MAENAD

Grant Agreement 260057

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

Report type — Deliverable D5.3.1

Report name — EATOP: An EAST-ADL Tool Platform for Eclipse

Dissemination level	**PU**
Status	**Final**
Version number	**3.0**
Date of preparation	**2014-02-14**

Authors

Editor	E-mail
Daniels Umanovskis	daniels.umanovskis@arccore.com

Authors	E-mail
Daniels Umanovskis	daniels.umanovskis@arccore.com
Stefan Voget	stefan.voget@continental-corporation.com

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

The Consortium

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

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

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

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

Revision chart and history log

Version	Date	Reason
0.1	2012-06-17	Initial version
0.2	2012-07-09	EATOP project description added
1.0	2012-08-31	First Release
1.1	2013-08-31	Second Release for review
2.0	2013-09-30	Second Release for M36
3.0	2014-02-14	Third Release for final tooling

Approval	Date
Henrik Lönn	2014-02-20

Table of contents

1 Introduction

1.1 Purpose of document

This document constitutes one part of Deliverable D5.3.1 on Tool adaptations for EAST-ADL. It introduces the Eclipse initiative “EATOP” which is an abbreviation for “EAST-ADL Tool Platform”.

EATOP is an Eclipse-based implementation of the EAST-ADL meta-model. EAST-ADL is a domain specific language established in the automotive industry. This document describes the adaptation of the Eclipse / Sphinx framework which constitutes EATOP.

1.2 Introduction to contents of the document

This document describes the general background of EATOP, the technical framework on which it is built, and gives a brief introduction to the use of EATOP. The document does not serve as an exhaustive user guide for EATOP but is intended to provide a good introductory-level overview.

1.3 Intended outcome

The general outcome will be an extensible platform, which provides model editors and analysis plug-ins, which will support the EAST-ADL language for functional-, system-, software- and hardware-description. For the integration of already existing tools, which work on proprietary meta-models, model to model transformations will be provided.

With the work of EATOP the usage of EAST-ADL should be simplified and the number of already existing intermediate formats should be reduced.

2 Background

2.1 EAST-ADL

EAST-ADL is a domain specific language to model functional-, system-, software-, and hardware-architecture in the automotive domain. EAST-ADL has been created by the ITEA (www.itea2.org) funded project EAST/EEA. Further development has been done in two funded projects ATESST and ATESST2. Since 2010 a European funded project MAENAD (www.maenad.eu) maintains and extends the language with respect to electrified vehicles and safety development lifecycle modeling. For long-term maintenance and dissemination the EAST-ADL association (www.east-adl.info) has been founded. It maintains the meta-model definition and makes the latest version available to the public.

Until now, the EAST-ADL has been used in several tools:

- In the Eclipse MDT project Papyrus a UML2 profile of EAST-ADL has been provided
- TopCased initiative and the project OPEES – both are established in the airbus industry – are using EAST-ADL
- The commercial tool Volcano Vehicle Systems Architect (VSA) from Mentor Graphics provides editors for EAST-ADL modeling
- The commercial tool MetaEdit+ from MetaCase provides a meta-model implementation of EAST-ADL and generic editors of this meta-model
- The commercial tool System Weaver from Systemite provides editors for the EAST-ADL modeling
- Further tools exist from Arcticus, Symtavision, Rapita, RealTime@Work, Chronos

EATOP shall support the work of the EAST-ADL association by providing an Eclipse based tool platform implementation of the EAST-ADL standard. Until now there had been several initiatives to create Eclipse based implementations of EAST-ADL. Goal of EATOP is to reconcile such initiatives in Eclipse.

2.2 Relationship to AUTOSAR

AUTOSAR is a design standard in the automotive industry. It focuses on the software architecture for electronic control units (ECU) in road vehicles, its deployment to networked ECUs in vehicles, and the configuration of the basic software in such ECUs. The AUTOSAR Tool Platform User Group (ARTOP) develops common base functionality for creating modeling tools supporting the AUTOSAR standard. ARTOP essentially encompasses implementations of the different releases of the AUTOSAR meta-model plus a rich set of services and components for managing and processing AUTOSAR models.

The EAST-ADL meta-model has a close relationship to the AUTOSAR meta-model as both address the model based development part in a process. East-ADL covers the function- and system-architecture and AUTOSAR acts with the system configuration and software architecture.

Therefore, EATOP will have a close relationship to the ARTOP (www.artop.org) initiative.

Figure 1 shows the principle how ARTOP generates the EMF- / Ecore-based meta-model implementation. Similar steps will be realized in EATOP for the EAST-ADL meta-model.

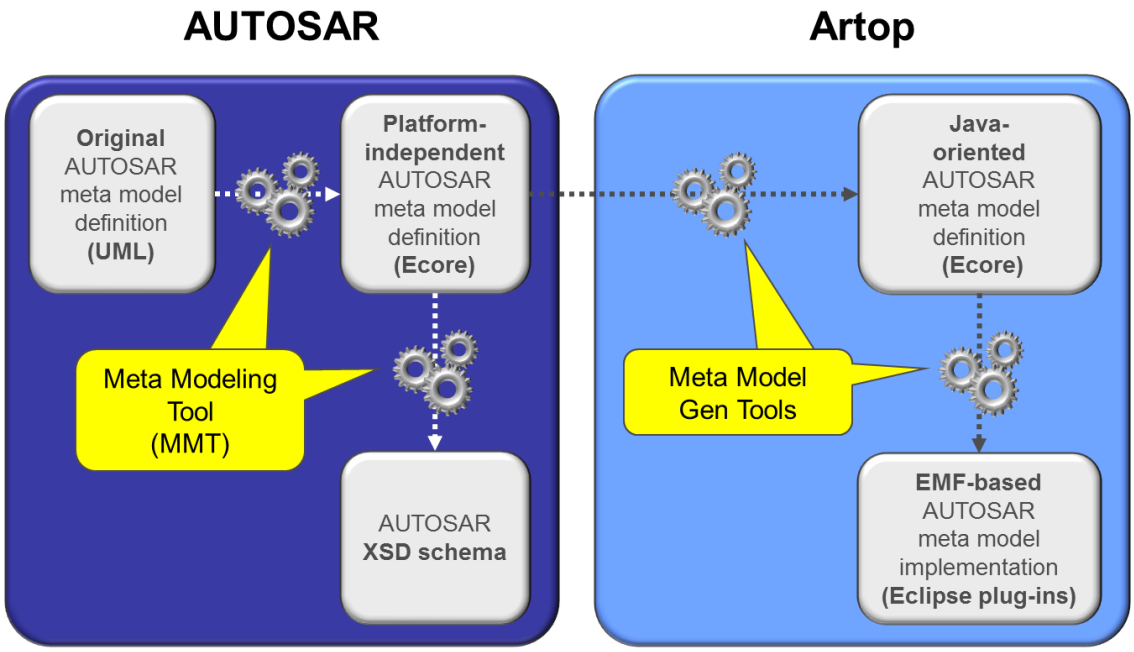

Figure 1: Principle relationship between meta-model and EMF-/Ecore-based implementation shown with the help of the example “AUTOSAR”.

