/BBW.simulink, 35, 145)
    FeatureNotFoundException: Feature 'connections' not found. (platform:/resource/EAST_Examples_SimulinkExchange/BBW.simulink, 39, 159)
    FeatureNotFoundException: Feature 'connections' not found. (platform:/resource/EAST_Examples_SimulinkExchange/BBW.simulink, 43, 153)
    FeatureNotFoundException: Feature 'connections' not found. (platform:/resource/EAST_Examples_SimulinkExchange/BBW.simulink, 46, 148)
    FeatureNotFoundException: Feature 'connections' not found. (platform:/resource/EAST_Examples_SimulinkExchange/BBW.simulink, 50, 157)
    FeatureNotFoundException: Feature 'connections' not found. (platform:/resource/EAST_Examples_SimulinkExchange/BBW.simulink, 54, 154)
    FeatureNotFoundException: Feature 'connections' not found. (platform:/resource/EAST_Examples_SimulinkExchange/BBW.simulink, 55, 151)
    FeatureNotFoundException: Feature 'connections' not found. (platform:/resource/EAST_Examples_SimulinkExchange/BBW.simulink, 59, 151)
    FeatureNotFoundException: Feature 'connections' not found. (platform:/resource/EAST_Examples_SimulinkExchange/BBW.simulink, 64, 165)
    FeatureNotFoundException: Feature 'connections' not found. (platform:/resource/EAST_Examples_SimulinkExchange/BBW.simulink, 68, 150)
```

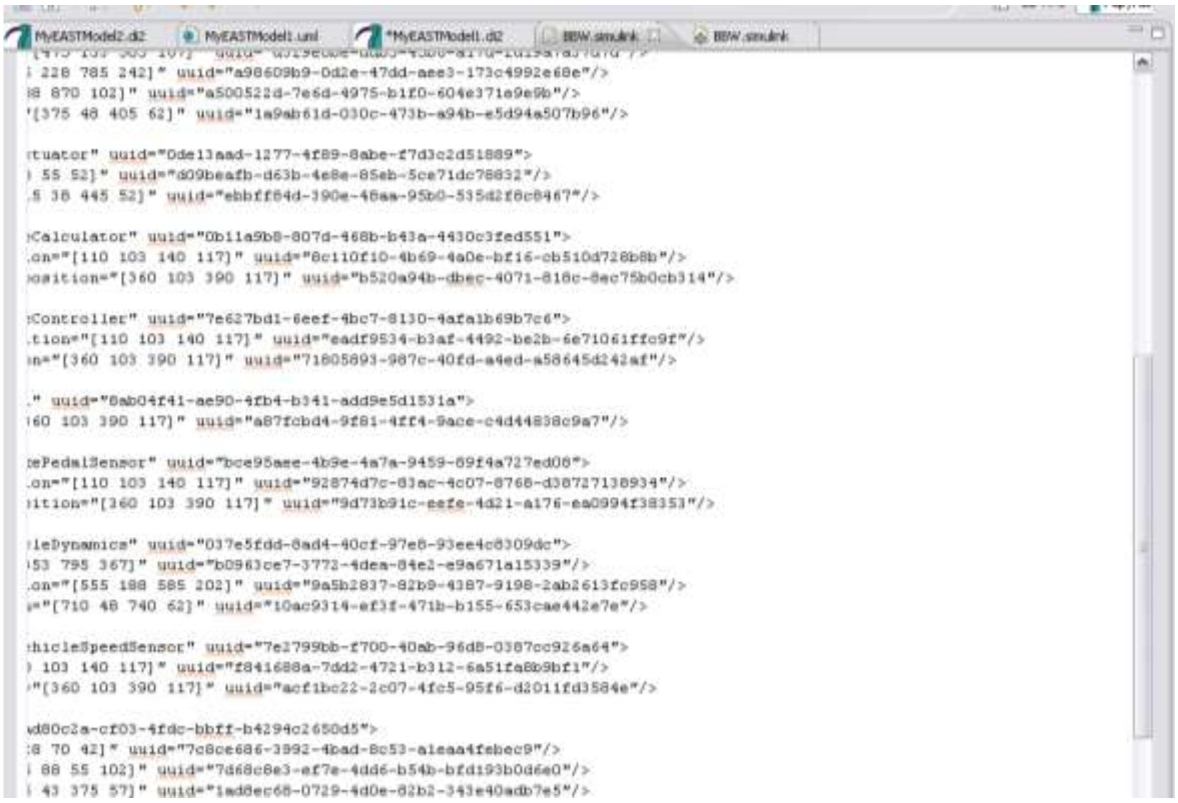
Figure D1.7/ Errors that are taken in Eclipse side when tried to open the .simulink file created in Matlab



MyEASTModel2.di2 | MyEASTModel1.uml | *MyEASTModel1.di2 | *BBW.simulink | BBW.simulink

```
98609b9-0d2e-47dd-aee3-173c4992e68e"/>
522d-7e6d-4975-b1f0-604e371e9e9b"/>
'1a9ab61d-030c-473b-a94b-e5d94a507b96" connections="//@parts.0/@lines.3"/>

-1277-4f89-8abe-f7d3c2d51889">
)-d63b-4e8e-85eb-5ce71dc78832"/>
)ff84d-390e-48aa-95b0-535d2f8c8467" connections="//@parts.0/@lines.4"/>

a9b8-807d-468b-b43a-4430c3fed551">
 uuid="8c110f10-4b69-4a0e-bf16-cb510d728b8b"/>
.17]" uuid="b520a94b-dbec-4071-818c-8ec75b0cb314" connections="//@parts.0/@lines.6"/>

'bd1-6eef-4bc7-8130-4afa1b69b7c6">
 " uuid="eadf9534-b3af-4492-be2b-6e71061ffc9f"/>
uid="71805893-987c-40fd-a4ed-a58645d242af" connections="//@parts.0/@lines.0"/>

4fb4-b341-add9e5d1531a">
'a87fcbd4-9f81-4ff4-9ace-c4d44838c9a7" connections="//@parts.0/@lines.7"/>

e95aee-4b9e-4a7a-9459-89f4a727ed08">
 uuid="92874d7c-83ac-4c07-8768-d38727138934"/>
?]" uuid="9d73b91c-eefe-4d21-a176-ea0994f38353" connections="//@parts.0/@lines.5"/>

5fdd-8ad4-40cf-97e8-93ee4c8309dc">
53ce7-3772-4dea-84e2-e9a671a15339"/>
 uuid="9a5b2837-82b9-4387-9198-2ab2613fc958" connections="//@parts.0/@lines.9"/>
d="10ac9314-ef3f-471b-b155-653cae442e7e" connections="//@parts.0/@lines.10"/>

="7e2799bb-f700-40ab-96d8-0387cc926a64">
341688a-7dd2-4721-b312-6a51fa8b9bf1"/>
.d="acf1bc22-2c07-4fc5-95f6-d2011fd3584e" connections="//@parts.0/@lines.1"/>

)4294c2650d5">
36-3992-4bad-8c53-a1eaa4febec9"/>
3c8e3-ef7e-4dd6-b54b-bfd193b0d6e0"/>
3ec68-0729-4d0e-82b2-343e40adb7e5" connections="//@parts.0/@lines.8 //@parts.0/@lines.11"/>
```

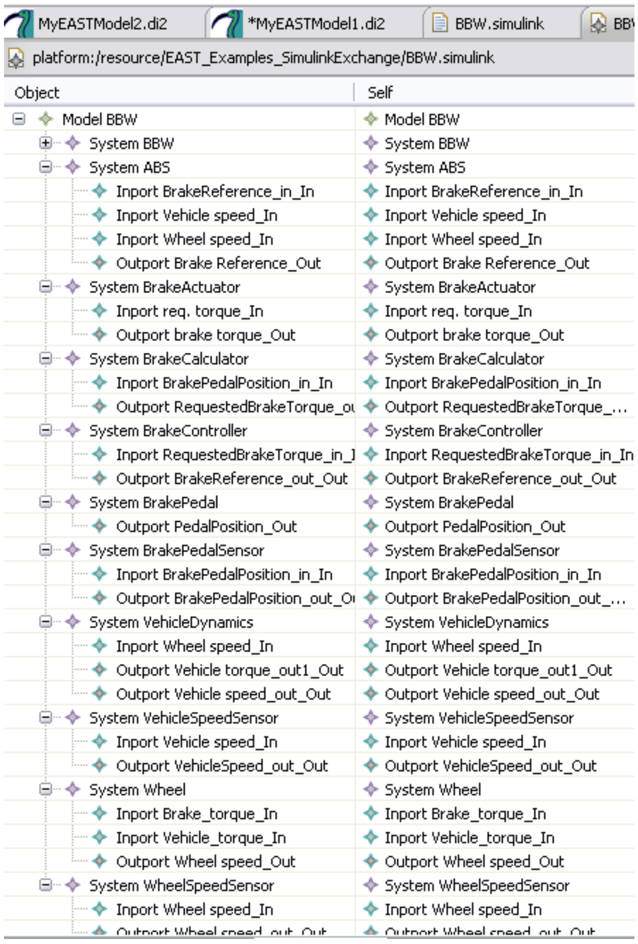
Figure D1.8/ The ”connections” variables in the .simulink file created by Matlab that is complained by the Eclipse compiler can be seen in the most right



MyEASTModel2.di2 | MyEASTModel1.uml | *MyEASTModel1.di2 | BBW.simulink | BBW.simulink

