The MOLA Tool

User Guide

Version 2.2

November, 2009

Table of Content

1 Introduction....4

2 Installation....5

2.1 Installation of Repository Browser....5

2.2 Details for C++ version....5

3 Quick Start....6

3.1 How to Create MOLA Transformation....6

3.2 How to Create a Class Diagram....8

3.3 How to Create a MOLA Procedure....10

3.4 How to Compile MOLA Transformation....11

3.5 How to Execute MOLA transformation....12

3.5.1 Preparing data....12

3.5.2 Transforming data....13

4 MOLA TDE....15

4.1 Metamodel Editor....16

4.1.1 Creating/Deleting Packages....16

4.1.2 Creating/Deleting Class Diagrams....16

4.1.3 Creating/Deleting Class Diagram Elements....16

4.1.4 Importing Ecore Metamodels....17

4.2 MOLA Editor....17

4.2.1 Creating/Deleting MOLA Packages (Units)....17

4.2.2 Creating/Deleting MOLA Procedures....17

4.2.3 Editing MOLA Procedures....17

4.2.4. Importing/Exporting MOLA Procedures....18

4.3 MOLA Compiler....18

4.3.1 Using MOLA to mii_rep (C++) Compiler....18

4.3.2 Using MOLA to JGralab Compiler....18

4.3.3 Using MOLA to EMF Compiler....18

4.3.4 Part of MOLA Language Recognized by Compiler....18
5 MOLA TEE....20
5.1 Transformation Runner....20
5.2 Repository Browser....20
Appendix A – Error Messages of the MOLA Compiler....21

1 Introduction

This paper contains the main guidelines for the usage of MOLA Tool. It includes the installation manual, the user manual and issues on the execution of the MOLA transformations.

The MOLA Tool is used to create, compile, run and store MOLA transformations. The MOLA Tool is designed as a **freeware** tool for supporting research in the MDA and MDSD areas. It consists of two parts - the **Transformation Development Environment (TDE)** and the **Transformation Execution Environment (TEE)**.

The main components of TDE are:

- **Graphical Editors** for metamodel and MOLA procedures built using the METAclipse tool building framework. The editors are syntax directed – they allow creating mostly syntactic correct constructions and offering sophisticated prompters. The integrity between metamodel and MOLA procedures are maintained automatically.
- **MOLA Compiler** (a set of compilers) which checks the syntax correctness of MOLA program and creates the executable. Compiler generates OOPL code from a MOLA transformation, which after the compilation is capable of executing against an appropriate model repository. Three different target repositories are available. Compiler generates C++ code against the API of high performance custom model repository **mii_rep** built by UL, IMCS. Two other compiler versions generate Java code, one against the open source high performance graph/metamodel based repository **JGraLab** built at the University of Koblenz. However, the most important for wide usage of MOLA is the compiler version generating Java against the API of **Eclipse EMF**, which is the most popular model repository kind so far. The result of compilation is the executable file (jar or dll). The definition of corresponding repository is also created by MOLA compiler.

Execution of MOLA transformations are performed using MOLA TEE. It consists of the metamodel-based in-memory repository and the appropriately compiled transformation executable. Repository must contain the respective metamodel (which also can be built by the MOLA TDE). MOLA TEE offers several possibilities how to run transformations:

- **Transformation Runner** is built as an Eclipse plug-in. It allows executing MOLA transformation on an **EMF model**.
- **Repository Browser** is built as Java application which allows browsing the content of repository in tabular view and executing transformations. It is compatible with all supported repositories for browsing, but only mii_rep (C++) transformations can be run.
- Transformation can be integrated directly in your software. If your software uses one of mii_rep, JGralab or EMF repositories, then transformations can be executed directly on models loaded in the memory. MOLA transformations are suitable for integration into a modelling tools as plug-ins, e.g. as Rational Software Architect plug-in (it uses EMF) or Enterprise Architect plug-in (import/export facilities needed from EA repository).

2 Installation

This chapter contains the installation guide of MOLA Tool. The MOLA Tool works on Microsoft Windows XP and VISTA platforms. The MOLA Tool can be downloaded from the MOLA web site http://mola.mii.lu.lv/TD/MOLA2Tool.zip . Simply unzip the archive file.
The folder *MOLA2Tool_vx.y* contains *METAclipse.exe* executable file which must be launched to start the MOLA TDE. Since MOLA TDE itself is an Eclipse plug-in, it includes also part of MOLA TEE – Transformation runner.

2.1 Installation of Repository Browser

Repository browser allows you browsing supported repositories. You can download the Repository Browser from http://mola.mii.lu.lv/TD/repBorwseris-0.4.0.zip . Unzip the archive. There is a file *start.bat* which must be launched to start the Repository Browser. Java is required for this tool.