3 Scope

EATOP is an infrastructure platform implementation of common base functionality for development tools that are used to design EAST-ADL compliant systems. This includes at least

- An EAST-ADL meta-model implementation supports several versions of EAST-ADL meta-model releases. These releases are published by the EAST-ADL association as an Enterprise Architect project and a XML schema. This input is taken to generate an Ecore-based meta-model implementation.
- Serialization is supported to enable file- and repository-based persistency of EAST-ADL models. Serializing and de-serializing EAST-ADL models to and from EAST-ADL XML files and databases.
- Refactoring contains a number of mechanisms to modify EAST-ADL models in a safe way.
- Workspace Management supports managing of EAST-ADL models, which are spread over more than one XML file.
- Model to model transformation between the different abstraction layers within the EAST-ADL meta-model.
- Further utilities simplify the handling of EAST-ADL models.
- To enable a seamless workflow in a development process, interfacing and model exchange with
    - Requirements Engineering (via ReqIF),
    - Software modeling (e.g. via AUTOSAR) and
    - HW modeling (e.g. via IPXACT) is enabled.
- Specific platform developments enable safety analysis and timing modeling.
- Consolidation of bridges between EATOP and Papyrus and the synchronization of EAST-ADL EMF API with the profile implementation.
- Variability management, supporting both the definition of variant-rich EAST-ADL models, as well as derivation of fully/partly configured instances of these models.
- Serves as an interoperability platform with domain independent abstractions of EAST-ADL like the CESAR reference technology platform (www.cesarproject.eu) or the MBAT reference technology platform (www.mbat-artemis.eu).

4 Description

4.1 Components

EATOP provides a set of EAST-ADL model editors. The components realizing these services already exist and are contributed along with the creation of this project (see Code contributions).

- An EMF based meta-model implementation of EAST-ADL.
- A set of graphical design editors for the different abstraction layers of an EAST-ADL based model. These editors enable a developer to design an automotive system on vehicle level, through the functional analysis level down to the functional design level and hardware level. A package diagram is the root diagram and the entry point for each EAST-ADL model. On each abstraction level several editors are provided to support the type/prototype principle of the meta-model.

As ARTOP is based on the Eclipse project Sphinx (www.eclipse.org/sphinx), so is EATOP. Sphinx eases the creation of integrated modeling tool environments supporting individual or multiple modeling languages. Figure 2 presents the Sphinx-based architecture.

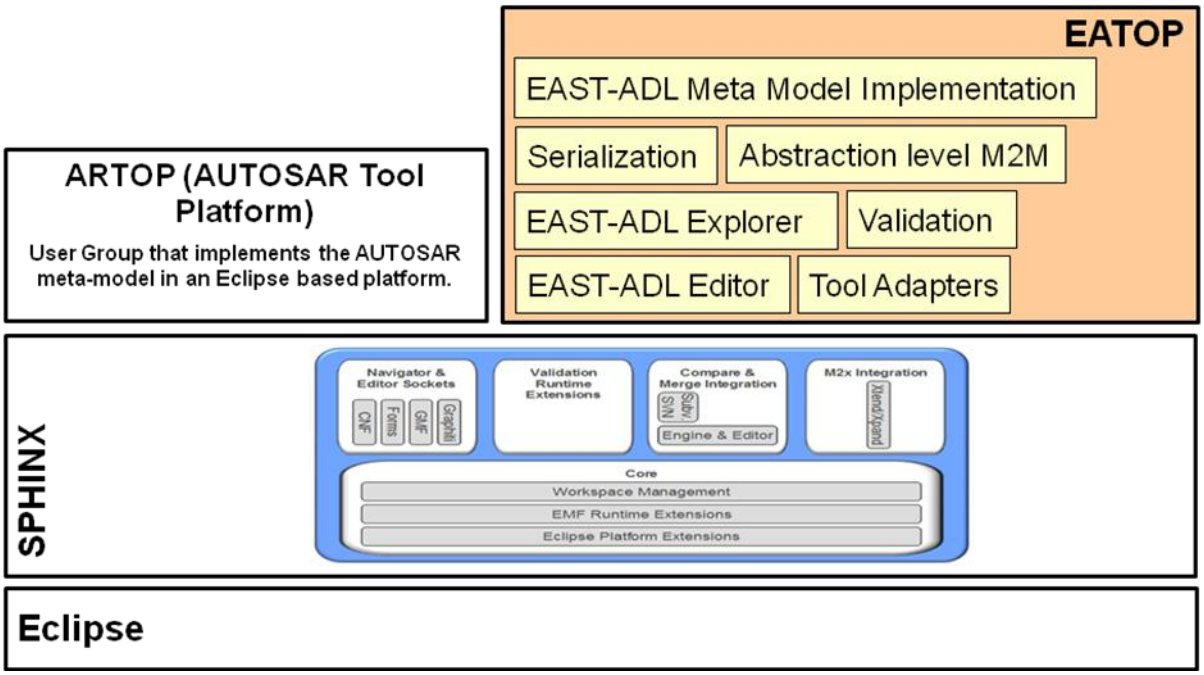

Figure 2: The architecture of EATOP resembles ARTOP and is based on Sphinx and Eclipse.

- The **EAST-ADL meta-model is implemented** using the Java programming language and the Eclipse Modeling Framework (EMF). The implementation supports several releases of the EAST-ADL meta-model. The goal is to provide meta-model implementations in EATOP close to the point in time when the EAST-ADL association releases a new meta-model version.
- The **Serialization component** provides file- and repository-based persistence for EAST-ADL models, serializing and de-serializing EAST-ADL models to and from EAST-ADL XML-files and databases.