```
 228 785 242]" uuid="a98609b9-0d2e-47dd-aee3-173c4992e68e"/>
8 870 102]" uuid="a500522d-7e6d-4975-b1f0-604e371a9e9b"/>
'[375 48 405 62]" uuid="1a9ab61d-030c-473b-a94b-e5d94a507b96"/>

tuator" uuid="0de13aad-1277-4f89-8abe-f7d3c2d51889">
 55 52]" uuid="d09beafb-d63b-4e8e-85eb-5ce71dc78832"/>
.5 38 445 52]" uuid="ebbff84d-390e-48aa-95b0-535d2f8c8467"/>

Calculator" uuid="0b11a9b8-807d-468b-b43a-4430c3fed551">
.on="[110 103 140 117]" uuid="8c110f10-4b69-4a0e-bf16-cb510d728b8b"/>
osition="[360 103 390 117]" uuid="b520a94b-dbec-4071-818c-8ec75b0cb314"/>

Controller" uuid="7e627bd1-6eef-4bc7-8130-4afa1b69b7c6">
.tion="[110 103 140 117]" uuid="eadf9534-b3af-4492-be2b-6e71061ffc9f"/>
n="[360 103 390 117]" uuid="71805893-987c-40fd-a4ed-a58645d242af"/>

." uuid="8ab04f41-ae90-4fb4-b341-add9e5d1531a">
60 103 390 117]" uuid="a87fcbd4-9f81-4ff4-9ace-c4d44838c9a7"/>

ePedalSensor" uuid="bce95aee-4b9e-4a7a-9459-89f4a727ed08">
.on="[110 103 140 117]" uuid="92874d7c-83ac-4c07-8768-d38727138934"/>
ition="[360 103 390 117]" uuid="9d73b91c-eefe-4d21-a176-ea0994f38353"/>

leDynamics" uuid="037e5fdd-8ad4-40cf-97e8-93ee4c8309dc">
53 795 367]" uuid="b0963ce7-3772-4dea-84e2-e9a671a15339"/>
.on="[555 188 585 202]" uuid="9a5b2837-82b9-4387-9198-2ab2613fc958"/>
="[710 48 740 62]" uuid="10ac9314-ef3f-471b-b155-653cae442e7e"/>

hicleSpeedSensor" uuid="7e2799bb-f700-40ab-96d8-0387cc926a64">
 103 140 117]" uuid="f841688a-7dd2-4721-b312-6a51fa8b9bf1"/>
"[360 103 390 117]" uuid="acf1bc22-2c07-4fc5-95f6-d2011fd3584e"/>

d80c2a-cf03-4fdc-bbff-b4294c2650d5">
8 70 42]" uuid="7c8ce686-3992-4bad-8c53-a1eaa4febec9"/>
 88 55 102]" uuid="7d68c8e3-ef7e-4dd6-b54b-bfd193b0d6e0"/>
 43 375 57]" uuid="1ad8ec68-0729-4d0e-82b2-343e40adb7e5"/>
```

Figure D1.9/ View of the modified .simulink file after the ”connections” variables are cleared

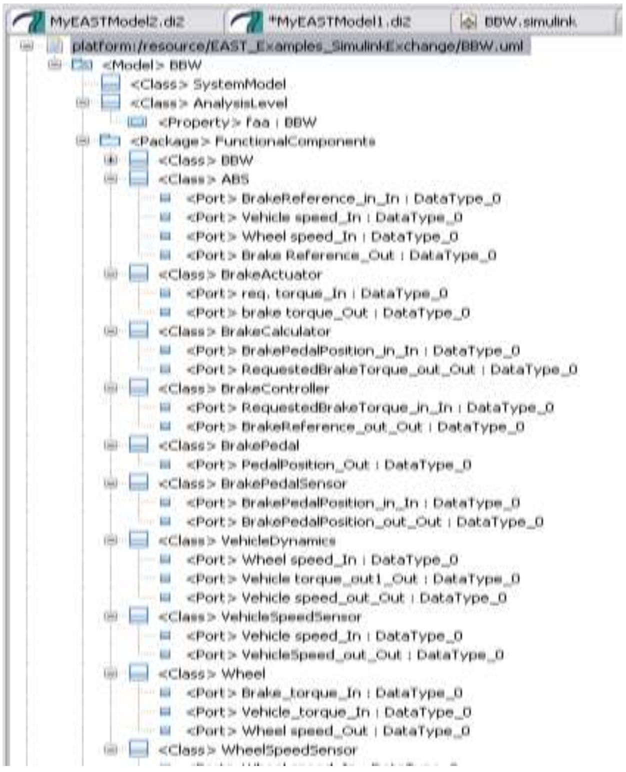


MyEASTModel2.di2 | *MyEASTModel1.di2 | BBW.simulink | BB

platform:/resource/EAST_Examples_SimulinkExchange/BBW.simulink

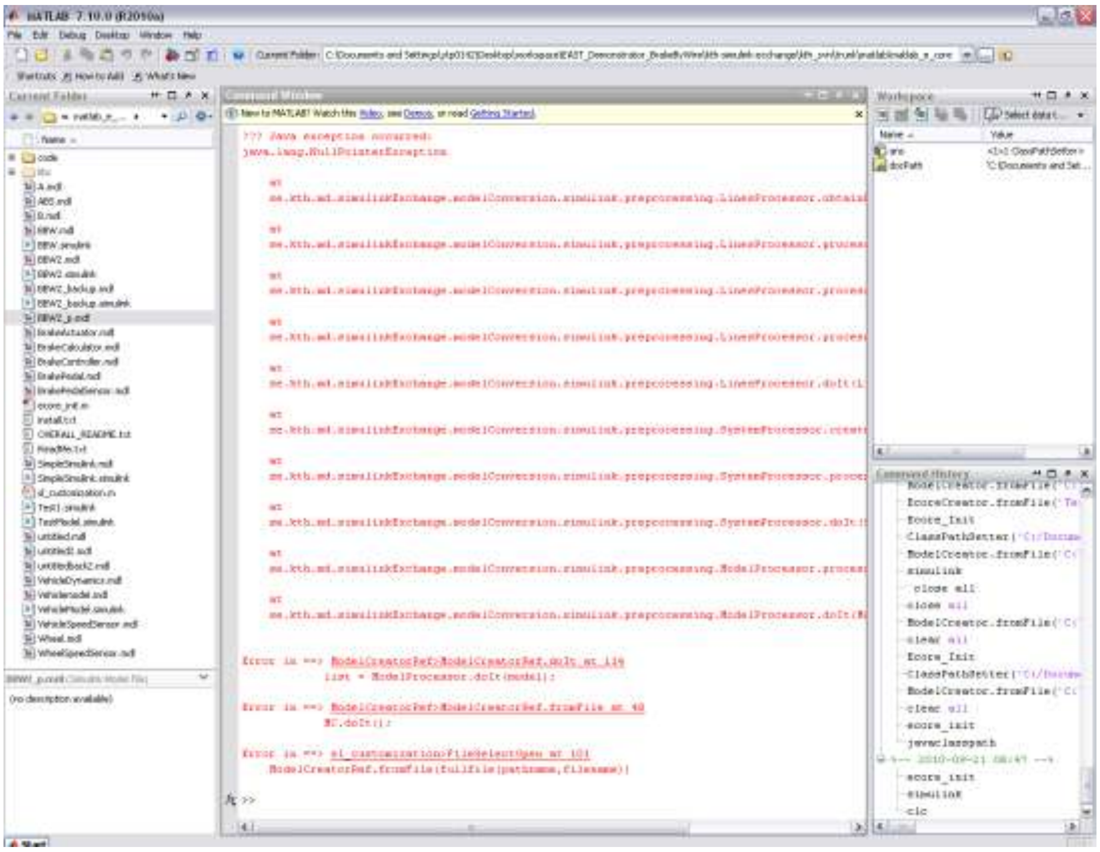
Object	Self
Model BBW	Model BBW
System BBW	System BBW
System ABS	System ABS
Inport BrakeReference_in_In	Inport BrakeReference_in_In
Inport Vehicle speed_In	Inport Vehicle speed_In
Inport Wheel speed_In	Inport Wheel speed_In
Outport Brake Reference_Out	Outport Brake Reference_Out
System BrakeActuator	System BrakeActuator
Inport req. torque_In	Inport req. torque_In
Outport brake torque_Out	Outport brake torque_Out
System BrakeCalculator	System BrakeCalculator
Inport BrakePedalPosition_in_In	Inport BrakePedalPosition_in_In
Outport RequestedBrakeTorque_ou	Outport RequestedBrakeTorque_…
System BrakeController	System BrakeController
Inport RequestedBrakeTorque_in_I	Inport RequestedBrakeTorque_in_In
Outport BrakeReference_out_Out	Outport BrakeReference_out_Out
System BrakePedal	System BrakePedal
Outport PedalPosition_Out	Outport PedalPosition_Out
System BrakePedalSensor	System BrakePedalSensor
Inport BrakePedalPosition_in_In	Inport BrakePedalPosition_in_In
Outport BrakePedalPosition_out_O	Outport BrakePedalPosition_out_…
System VehicleDynamics	System VehicleDynamics
Inport Wheel speed_In	Inport Wheel speed_In
Outport Vehicle torque_out1_Out	Outport Vehicle torque_out1_Out
Outport Vehicle speed_out_Out	Outport Vehicle speed_out_Out
System VehicleSpeedSensor	System VehicleSpeedSensor
Inport Vehicle speed_In	Inport Vehicle speed_In
Outport VehicleSpeed_out_Out	Outport VehicleSpeed_out_Out
System Wheel	System Wheel
Inport Brake_torque_In	Inport Brake_torque_In
Inport Vehicle_torque_In	Inport Vehicle_torque_In
Outport Wheel speed_Out	Outport Wheel speed_Out
System WheelSpeedSensor	System WheelSpeedSensor
Inport Wheel speed_In	Inport Wheel speed_In
Outport Wheel speed_out_Out	Outport Wheel speed_out_Out

Figure D1.10/ View of the working .simulink file after the modification. Everything seems correct.