2.2 Details for C++ version

Currently the Borland Turbo C++ Explorer is required to finish a compilation process of MOLA transformation for mii_rep. If you do not plan using mii_rep, then you can omit this step. More information about it is found in http://www.turboexplorer.com/cpp and http://www.codegear.com/downloads/free/turbo . Note that a registration is required to obtain the product activation key.
Download the full prerequisite install (228 MB zipped) from ftp://ftpd.codegear.com/download/bds/bds_2006_trial/english/arch/disk2/prereqs.zip . The prerequisite installer will install the software necessary for your Turbo product to run, including Microsoft .NET Framework v1.1 Redistributable, Microsoft .NET SDK v1.1, Microsoft Visual J# v1.1 Redistributable, Microsoft Internet Explorer 6 SP1 and Microsoft XML Core Services v4.0 SP2. If you already have any of these items installed, the installer will skip them.
Download Turbo C++ Explorer installer (390 MB zipped) from http://altd.codegear.com/akdlm/download/turbo/English/C++_Explorer/TCPP_EXPL_EN_DL.exe . Run it and follow the instructions.

3 Quick Start

This chapter contains a short list of the main actions which are performed by a user of MOLA Tool – starting MOLA project, creating MOLA transformation, creating class diagram, creating MOLA procedure and compiling MOLA transformation. Since it is "a quick start" this chapter does not contain detailed description of Metamodel Editor (see Chap. 4.1 for details) and MOLA Editor (see Chap. 4.2 for details). We suppose that these editors are quite intuitive.

3.1 How to Create MOLA Transformation

1) Launch *METAclipse.exe*
2) Choose a workspace folder, where MOLA projects are stored (see Fig.1).

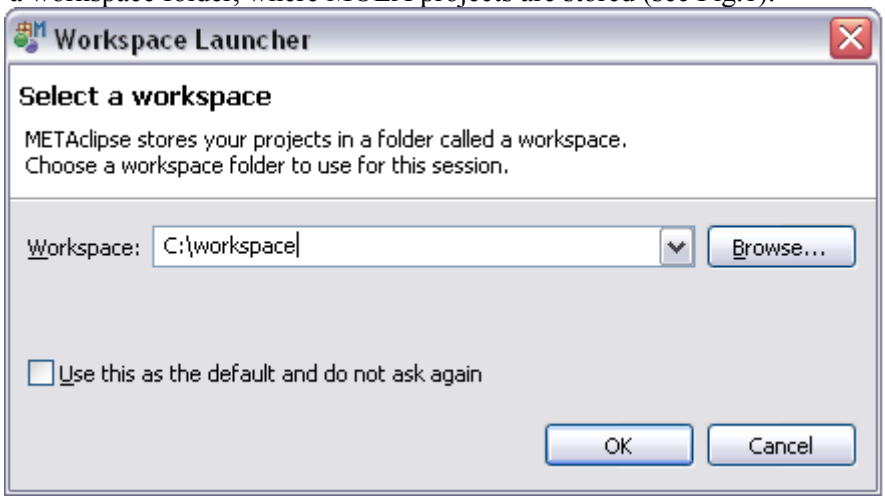

Figure 1 Selection of workspace

3) Right-click on the METAclipse Explorer, choose "New->Project..." (see Fig. 2).