- The **abstraction level M2M** component realizes a set of model to model converters that enable the generation of a skeleton on a more concrete meta-model level based on a model on a more abstract meta-model level.
- The **EAST-ADL Explorer** organizes the model elements and provides an adapted view of the generic Eclipse explorer.
- A **validation** engine partially based on Sphinx runtime extensions that minimizes the effort to create validation constraints for EAST-ADL models. A set of constraints is included and specified in the OCL language.
- Specific **tool adapters** enable the integration of the EATOP components into existing commercial or non-commercial tools. These adapters either enable import/export of models between the tools and EATOP or enable the integration using the Eclipse plug-in concept.

One core part of the work in EATOP is the translation of the EAST-ADL meta-model definition, which is provided in the form of a XML model authorized with Enterprise Architect, into an EMF-/Ecore-based implementation. The following picture summarizes the most important relationships.

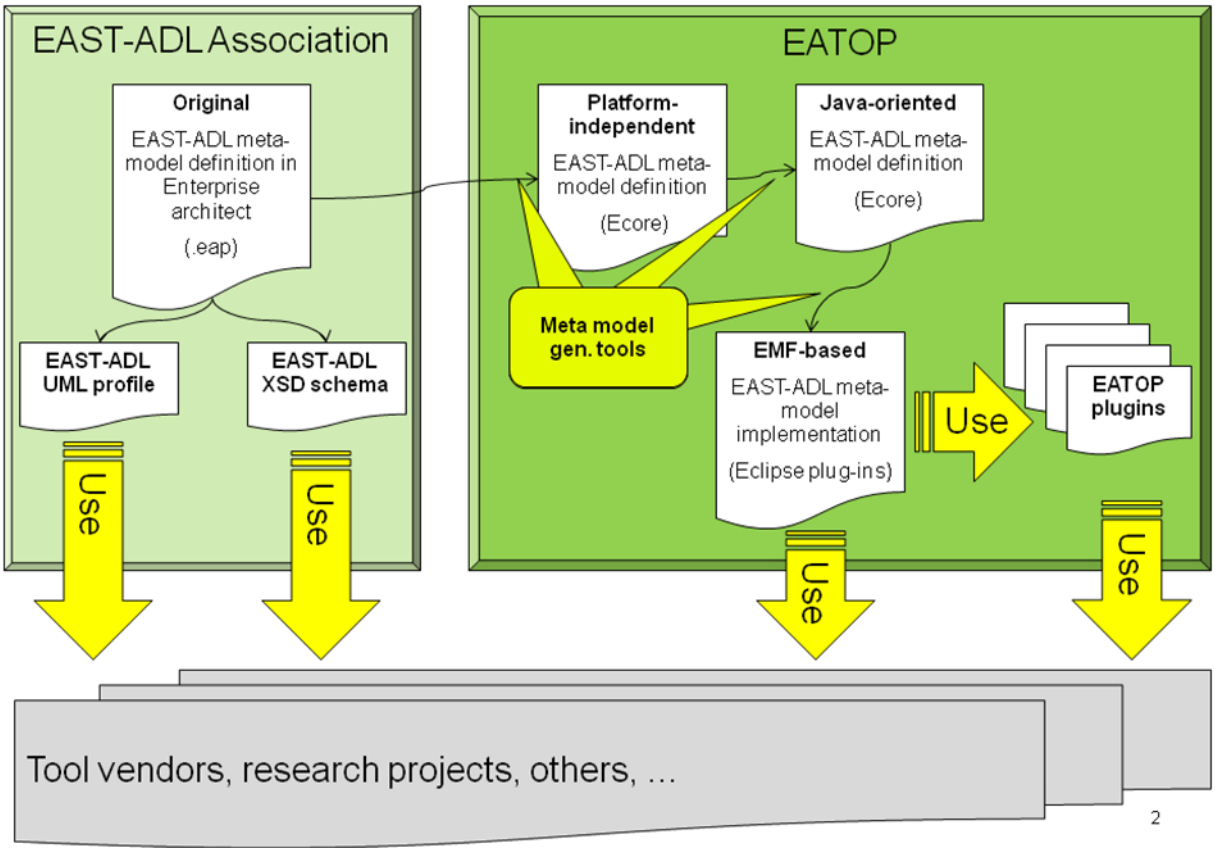

Figure 3: A tool chain for providing an EAST-ADL meta-model implementation ranging from language definition, via language development tools to user modeling tools. Intermediate artifacts are also indicated to show the language information needed to implement tools.

Figure 3 shows the different steps. These are described below.

1. Enterprise Architect is used to model the EAST-ADL meta-model in UML. This meta-model contains all elements of the language, including their attributes and relationships. Diagrams

are used to represent these elements. The resulting meta-model is in the Enterprise Architect project format (EAP). Enterprise Architect can also export to XMI format.

2. The EAST-ADL association already provides a schema definition and a UML2 profile which can directly be used. E.g. the Eclipse project Papyrus uses the UM2 profile.
3. EATOP generates an API to access and manipulate a model. This API can be used for building all kinds of applications on top of EAST-ADL. Examples are form-based and graphical EAST-ADL editors. EATOP provides a set of such editors. These can be used as starting point for full-featured EAST-ADL editors in commercial tools or custom editors developed by the EAST-ADL tool users.

5 Relationship to other Eclipse Projects

- EATOP is built on top of the Eclipse Platform and EMF.
- EATOP also uses complementary components of the Eclipse Modeling Project (EMP). These include EMF Transaction, EMF Validation and GMF.
- The architecture is set up on top of Sphinx mainly using the workspace management, the validation runtime extensions, compare & merge integration, and the EMF runtime and Eclipse platform extensions. Where possible, it follows the patterns set by EATOP.
- EATOP itself is an official Eclipse Project in the Modeling category.

6 Environment Setup

6.1 Introduction

This chapter describes how to set up a development environment that can be used to run and further develop EATOP. The environment is a plugin development environment for Eclipse. The instructions provided in this chapter are not the only way to get a working development environment, and the applicability of this information may change in the future as newer versions of third-party software get released.