```
MyEASTModel2.di2   *MyEASTModel1.di2   BBW.simulink
platform:/resource/EAST_Examples_SimulinkExchange/BBW.uml
  <Model> BBW
    <Class> SystemModel
    <Class> AnalysisLevel
      <Property> faa : BBW
    <Package> FunctionalComponents
      <Class> BBW
      <Class> ABS
        <Port> BrakeReference_in_In : DataType_0
        <Port> Vehicle speed_In : DataType_0
        <Port> Wheel speed_In : DataType_0
        <Port> Brake Reference_Out : DataType_0
      <Class> BrakeActuator
        <Port> req. torque_In : DataType_0
        <Port> brake torque_Out : DataType_0
      <Class> BrakeCalculator
        <Port> BrakePedalPosition_in_In : DataType_0
        <Port> RequestedBrakeTorque_out_Out : DataType_0
      <Class> BrakeController
        <Port> RequestedBrakeTorque_in_In : DataType_0
        <Port> BrakeReference_out_Out : DataType_0
      <Class> BrakePedal
        <Port> PedalPosition_Out : DataType_0
      <Class> BrakePedalSensor
        <Port> BrakePedalPosition_in_In : DataType_0
        <Port> BrakePedalPosition_out_Out : DataType_0
      <Class> VehicleDynamics
        <Port> Wheel speed_In : DataType_0
        <Port> Vehicle torque_out1_Out : DataType_0
        <Port> Vehicle speed_out_Out : DataType_0
      <Class> VehicleSpeedSensor
        <Port> Vehicle speed_In : DataType_0
        <Port> VehicleSpeed_out_Out : DataType_0
      <Class> Wheel
        <Port> Brake_torque_In : DataType_0
        <Port> Vehicle_torque_In : DataType_0
        <Port> Wheel speed_Out : DataType_0
      <Class> WheelSpeedSensor
```

Figure D1.11/ View of the .uml file that is obtained by converting the modified .simulink file. Everything seems OK.



```
??? Java exception occurred:
java.lang.NullPointerException

    at
    se.kth.md.simulinkExchange.modelConversion.simulink.preprocessing.LinesProcessor.obtain
    at
    se.kth.md.simulinkExchange.modelConversion.simulink.preprocessing.LinesProcessor.proces
    at
    se.kth.md.simulinkExchange.modelConversion.simulink.preprocessing.LinesProcessor.proces
    at
    se.kth.md.simulinkExchange.modelConversion.simulink.preprocessing.LinesProcessor.proces
    at
    se.kth.md.simulinkExchange.modelConversion.simulink.preprocessing.LinesProcessor.doIt(L
    at
    se.kth.md.simulinkExchange.modelConversion.simulink.preprocessing.SystemProcessor.creat
    at
    se.kth.md.simulinkExchange.modelConversion.simulink.preprocessing.SystemProcessor.proce
    at
    se.kth.md.simulinkExchange.modelConversion.simulink.preprocessing.SystemProcessor.doIt(S
    at
    se.kth.md.simulinkExchange.modelConversion.simulink.preprocessing.ModelProcessor.proces
    at
    se.kth.md.simulinkExchange.modelConversion.simulink.preprocessing.ModelProcessor.doIt(M

Error in ==> ModelCreatorRef>ModelCreatorRef.doIt at 114
            list = ModelProcessor.doIt(model);

Error in ==> ModelCreatorRef>ModelCreatorRef.fromFile at 40
            MC.doIt();

Error in ==> sl_customization>FileSelectOpen at 101
    ModelCreatorRef.fromFile(fullfile(pathname,filename));

>>
```

Figure D1.12/ View of Matlab Command Window when trying to open the .simulink file that is modified in Eclipse by clearing out the “connections” variables and their values

Case 3/ If we have a model(a .mdl file) that constitutes of "several same subsystems(severeal prototypes of the same type) which every subsystem have some connections with the others" we can save them as a .simulink file at Matlab. This case does not work correctly neither if we try to open that .simulink file(where we erase the .mdl file in Matlab to check if the same .mdl file which is used to create this .simulink file is created) from Matlab side again(where doing this is equivalent to create the .mdl file of the .simulink file) or from the Eclipse side.

When we try to open this file in Matlab, we lose the connections of the additional copies of the same prototypes(subsytems) with the others where only just one of the prototypes have the connections.In other words we only have the connections of one of the same typed prototypes and the other same type prototypes lose their connections.(In the Case 1 and 2 nothing was wrong when we re-open the .simulink files created in Matlab. There was no difference between the .mdl file created by this .simulink file and the .mdl file that created this .simulink file)

When we try to open this from Eclipse again the compiler complains about the same variable. When we correct the errors again as we did in the 2nd case, we can again convert the file to a .uml. When we make the convertion we have a model in hand that only has the connections of the same typed prototypes as it was in Matlab side. Everything seems OK other than the connections but.

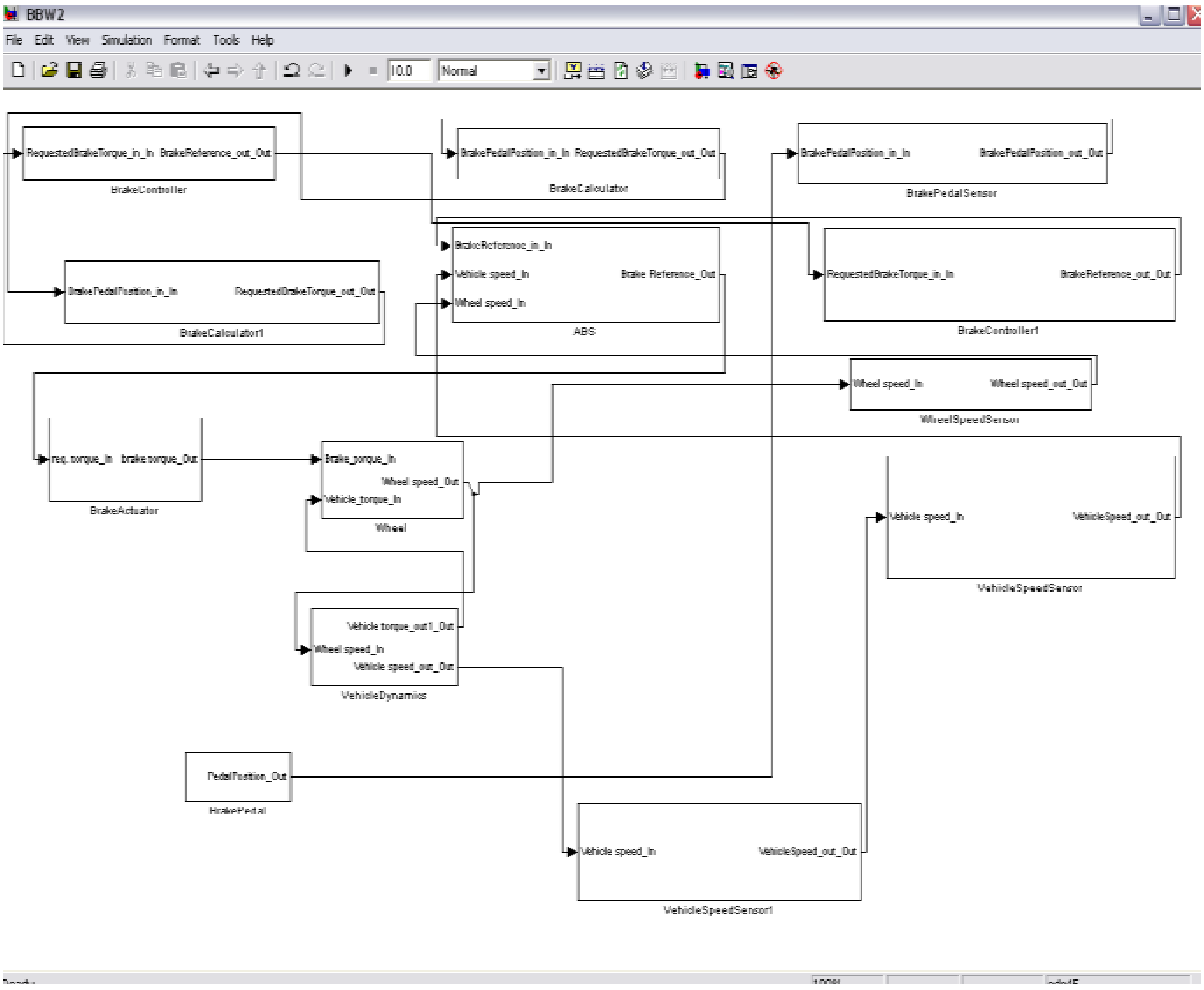
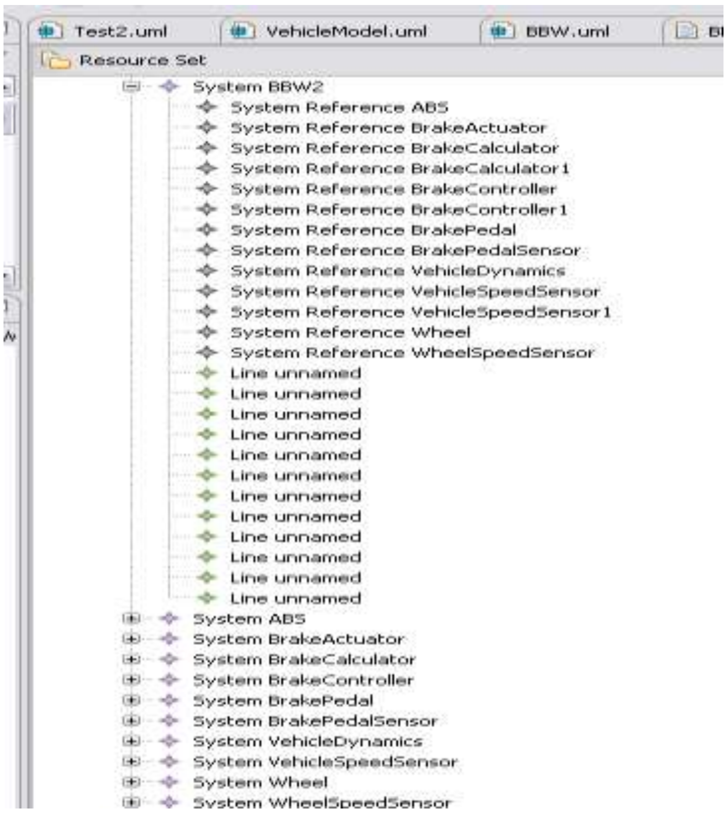
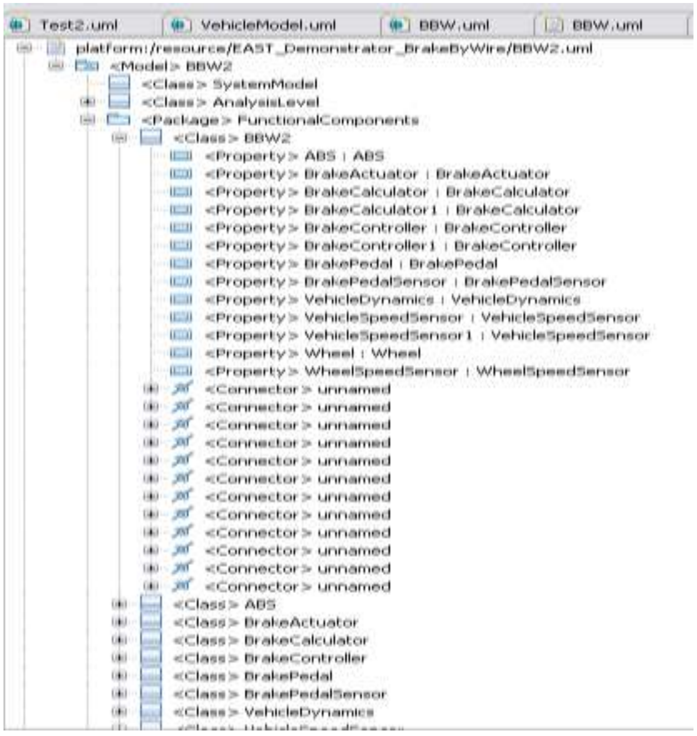


Figure D1.13/ View of the designed system in Simulink. This system has several prototypes which are same type. All the prototypes are connected to each other.