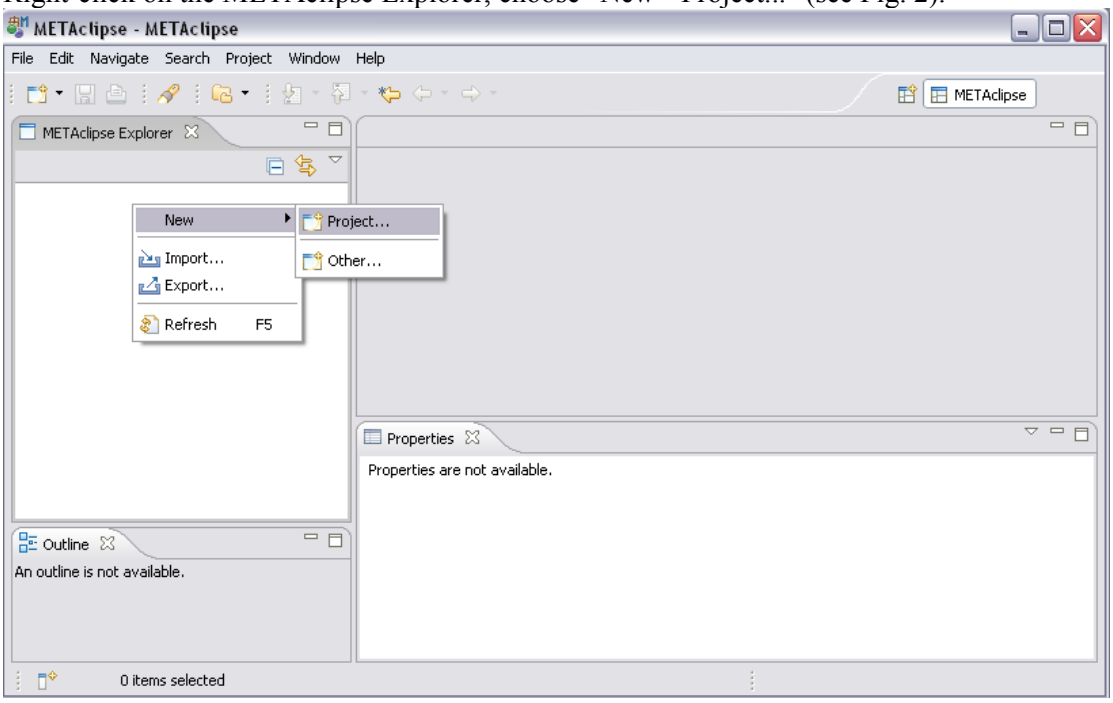

Figure 2 Creation of MOLA project, step 1.

4) Choose "METAclipse->METAClipse Project" in the tree, press "Next >" (see Fig. 3)

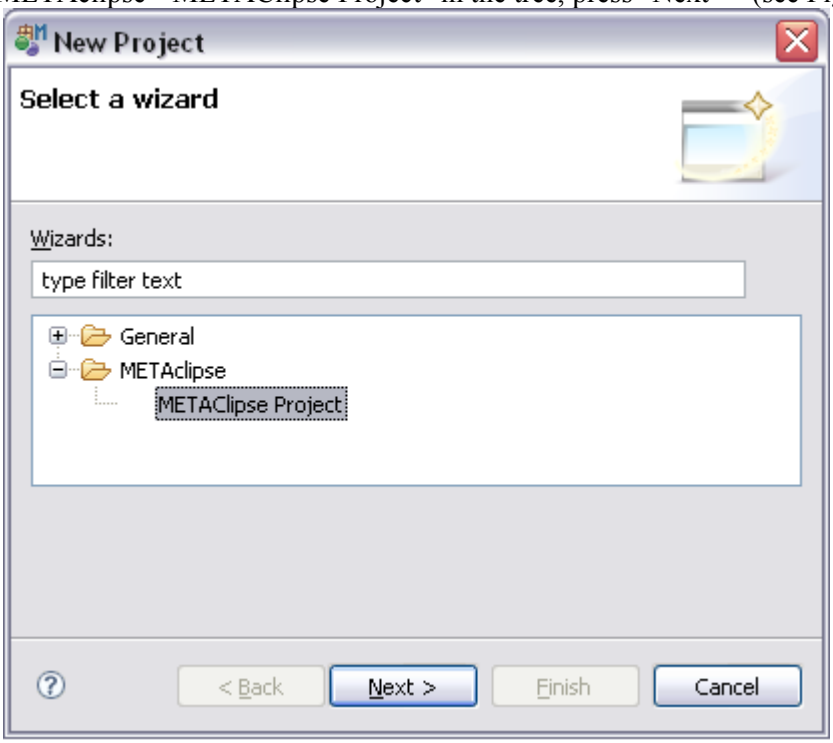

Figure 3 Creation of MOLA project, step 2

5) Name the MOLA project and press "Finish". (see Fig. 4)