6.2 Install Java

EATOP requires a Java installation. While a Java Runtime Environment is sufficient to run EATOP, a JDK (Java Development Kit) is recommended for development. The latest version of Java at the time of this Deliverable's release is Java 7 Update 51. EATOP is also known to work with the final version of Java 6, which is Java 6 Update 45.

The latest version of Java for various operating systems can be downloaded from:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

EATOP is also known to work with the alternative OpenJDK implementation 1.6.0_27 or later.

On Windows, Java may be installed by launching the executable downloaded from the Oracle website. On Linux, Java may additionally be available as a package from your distribution's repositories.

6.3 Install Eclipse

Eclipse is required as the IDE (Integrated Development Environment) and platform on top of which EATOP would run. Eclipse may be downloaded from:

http://www.eclipse.org/downloads/

Versions 4.3 (Kepler) and 4.2 (Juno) are known to work with EATOP. Eclipse comes packaged in archive files that should be extracted, and can be run by launching the Eclipse executable from the extracted location. For the last MAENAD version of EATOP, 4.3 Kepler is recommended.

Further information on downloading and installing Eclipse can be found in the Eclipse FAQ and in the Eclipse Readme. A general introduction to Eclipse as a Java IDE can be found in this article.

Note: Under Windows, the path to Eclipse should preferably be short, such as: C:\Work\Eclipse

6.4 Obtain the source code

The source code for EATOP can be acquired from the EATOP repository. The repository is located at https://code.google.com/a/eclipselabs.org/p/eclipse-auto-iwg/ The best way to get a copy is to use Git to clone the repository. This can be accomplished by running the following command:

git clone https://code.google.com/a/eclipselabs.org/p/eclipse-auto-iwg/

An alternative to using Git from the command line is to use the EGit plugin which comes with Eclipse to access the Git repository from within Eclipse.

The source code should then be imported into an Eclipse workspace, by choosing **File->Import** from the menu. If you checked out the source using Git, you can use the option to import from Git, which will be provided in the Import wizard.

Not all projects that are in the repository are used in EATOP bundles. Certain projects may be outdated or unnecessary. The list of projects currently being used can be obtained from the definition in the file EATOP_Trunk/eel/releng/org.eatop.releng.builds/pom.xml.

6.5 Set up the target platform

It is important to set up the correct target platform, otherwise you will not be able to run or develop for EATOP. The target platform is part of the source code repository, and exists as an Eclipse project called **org.eatop.targetdefs**. The project consists of two files specifying different targets, called **4.2.target** and **4.3.target.** These correspond, respectively, to target platforms for versions 4.2 and 4.3 of Eclipse. Opening either file within Eclipse will allow you to set it as the current target platform by clicking a linker in the upper right corner.

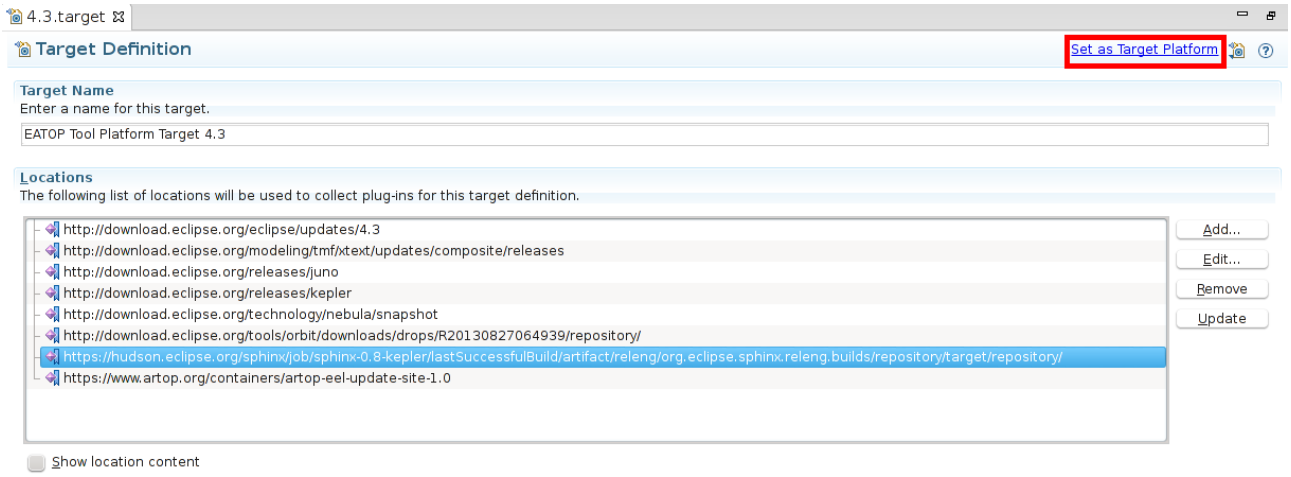

Figure 4 selecting the target platform

Version 4.3 is newer and therefore recommended, but version 4.2 is also functional.

6.6 Create or verify a run configuration

To run EATOP, you need to have the correct Run Configuration within Eclipse. Choose **Run->Run Configurations** from the Eclipse menu, and there should already be a configuration called Eatop Technology Demonstrator. If there is not, then create a new configuration under the section Eclipse Application, and configure it to run the product **org.eatop.eel.examples.demonstrator.product**.