```
Test2.uml    VehicleModel.uml    BBW.uml
Resource Set
⊟ System BBW2
    System Reference ABS
    System Reference BrakeActuator
    System Reference BrakeCalculator
    System Reference BrakeCalculator1
    System Reference BrakeController
    System Reference BrakeController1
    System Reference BrakePedal
    System Reference BrakePedalSensor
    System Reference VehicleDynamics
    System Reference VehicleSpeedSensor
    System Reference VehicleSpeedSensor1
    System Reference Wheel
    System Reference WheelSpeedSensor
    Line unnamed
    Line unnamed
    Line unnamed
    Line unnamed
    Line unnamed
    Line unnamed
    Line unnamed
    Line unnamed
    Line unnamed
    Line unnamed
    Line unnamed
    Line unnamed
⊞ System ABS
⊞ System BrakeActuator
⊞ System BrakeCalculator
⊞ System BrakeController
⊞ System BrakePedal
⊞ System BrakePedalSensor
⊞ System VehicleDynamics
⊞ System VehicleSpeedSensor
⊞ System Wheel
⊞ System WheelSpeedSensor
```

Figure D1.14/ View of the .simulink represantation of the system opened in Eclipse(Note that the needed modification done to the .simulink, which is created in Matlab, that is done in also Case 2 hasn't been shown). All the subsystems are in hand.