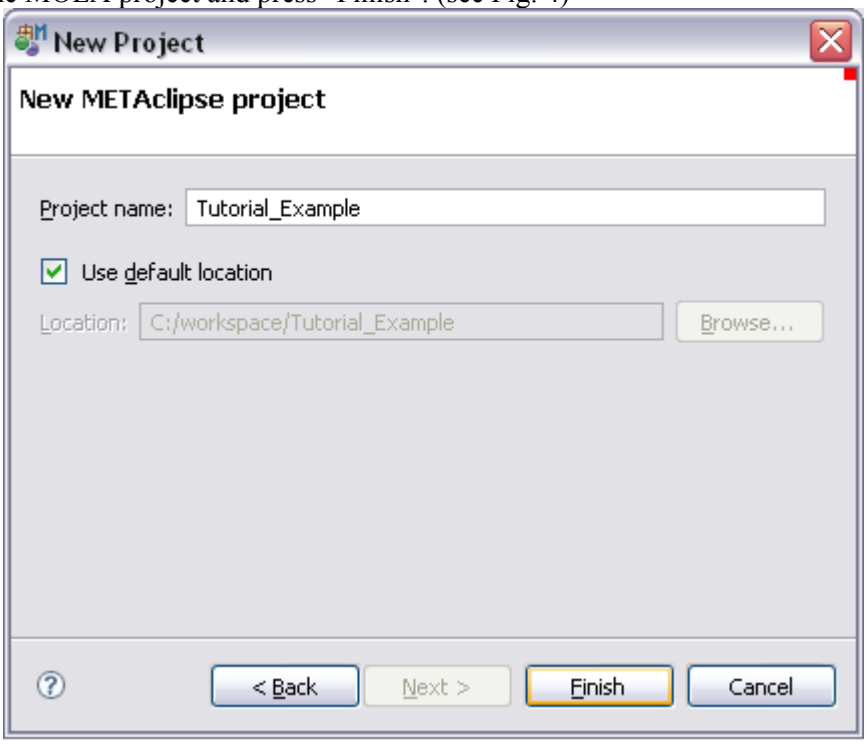

Figure 4 Creation of MOLA project, step 3

6) There may be several projects in the workspace. To create another METAclipse project, close the opened one – right-click on the project node and choose “Close Project”(see Fig. 5) and follow this instruction (steps 3-5)

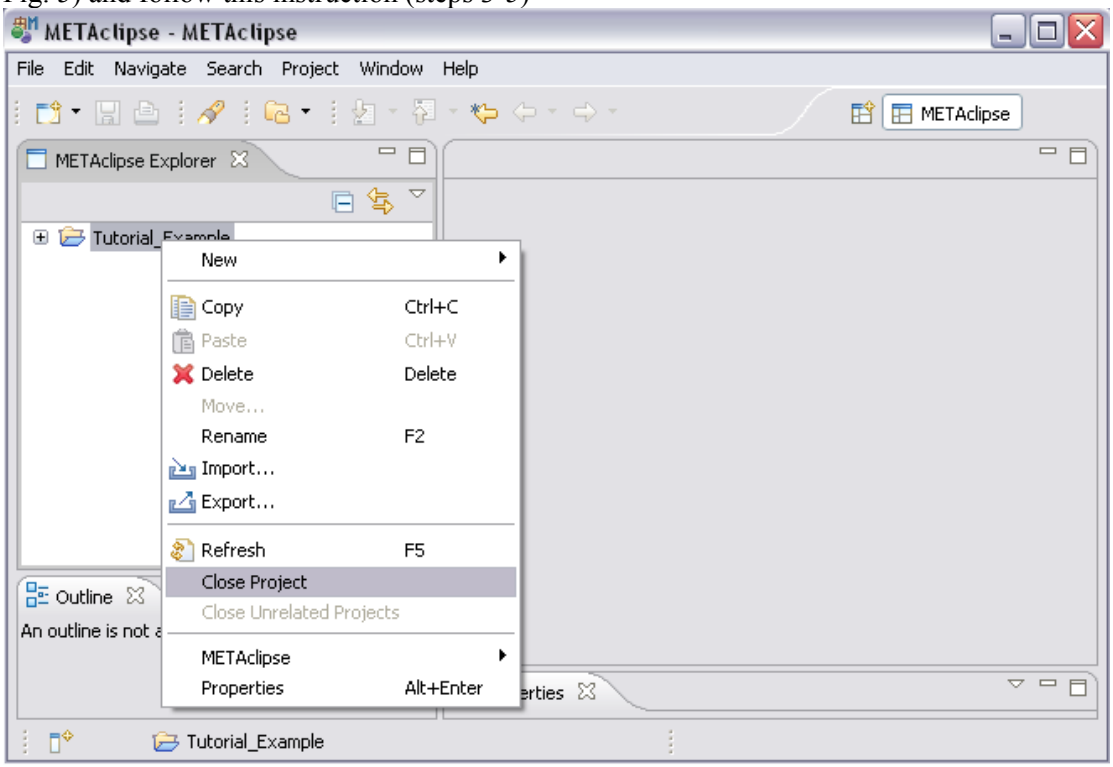

Figure 5 How to close a MOLA project

3.2 How to Create a Class Diagram

1) Right-click on the metamodel node and choose “Add class diagram”. (see Fig. 6)