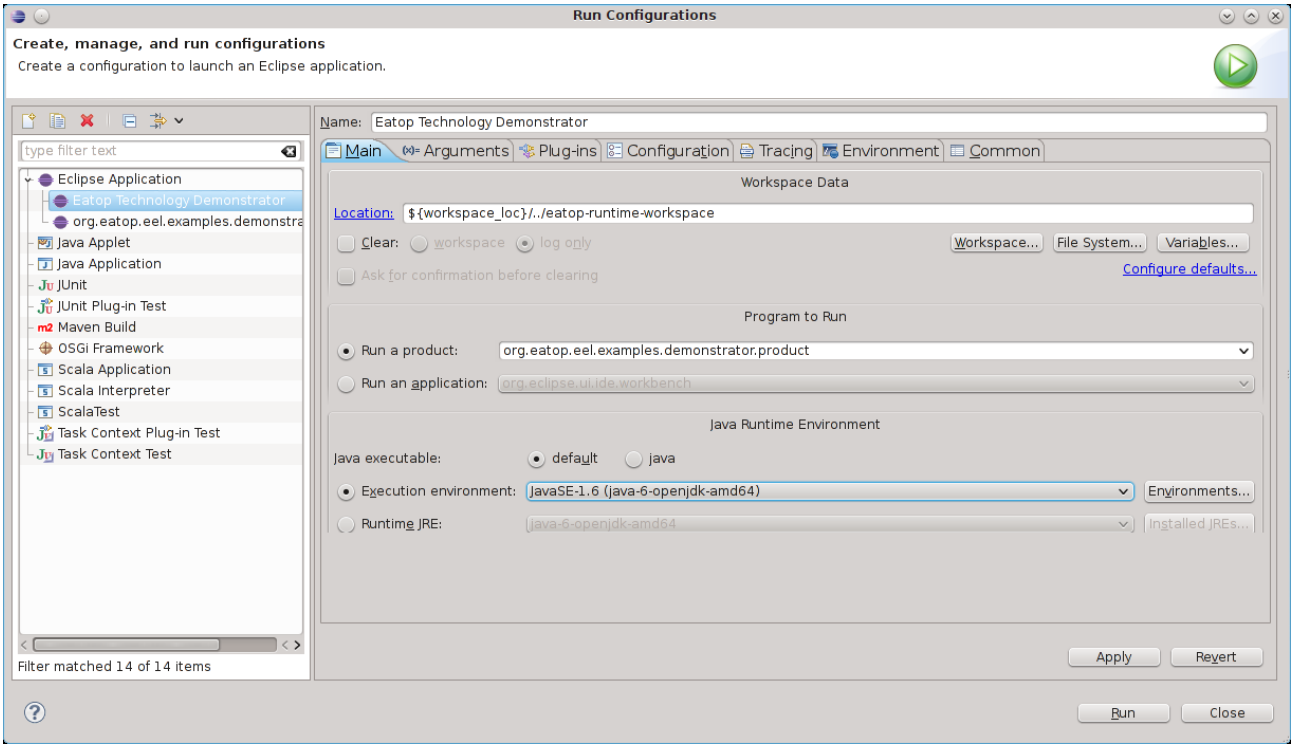

Figure 5: Eclipse Run Configuration

Further, you should make sure that correct arguments are provided to the Java Virtual Machine for the run configuration. These are available in the *VM Arguments* section of the *Arguments* tab.

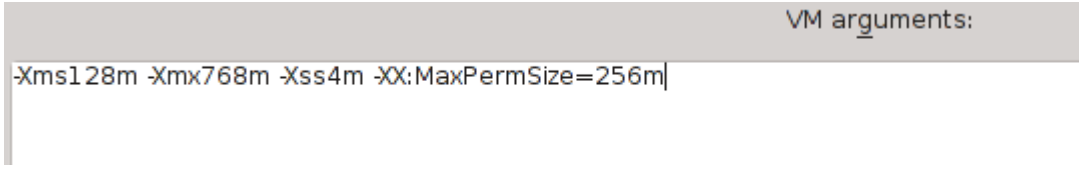

Figure 6: Java Virtual Machine arguments

These arguments allocate sufficient amounts of memory to the Java Virtual Machine. If you experience frequent crashes in the demonstrator, of if the demonstrator crashes when trying to open or create a project, then the most likely cause is that the necessary VM arguments are missing and thus the demonstrator is not receiving enough memory.

6.7 Run the demonstrator

If, at this point, you are seeing compile errors in Eclipse but you have correctly set the target platform, it likely means that you have imported some projects from the repository that are outdated and not required for EATOP. You may close such projects altogether, or disable them from the run configurations Plug-ins tab.

Once you have completed the previous steps, you should be able to run the EATOP demonstrator from within Eclipse. To do it, just run the run configuration that is already present or that you created in the previous section. If everything has been done correctly, you'll see the EATOP demonstrator logo as the application begins to load.

Figure 7: EATOP demonstrator starting up

7 Using the EATOP demonstrator

7.1 Introduction

Overall, the EATOP demonstrator should be familiar to those who have previously used Eclipse, and especially to those who have used Eclipse-based products from the Artop project, such as the Artop demonstrator or the Artext demonstrator.

Please note that this Deliverable focuses on the functionality which has been introduced to EATOP as part of the MAENAD project, and does not provide information about the metamodel generation from Enterprise Architect or other such non-MAENAD features.

In its default view, the EATOP demonstrator looks similar to the figure below.