```
Test2.uml    VehicleModel.uml    BBW.uml    BBW.uml
⊟ platform:/resource/EAST_Demonstrator_BrakeByWire/BBW2.uml
  ⊟ <Model> BBW2
      <Class> SystemModel
    ⊞ <Class> AnalysisLevel
    ⊟ <Package> FunctionalComponents
      ⊟ <Class> BBW2
          <Property> ABS : ABS
          <Property> BrakeActuator : BrakeActuator
          <Property> BrakeCalculator : BrakeCalculator
          <Property> BrakeCalculator1 : BrakeCalculator
          <Property> BrakeController : BrakeController
          <Property> BrakeController1 : BrakeController
          <Property> BrakePedal : BrakePedal
          <Property> BrakePedalSensor : BrakePedalSensor
          <Property> VehicleDynamics : VehicleDynamics
          <Property> VehicleSpeedSensor : VehicleSpeedSensor
          <Property> VehicleSpeedSensor1 : VehicleSpeedSensor
          <Property> Wheel : Wheel
          <Property> WheelSpeedSensor : WheelSpeedSensor
        ⊞ <Connector> unnamed
        ⊞ <Connector> unnamed
        ⊞ <Connector> unnamed
        ⊞ <Connector> unnamed
        ⊞ <Connector> unnamed
        ⊞ <Connector> unnamed
        ⊞ <Connector> unnamed
        ⊞ <Connector> unnamed
        ⊞ <Connector> unnamed
        ⊞ <Connector> unnamed
        ⊞ <Connector> unnamed
        ⊞ <Connector> unnamed
      ⊞ <Class> ABS
      ⊞ <Class> BrakeActuator
      ⊞ <Class> BrakeCalculator
      ⊞ <Class> BrakeController
      ⊞ <Class> BrakePedal
      ⊞ <Class> BrakePedalSensor
      ⊞ <Class> VehicleDynamics
```

Figure D1.15/ View of the .uml represantation of the model. Everything seems alright except the connections. There are only 12 connections in the .uml file where it has to be 18. This is due to the fact that, only one of the subsystem's connections are included among the other subsystems that are typed by same type. There were 3 same typed prototypes which had totally 6 connections which is not included in the current model.

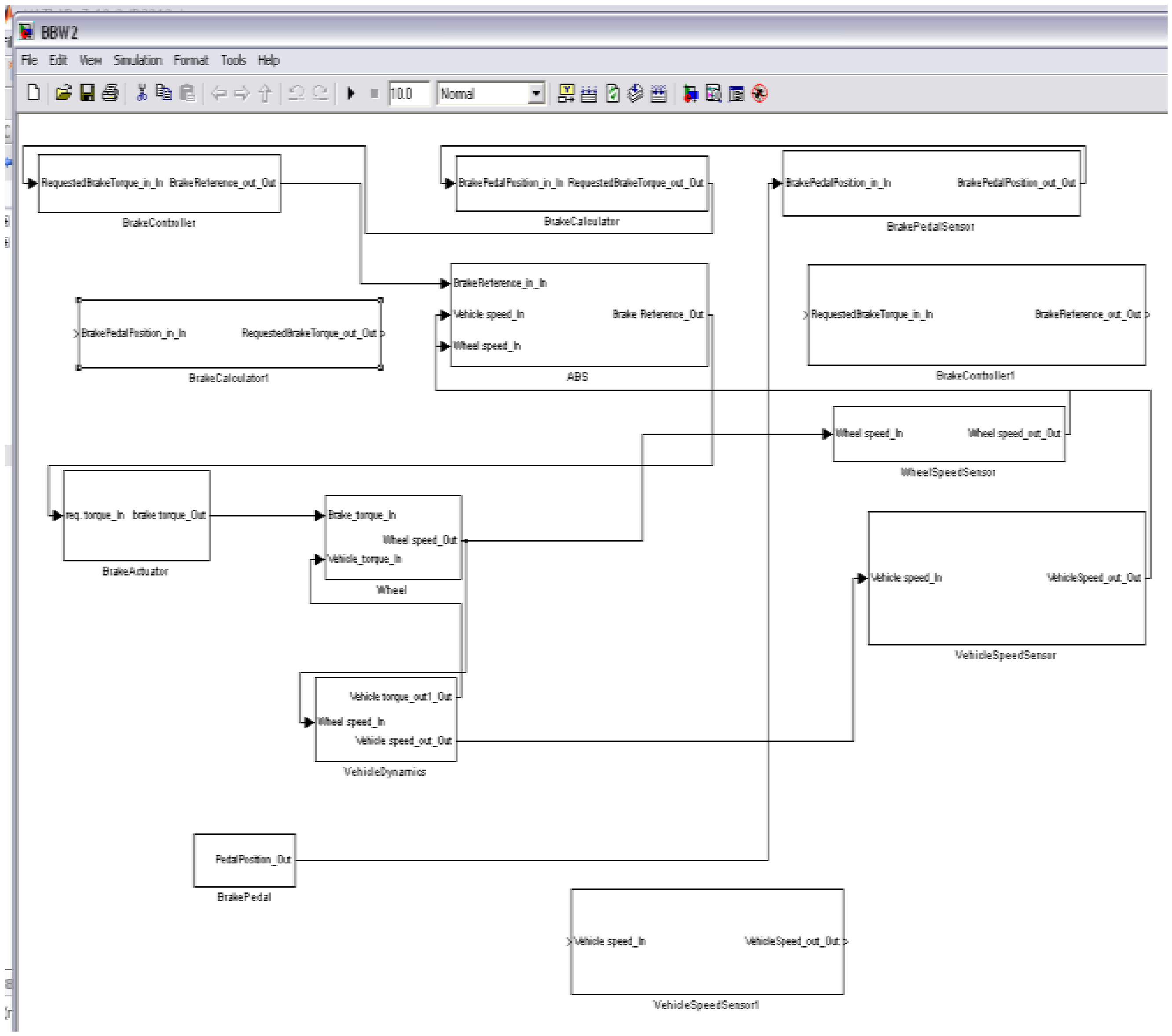


Figure D1.16/ The view of the .mdl file that is created by the .simulink file in hand. This .simulink file is supposed to be the representation of the .mdl file that it is created from but notice that we lost the connections of the 3 prototypes. This is because, in the system there are other prototypes that are typed by the same type. So when the .simulink is created from the .mdl, all the subsystems are included but not all the connections.

Case 4/ If we have a model that is similar to Case 3 but without any connections, then everyhing works fine.

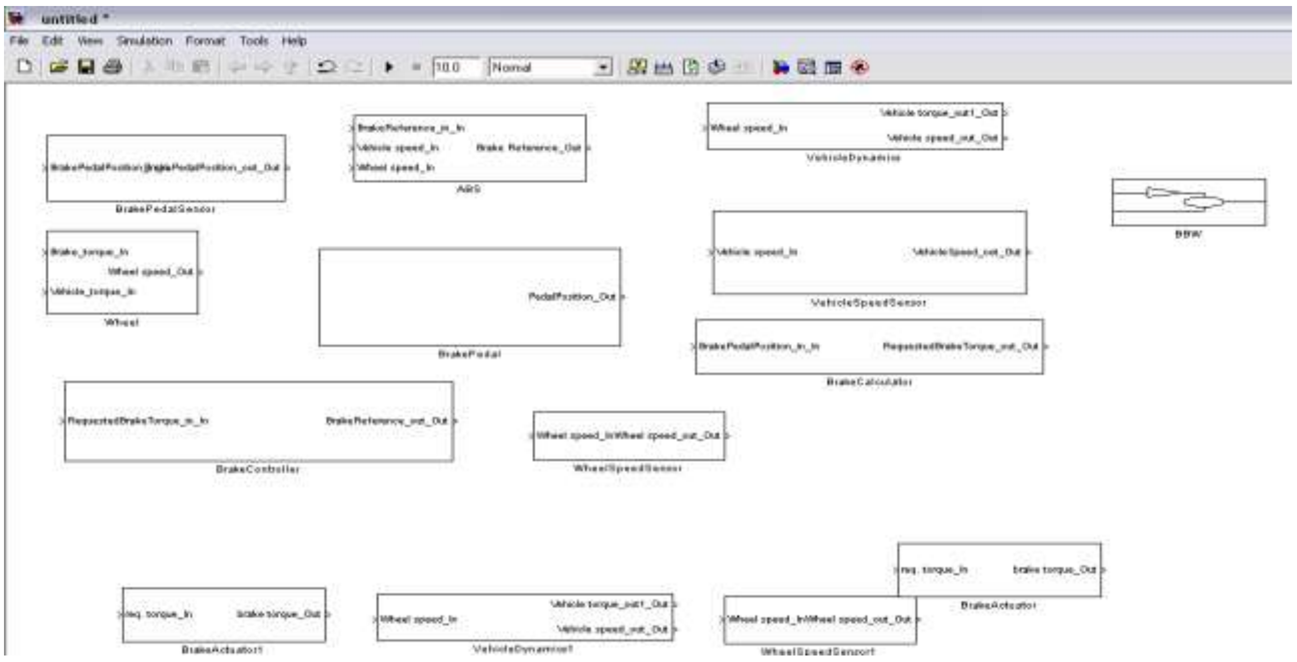


Figure D1.17/ View of the system in Matlab side

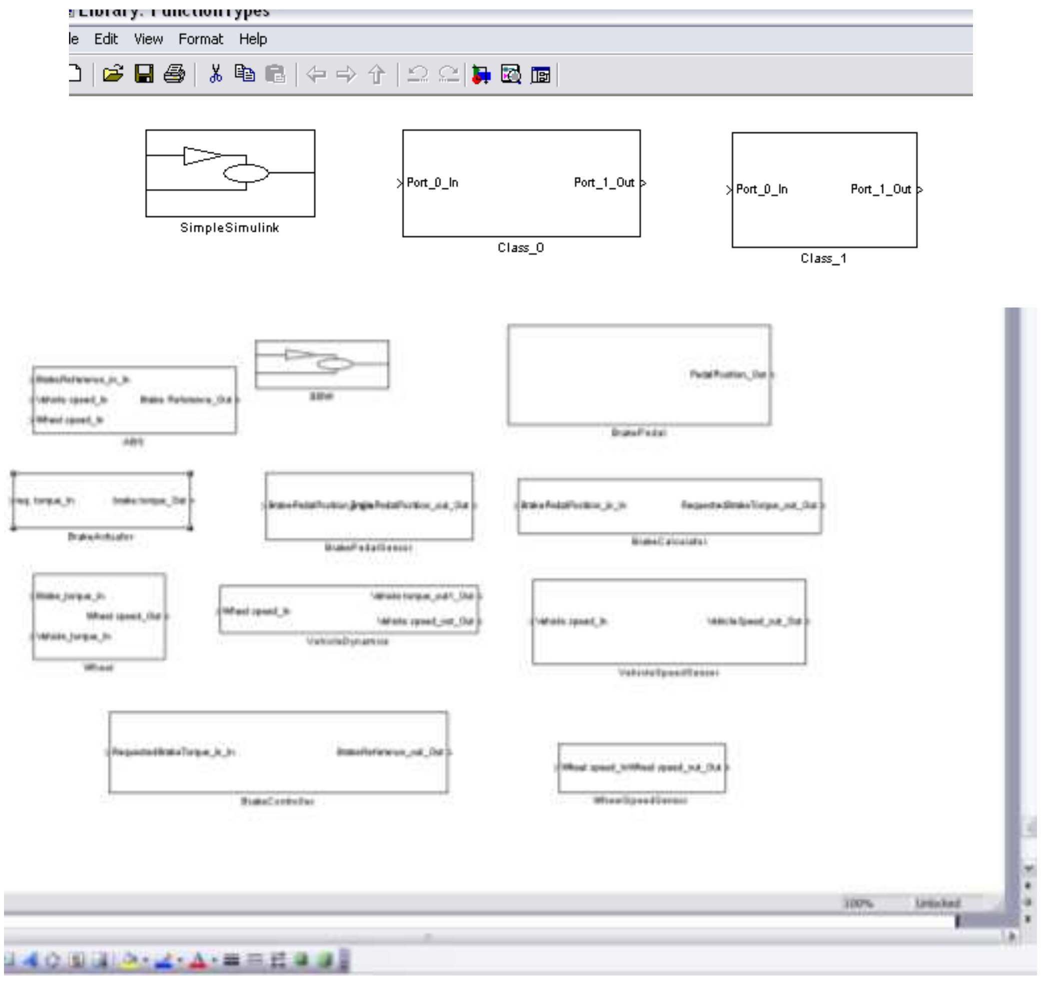
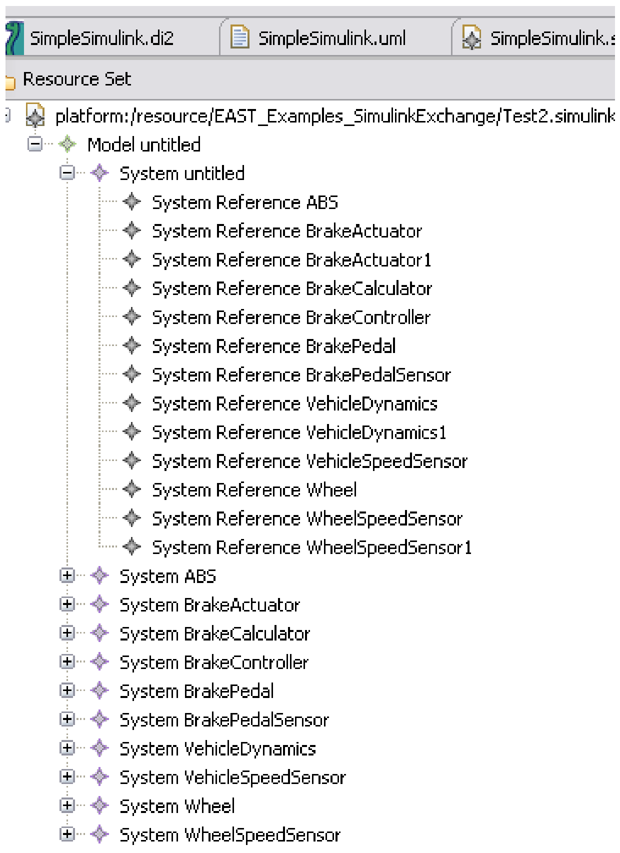
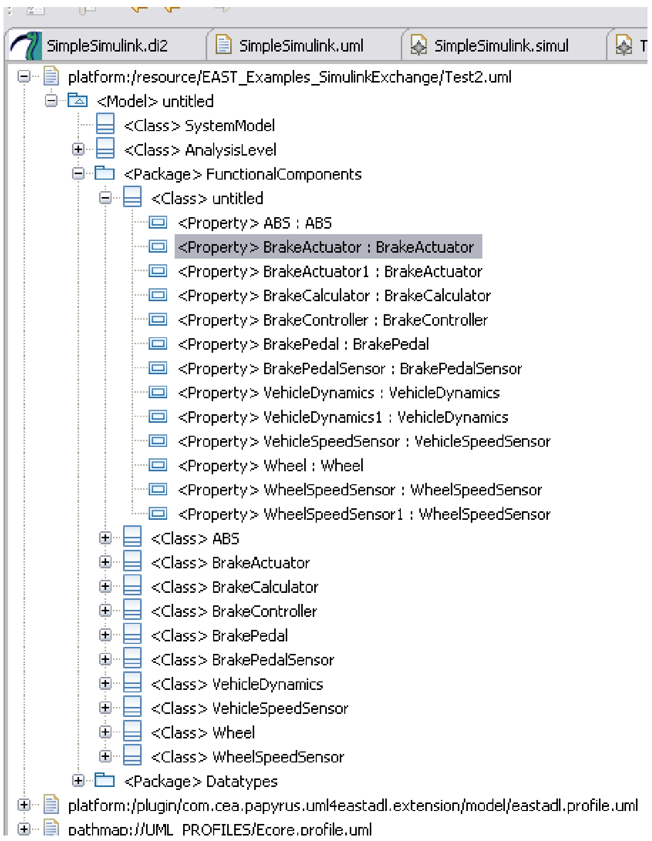


Figure D1.18/ View of the FunctionTypes.mdl in Matlab