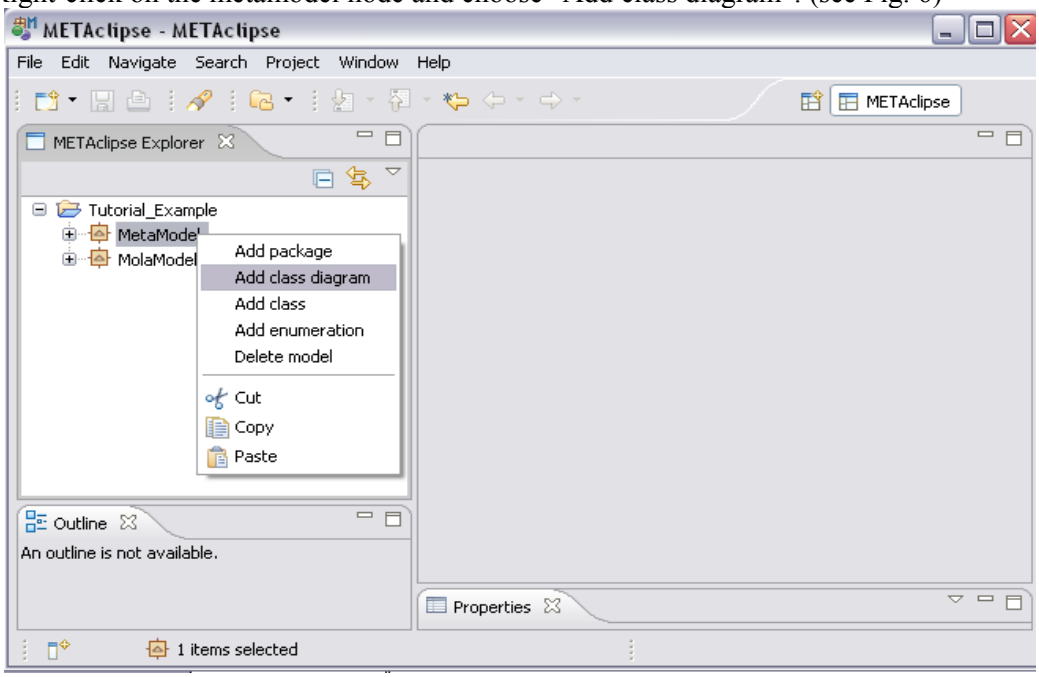

Figure 6 Creation of class diagram

2) Double-click on the class diagram node to open the metamodel editor (see Fig. 7). The metamodel may contain several class diagrams. The elements of the metamodel (classes, associations, etc. …) may be displayed in any class diagram. Note, if you plan using EMF, then all classes *should* be packaged. Every root package (included directly under *MetaModel* node) will become distinct *ecore* file.

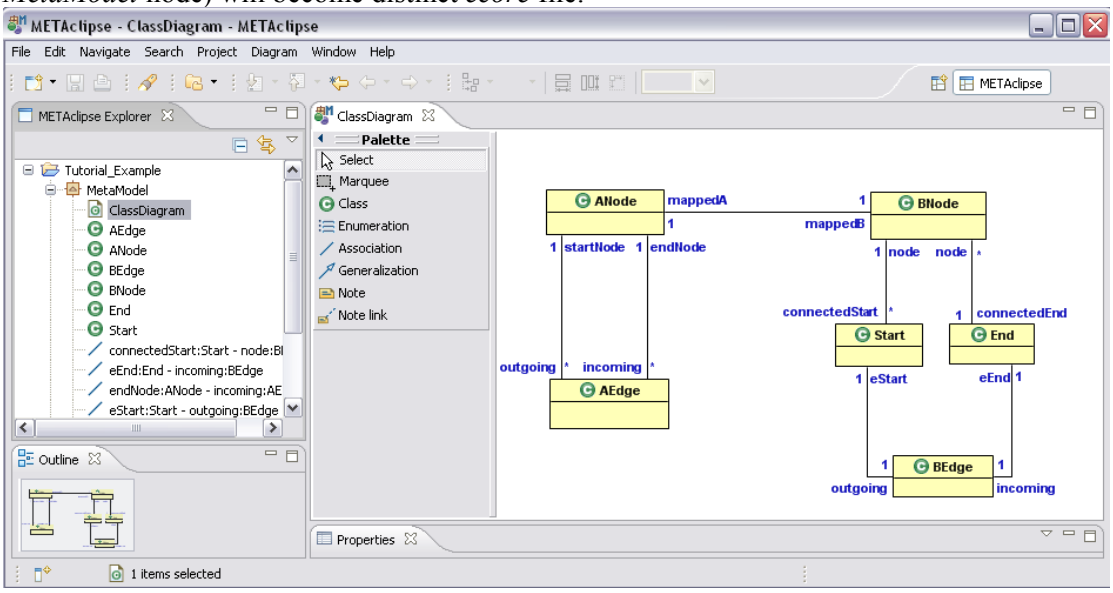

Figure 7 Editing of class diagram

3.3 How to Create a MOLA Procedure

1) Right-click on the MOLA model node and choose “Add procedure”. (see Fig. 8)