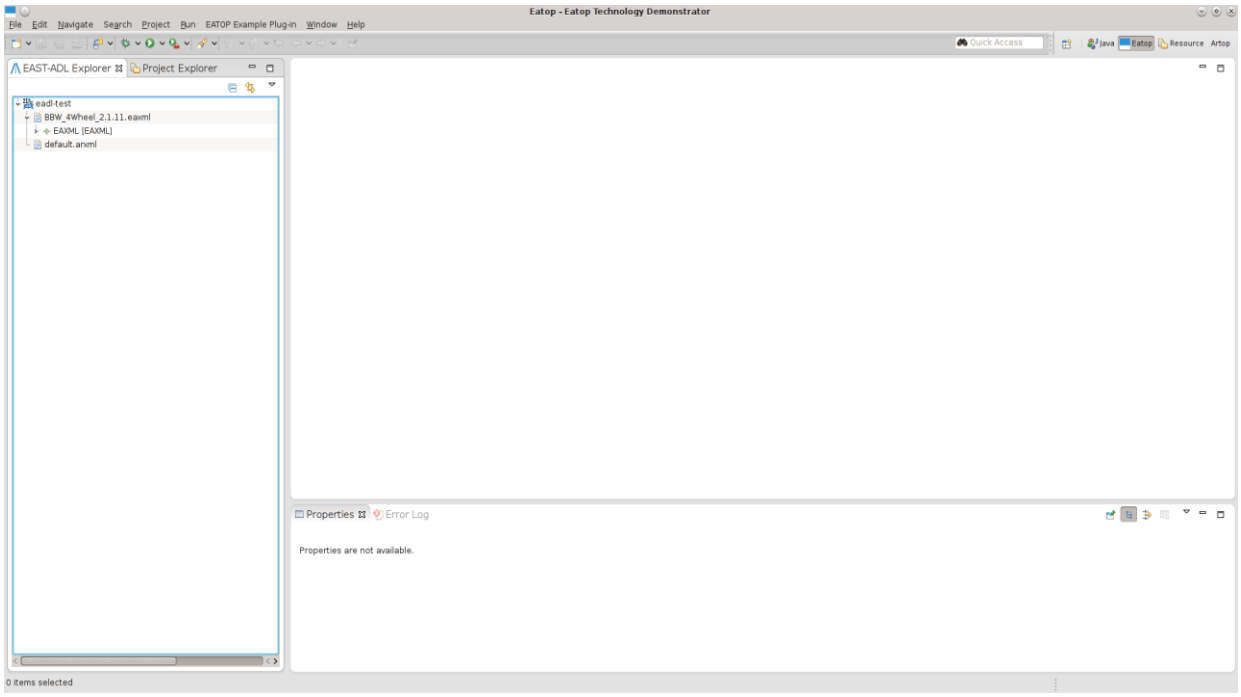

Figure 8: EATOP default view

The most important EATOP Demonstrator component is the EATOP Explorer, located in the left part of the screen. It shows any EADL models that are loaded in the workspace, and provides an interface for modifying them. The context menu that can be accessed by right-clicking in the Explorer also provides options to import existing projects, create new ones, and others.

7.2 Creating an EAST-ADL model

This section shows the simplest procedure to create an EAST-ADL model and add some elements to it.

To create a model, first you need to create an EAST-ADL project. In the **File** menu of the demonstrator, or the context menu in the Explorer, choose **New** → **EAST-ADL Project**. You will be presented with a window asking you to name your project and select a location for it. By default, the project is automatically set to use the latest EAST-ADL release.

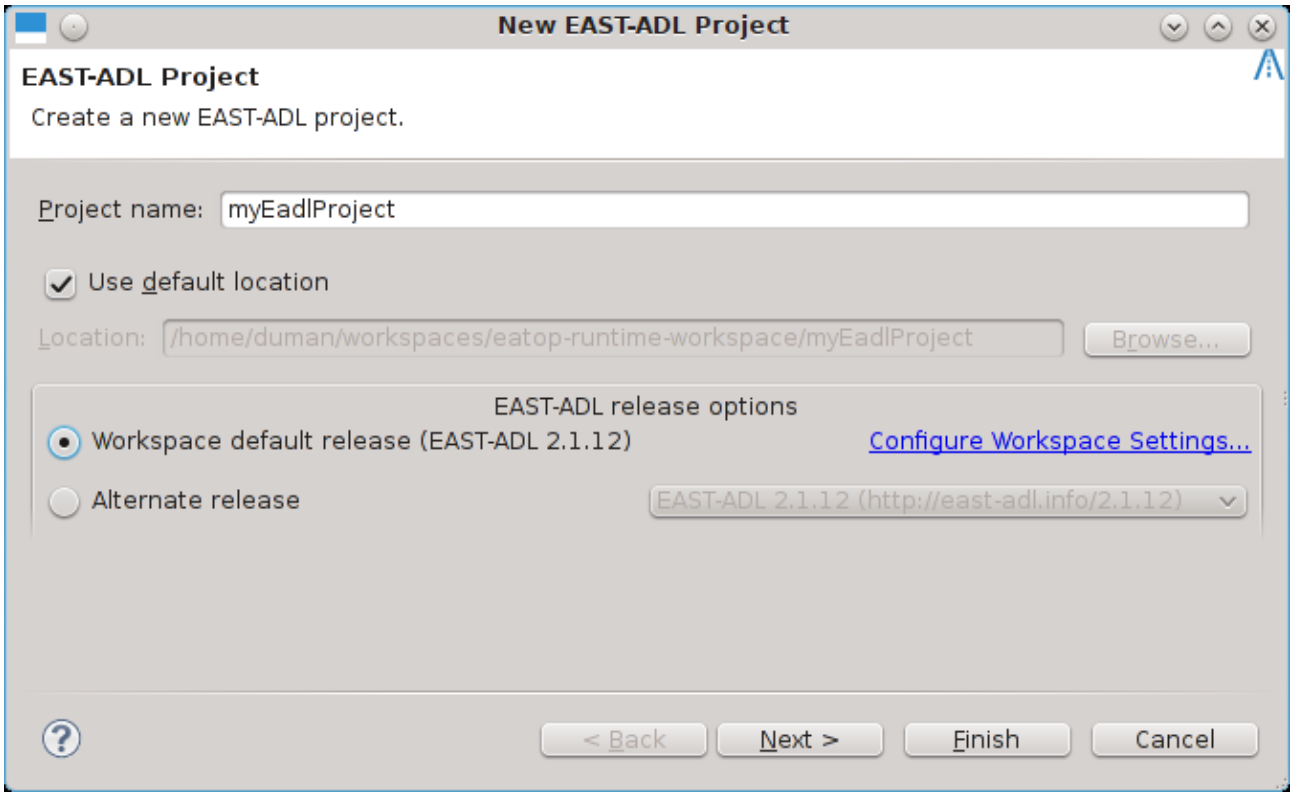

Figure 9 Creating a new project

The newly created project will show up in the EATOP Explorer. Right-click it and choose **New** → **EAST-ADL File** to create an eaxml file that will store your model.

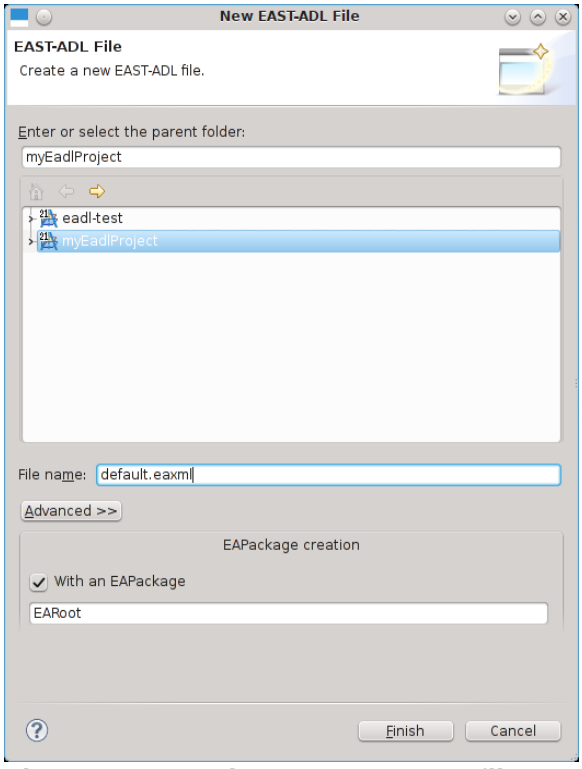

Figure 10: Creating an EAST-ADL file

Note that, when you create a new EAST-ADL file, you're also given the option to immediately create a top-level EAPackage in the new file. When you have created the file, you can expand it in the Explorer and you will see that it consists of an EAXML element and an EAPackage (unless you chose not to create a package when creating the file).