```
SimpleSimulink.di2 | SimpleSimulink.uml | SimpleSimulink.s
Resource Set
platform:/resource/EAST_Examples_SimulinkExchange/Test2.simulink
  Model untitled
    System untitled
      System Reference ABS
      System Reference BrakeActuator
      System Reference BrakeActuator1
      System Reference BrakeCalculator
      System Reference BrakeController
      System Reference BrakePedal
      System Reference BrakePedalSensor
      System Reference VehicleDynamics
      System Reference VehicleDynamics1
      System Reference VehicleSpeedSensor
      System Reference Wheel
      System Reference WheelSpeedSensor
      System Reference WheelSpeedSensor1
    System ABS
    System BrakeActuator
    System BrakeCalculator
    System BrakeController
    System BrakePedal
    System BrakePedalSensor
    System VehicleDynamics
    System VehicleSpeedSensor
    System Wheel
    System WheelSpeedSensor
```

Figure D1.19/ View of the .simulink file of the model opened in Eclipse. Everything seems fine.



```
SimpleSimulink.di2 | SimpleSimulink.uml | SimpleSimulink.simul | T
platform:/resource/EAST_Examples_SimulinkExchange/Test2.uml
  <Model> untitled
    <Class> SystemModel
    <Class> AnalysisLevel
    <Package> FunctionalComponents
      <Class> untitled
        <Property> ABS : ABS
        <Property> BrakeActuator : BrakeActuator
        <Property> BrakeActuator1 : BrakeActuator
        <Property> BrakeCalculator : BrakeCalculator
        <Property> BrakeController : BrakeController
        <Property> BrakePedal : BrakePedal
        <Property> BrakePedalSensor : BrakePedalSensor
        <Property> VehicleDynamics : VehicleDynamics
        <Property> VehicleDynamics1 : VehicleDynamics
        <Property> VehicleSpeedSensor : VehicleSpeedSensor
        <Property> Wheel : Wheel
        <Property> WheelSpeedSensor : WheelSpeedSensor
        <Property> WheelSpeedSensor1 : WheelSpeedSensor
      <Class> ABS
      <Class> BrakeActuator
      <Class> BrakeCalculator
      <Class> BrakeController
      <Class> BrakePedal
      <Class> BrakePedalSensor
      <Class> VehicleDynamics
      <Class> VehicleSpeedSensor
      <Class> Wheel
      <Class> WheelSpeedSensor
    <Package> Datatypes
platform:/plugin/com.cea.papyrus.uml4eastadl.extension/model/eastadl.profile.uml
pathmap://UML_PROFILES/Ecore.profile.uml
```

Figure D1.20/ View of the .uml file that is obtained from the .simulink file of the model. Everything seems still fine.

FROM ECLIPSE 2 MATLAB

Case 1/ If we have a Functional Analysis Architecture that does not have connections but only Prototypes which have ports, no problem is experienced through creating the .simulink file from the Eclipse side and opening that file from the Matlab side. Everything seems working and correct.

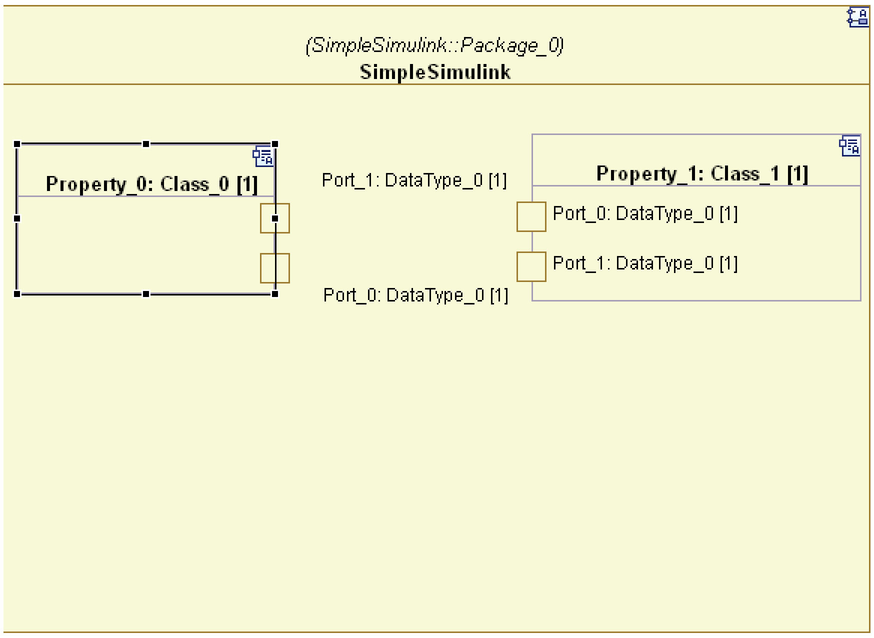
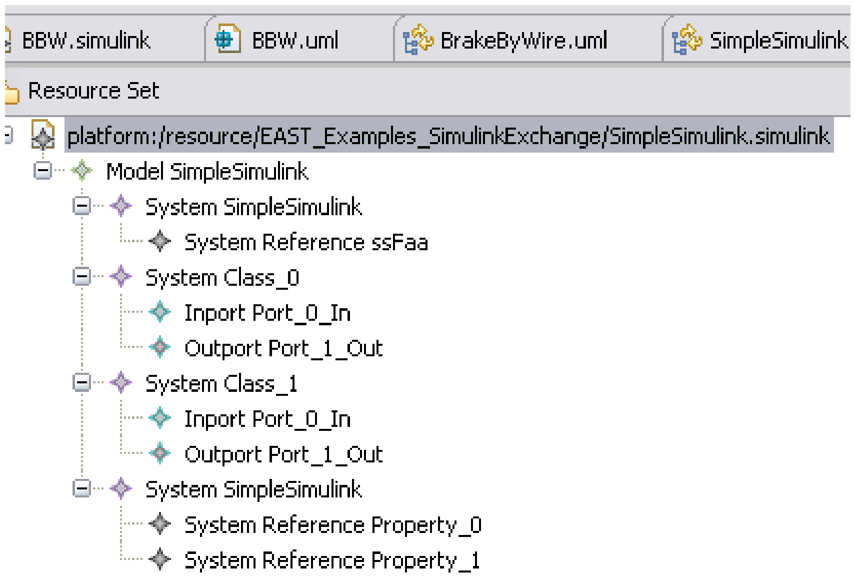


Figure D1.21/ View of the model that is created in Eclipse. Note that this model has only 2 prototypes and several ports but no connection.