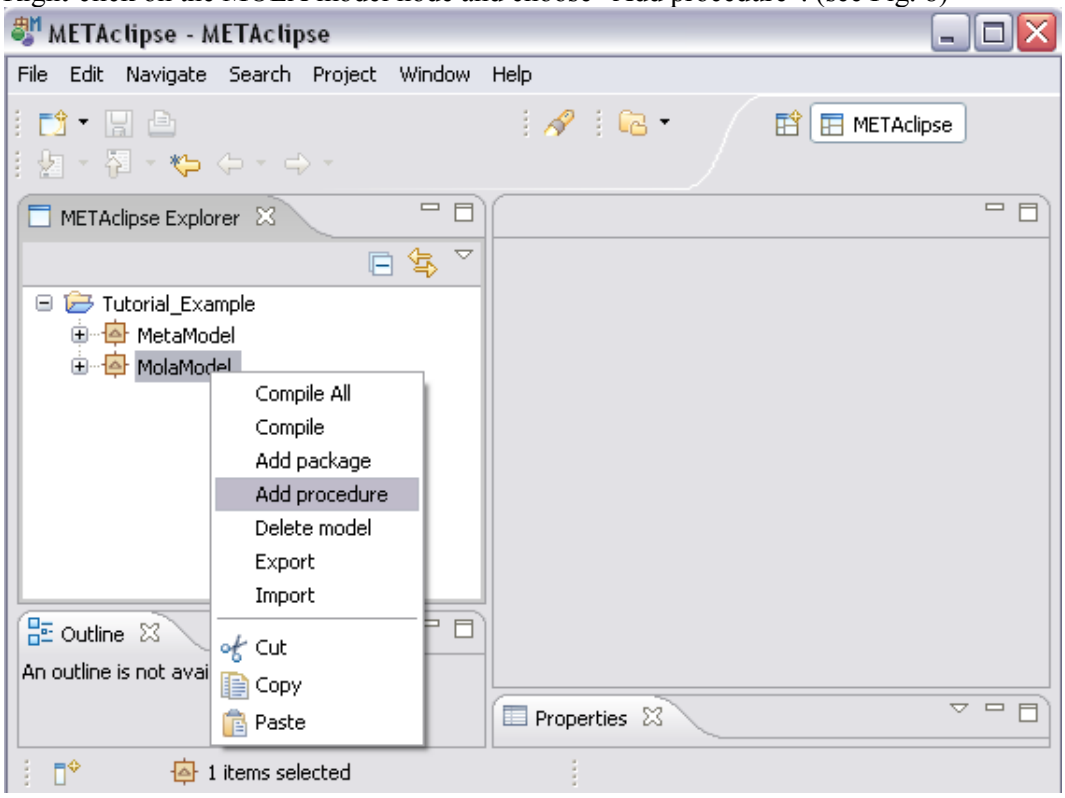

Figure 8 Creation of MOLA procedure

2) Double-click on the MOLA procedure node to open the MOLA editor (see Fig.9). Note, that one procedure in the transformation must be **the main**.