Now you can create your model by adding elements to the packages. Just right-click an element in the Explorer to add a child to it. For example, you can right-click a package and add a HardwareComponentType to it, and then right-click the HardwareComponentType to add an IOHardwarePin to it. You can select the child to be added from a metamodel-based category, or just select among all eligible elements.

Figure 11: Created elements

To set the names of model elements, and to edit their other properties, you should use the Properties view that is located horizontally at the bottom part of the EATOP demonstrator's screen.

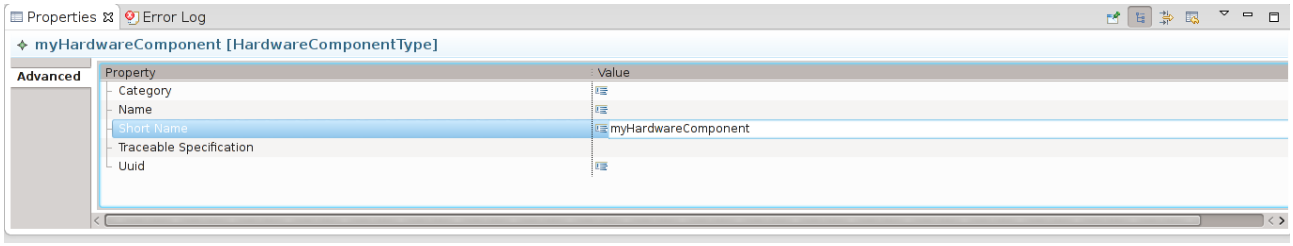

Figure 12: Properties section when editing an element's name

7.3 Type-Prototype browsing in the Explorer

One of the features that the EATOP Explorer adds for extra user convenience is the display of a virtual containment tree in the Explorer that reflects Type-Prototype relationships which are common in the model.

Suppose you have a HardwareComponentType called *hwCompType1,* and you have a HardwareComponentPrototype called *prototypeOfType1*, and the type of *prototypeOfType1* is *hwCompType1*. If such a relationship exists, then any children of *hwCompType1* will also be shown if you expand *prototypeOfType1* in the Explorer because the prototype has access to them through its type. This is shown in the figure below.

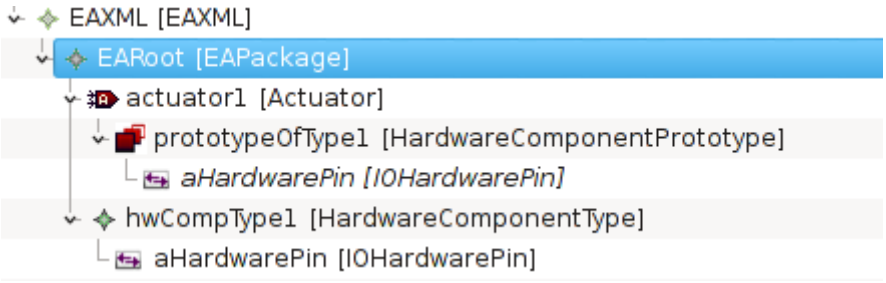

Figure 13 A prototype's virtual children

In this case, *aHardwarePin* is an IOHardwarePin that exists in the *hwCompType1* type, but it is also shown as a child of the *prototypeOfType1* element. Also note that the *prototypeOfType1*

element is shown in italics, indicating that it's a prototype with a set type and therefore shows virtual child elements that belong to the type.

To make it easier to recognize elements, many element types have their own icons, as you can see above for the Actuator, HardwareComponentPrototype and IOHardwarePin types.

Of course, viewing the prototype's properties in the Properties view also reveals the type that it belongs to.

Figure 14: A prototype with a set Type property

Also, if you right-click the prototype in the Explorer, the context menu will show a *Goto type* command which will then take you to the prototype's type in the Explorer.

7.4 Editing instance references

Many elements of the EAST-ADL meta-model include instance references. The EATOP demonstrator provides a more convenient interface for editing instance references, as well as intelligent functionality for automatically completing the context of an instance reference.

Figure 15: An EventFunction element

As an example, let's use an *EventFunction* element, shown above, which contains an instance reference to a *DesignFunctionPrototype.* When you select an element with an instance reference in the EATOP Explorer, then the properties section, which usually looks like Figure 12, gets two extra tabs. One is *Edit Instance Refs,* and the other is *Advanced Instance Refs*, shown below.

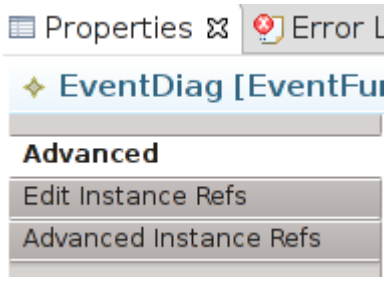

Figure 16: Extra tabs in the property section

You do not have to add an instance reference manually to the parent element. If the element can have instance references but does not have any, an instance reference will be created automatically when you try to edit it. For example, if you add a new *EventFunction* to your model, you can proceed straight to instance reference editing.

The simplest way to edit an instance reference is from the *Edit Instance Refs* tab, where you only need to select the target for the instance reference. As shown in the figure below, you have access to the instance reference's target property. Note that, if you click in the Value field, then instead of the usual Eclipse-based list of items, you get a button that calls the instance reference editor.

Figure 17: Editing of an instance reference

The editing of an instance reference target occurs through a tree view like that which is shown below. The tree in the instance reference editing dialog acts the same way as the Explorer tree, that is, it will show virtual children of prototypes. Thus you can navigate directly to your desired target by going through the prototypes.