```
BBW.simulink | BBW.uml | BrakeByWire.uml | SimpleSimulink
Resource Set
platform:/resource/EAST_Examples_SimulinkExchange/SimpleSimulink.simulink
  Model SimpleSimulink
    System SimpleSimulink
      System Reference ssFaa
    System Class_0
      Inport Port_0_In
      Outport Port_1_Out
    System Class_1
      Inport Port_0_In
      Outport Port_1_Out
    System SimpleSimulink
      System Reference Property_0
      System Reference Property_1
```

Figure D1.22/ View of the .simulink file created in Eclipse. The representation seems alright.

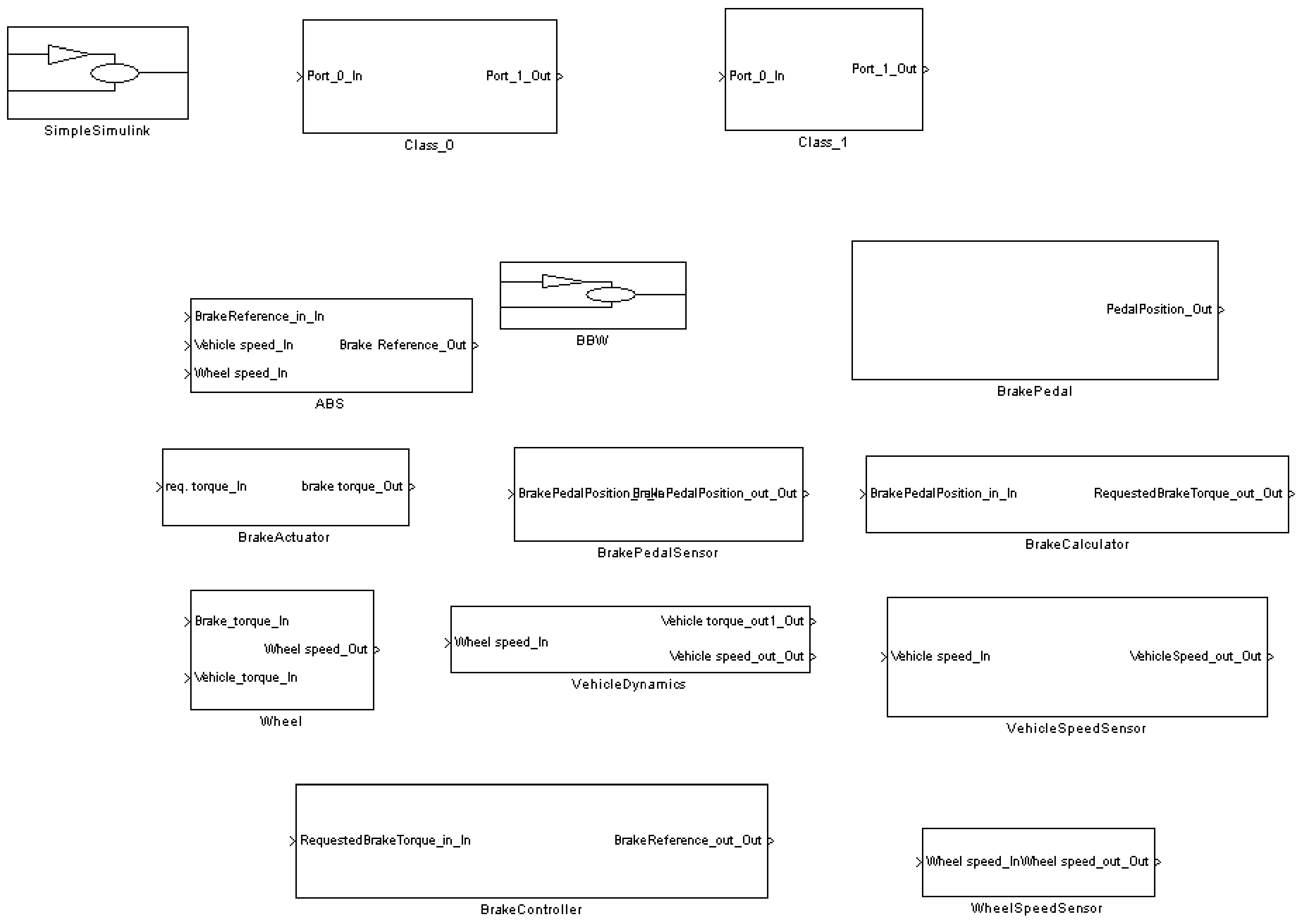


Figure D1.22/ The functiontypes.mdl file. All the building blocks of the model are added(Class 0 and Class 1).

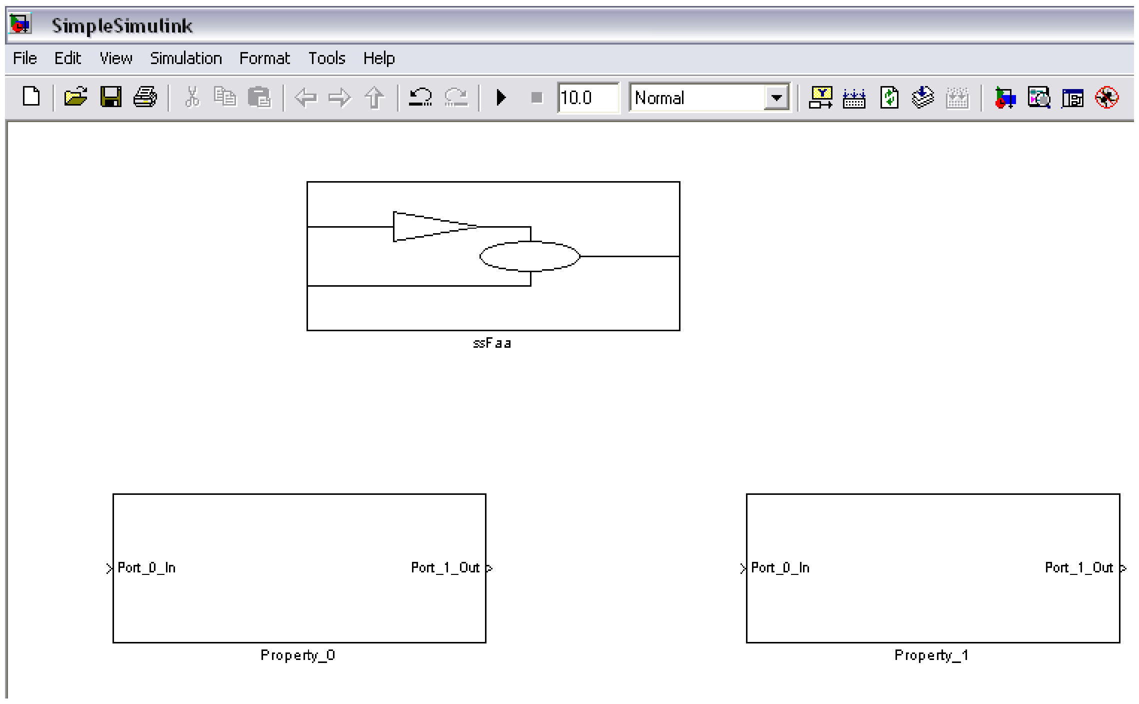


Figure D1.23/ .mdl representation of the .uml file which we begin with. The conversion seems successful.

Case 2/ If we have a Functional Analysis Architecture with Prototypes that has connections with eachother, no problem is experienced through creating the .simulink file from the Eclipse side and opening that file from the Matlab side. But the .simulink file created in Eclipse does not hold the information of the connections so that the .mdl file created in Matlab by this .simulink file. So we do not have the connections in Matlab side.

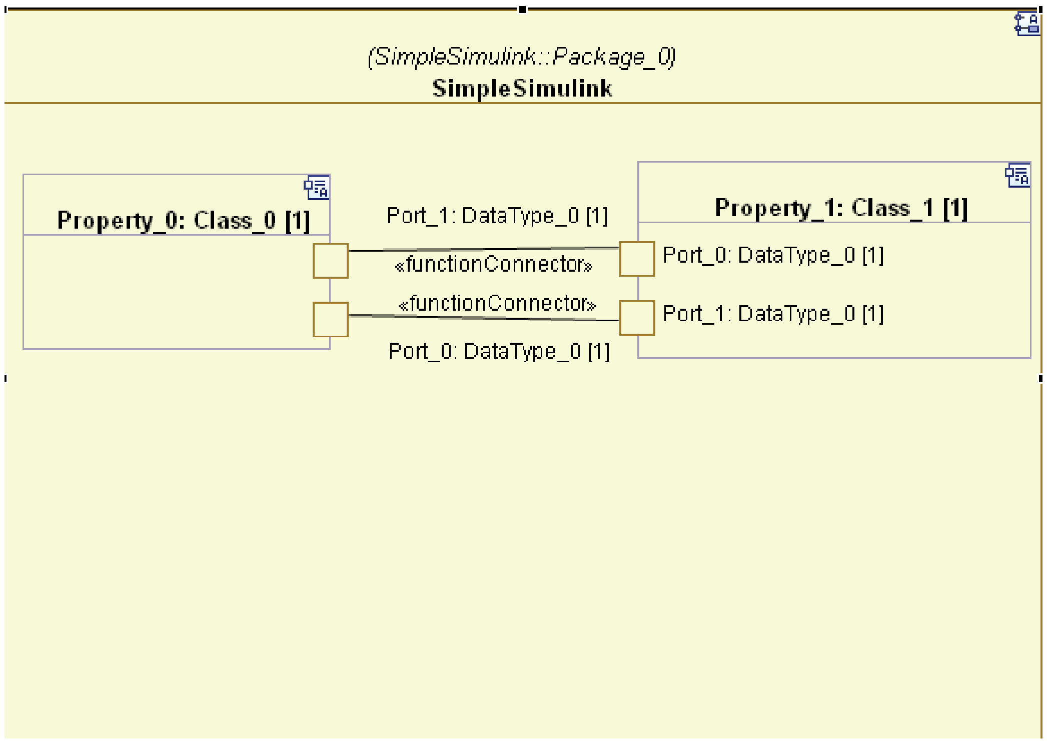
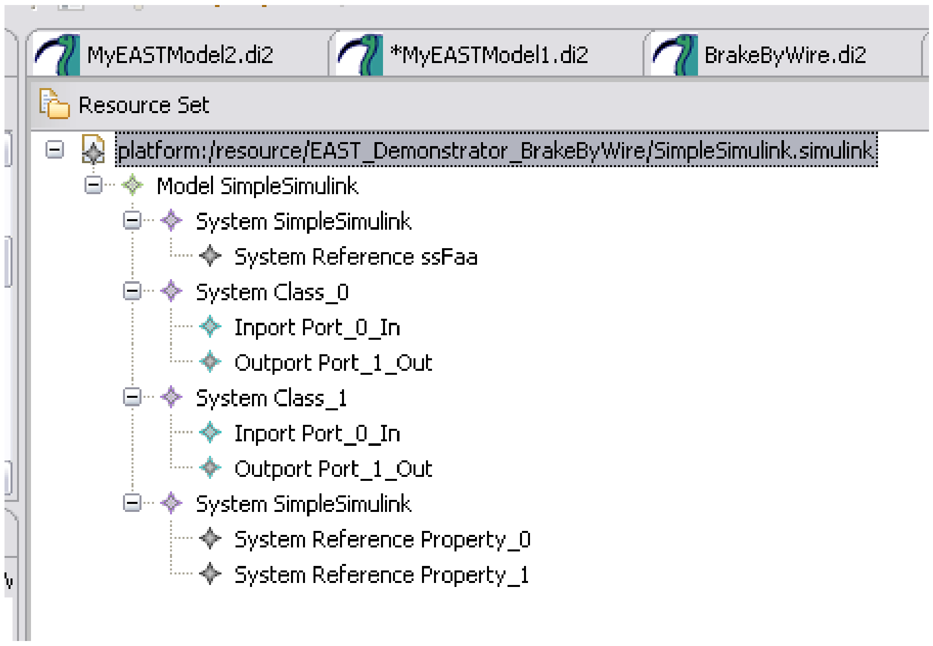
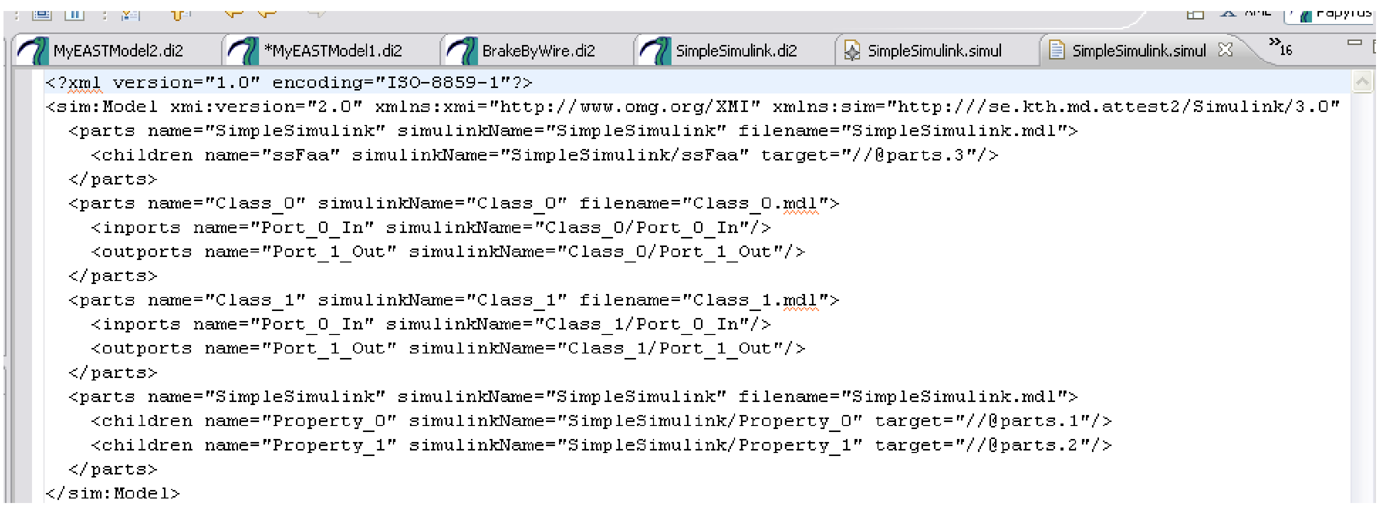


Figure D1.24/ The same model that is designed in Case 1 with the additon of 2 function connectors.



platform:/resource/EAST_Demonstrator_BrakeByWire/SimpleSimulink.simulink
- Model SimpleSimulink
  - System SimpleSimulink
    - System Reference ssFaa
  - System Class_0
    - Inport Port_0_In
    - Outport Port_1_Out
  - System Class_1
    - Inport Port_0_In
    - Outport Port_1_Out
  - System SimpleSimulink
    - System Reference Property_0
    - System Reference Property_1

Figure D1.25/ It can be seen that same .simulink file is also obtained from this model which is not expected. The connections are missing.



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<sim:Model xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:sim="http:///se.kth.md.attest2/Simulink/3.0"
  <parts name="SimpleSimulink" simulinkName="SimpleSimulink" filename="SimpleSimulink.mdl">
    <children name="ssFaa" simulinkName="SimpleSimulink/ssFaa" target="//@parts.3"/>
  </parts>
  <parts name="Class_0" simulinkName="Class_0" filename="Class_0.mdl">
    <inports name="Port_0_In" simulinkName="Class_0/Port_0_In"/>
    <outports name="Port_1_Out" simulinkName="Class_0/Port_1_Out"/>
  </parts>
  <parts name="Class_1" simulinkName="Class_1" filename="Class_1.mdl">
    <inports name="Port_0_In" simulinkName="Class_1/Port_0_In"/>
    <outports name="Port_1_Out" simulinkName="Class_1/Port_1_Out"/>
  </parts>
  <parts name="SimpleSimulink" simulinkName="SimpleSimulink" filename="SimpleSimulink.mdl">
    <children name="Property_0" simulinkName="SimpleSimulink/Property_0" target="//@parts.1"/>
    <children name="Property_1" simulinkName="SimpleSimulink/Property_1" target="//@parts.2"/>
  </parts>
</sim:Model>
```