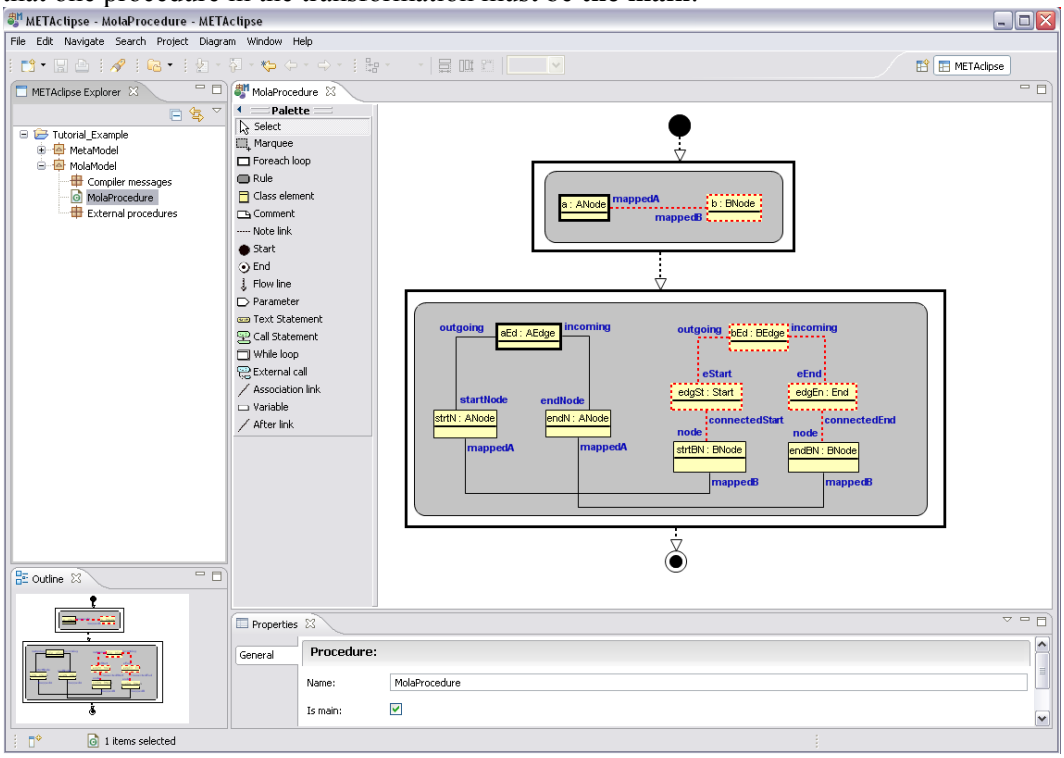

Figure 9 Editing of MOLA procedure

3.4 How to Compile MOLA Transformation

1) Right-click on the MOLA model node and choose “Compile All – Compile toEMF” (see Fig. 10). If syntax check is successful, then *ant* script is launched automatically.

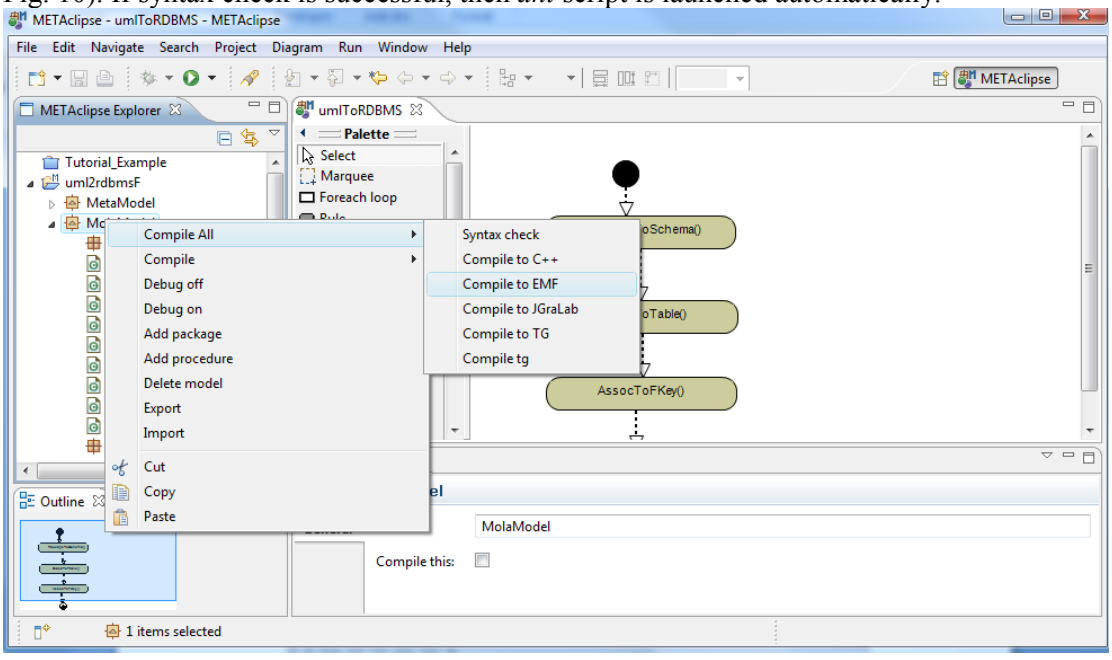

Figure 10 The compilation of a MOLA transformation

2) If compilation fails, choose the compiler messages node in the METAclipse Explorer (see Fig. 11). The list of errors is shown in the properties tab. Double-click on the list item

navigates to the appropriate MOLA procedure (if possible). The “problematic” elements are highlighted in the MOLA editor.