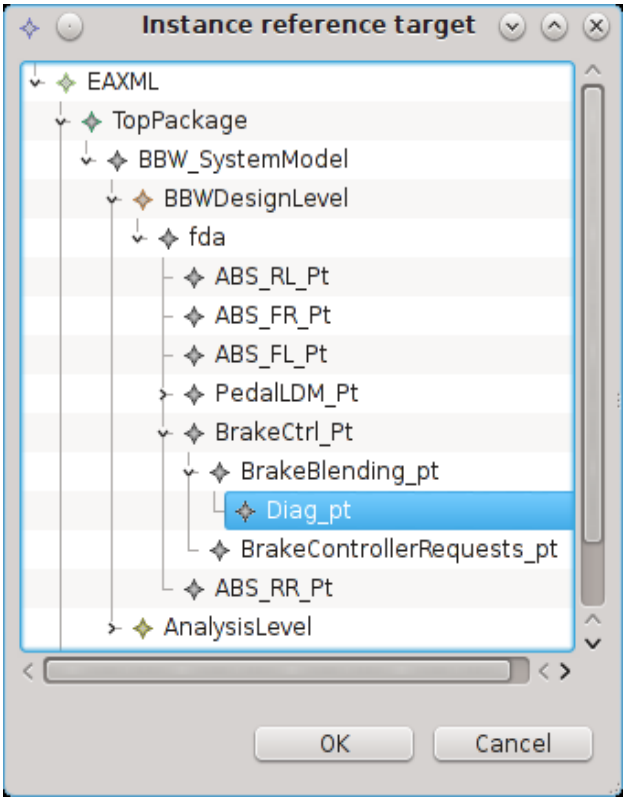

Figure 18: Selecting a target for an instance reference

After you have selected the target in the dialog window shown above, the instance reference's context will be automatically filled, and the instance reference tab will now show the full path to the target through its context.

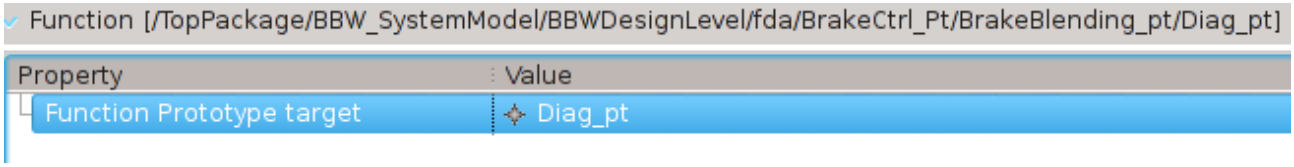

Figure 19: Target being shown along with full path

7.5 Advanced instance reference editing

Sometimes you may wish to manually set the context of an instance reference, or to disable the behaviour of automatically completing the context after setting the target. This may especially be desirable when working with incomplete or inconsistent models.

From the *Advanced Instance Refs* tab in the property section, you have explicit access to the instance reference's context which you can then edit using a dialog just like the one shown in Figure 18. Also, when you edit a target from the *Advanced Instance Refs* tab, you will be able to control whether the EATOP demonstrator attempts to automatically set the context. The editor window gets a checkbox controlling this behaviour. If you deselect it, then context will not be changed.

Finally, if you wish, you can select the instance reference itself (which is a separate EMF element) in the Explorer tree, as shown below. If you select the raw reference that way, then the property section will be the default one from Eclipse, and editing the instance reference from it will not show the virtual path-through-context or have any other EATOP-specific assistance.

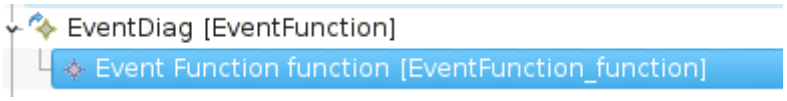

Figure 20: The raw instance reference selected

7.6 Navigation help

EATOP provides certain features that make model navigation faster in the **EAST-ADL Explorer**. When you right-click an element, additional submenus will be shown if there are other elements that refer to this one. You can then see a list of such referring elements and jump to any of them in the Explorer.

For example, below you can see that the element *MyTimeQuantity* is referred to by the *MyTimeUnit* element in the *DataTypesPackage* package. Clicking on *MyTimeUnit* in the popup menu would show it in the Explorer.

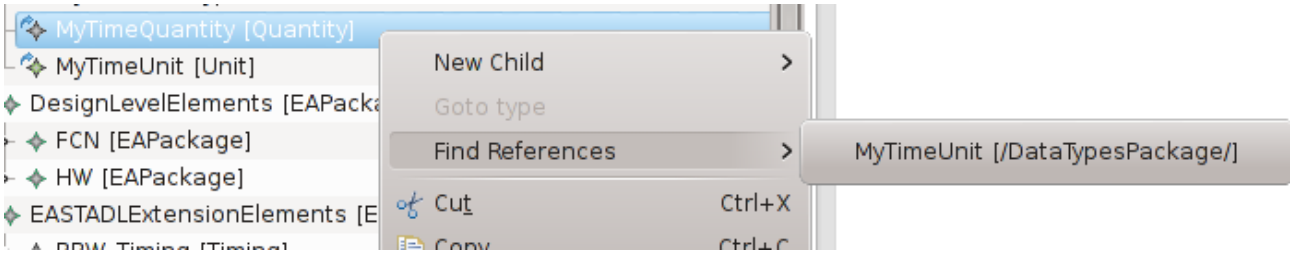

Figure 21: Navigation help for references

8 Summary

The EATOP demonstrator is a complete Eclipse-based product that can be used to create and edit EAST-ADL models. The demonstrator is technologically based on Eclipse and Sphinx, and its functionality closely resembles that of the Artop demonstrator which can be used with AUTOSAR models. EATOP is an official Eclipse Project within the Modeling category.

In addition to the usual model editing functionality provided by Sphinx, the EATOP demonstrator includes a number of enhancements specifically for ease of use with EAST-ADL models. These enhancements include features for working with type-prototype relationships and instance references in the model, as well as features for ease of navigation.

9 References

SAFE project - http://www.safe-project.eu/

EATOP Wiki pages - https://code.google.com/a/eclipselabs.org/p/eclipse-auto-iwg/w/list

EATOP repositories - https://code.google.com/a/eclipselabs.org/p/eclipse-auto-iwg/source/browse/

ARTOP project - https://www.artop.org/

EAST-ADL Association - http://www.east-adl.info/