Figure D1.26/ It can be seen that there is no description or specification about the connections when opening the .simulink file by text editor.

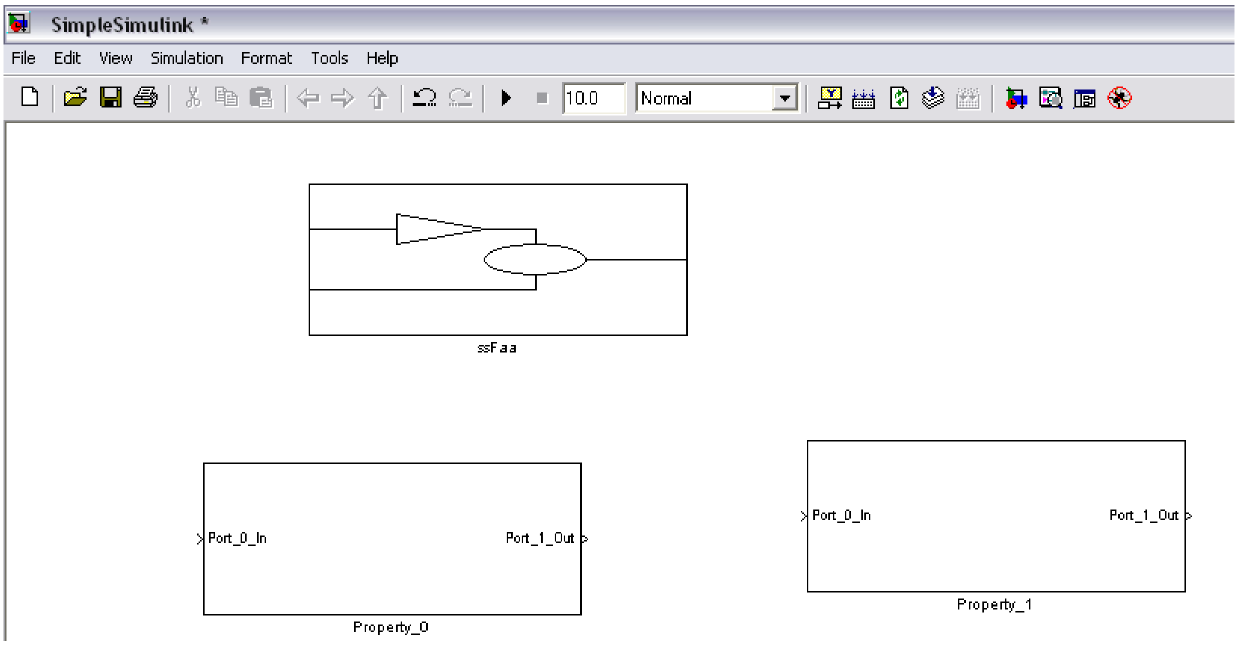


Figure D1.27/ View of the .mdl obtained by the model. We don't have the connections.