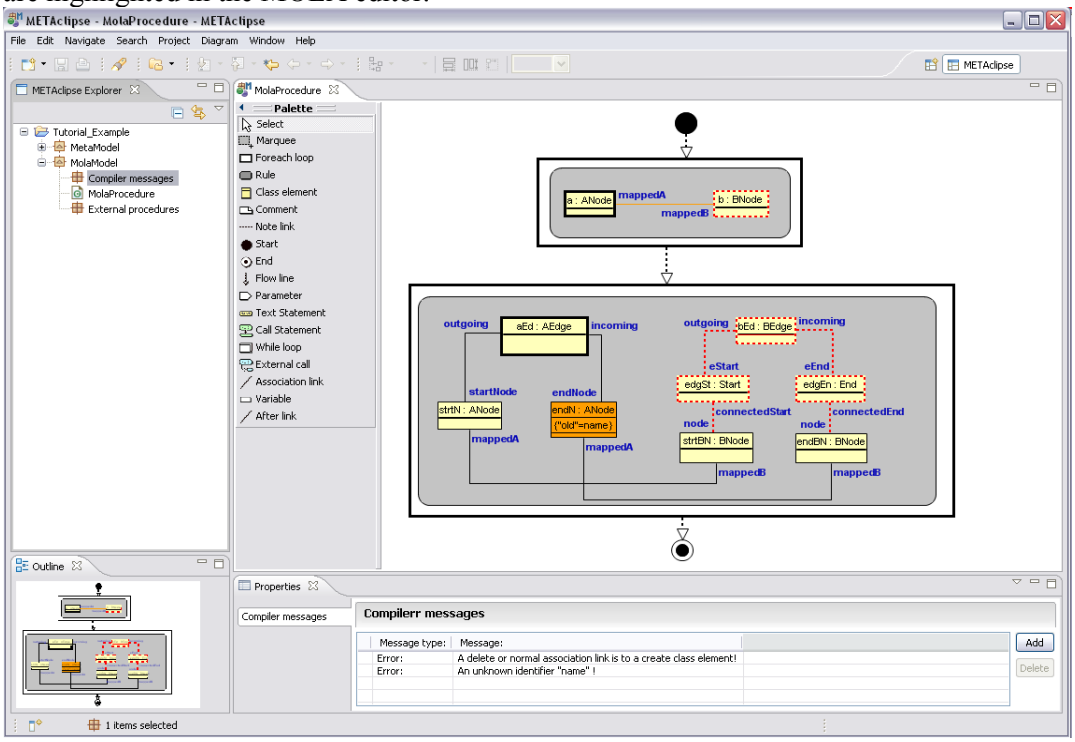

Figure 11 Error handling in MOLA Editor

3) The result of compilation result (*L0_CompRes.jar*, **.ecore* files) is stored in *result/MolaModel*. (see. Fig. 12)

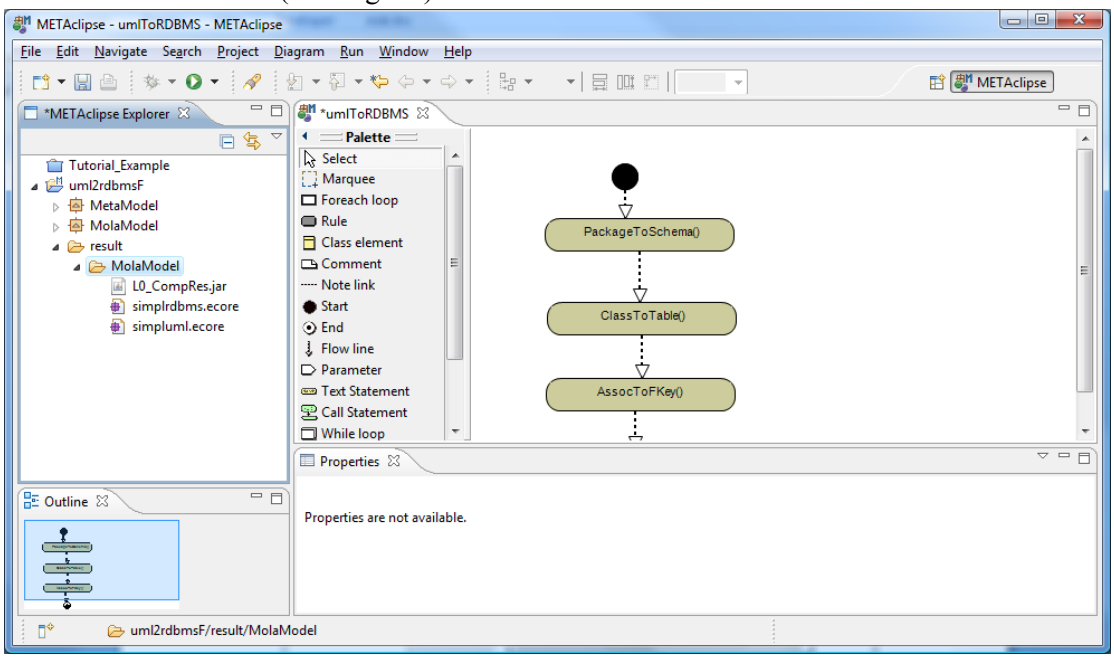

Figure 12 Compilation result

3.5 How to Execute MOLA transformation

This section contains a brief description, how to prepare data for transformation using means by Eclipse EMF and run transformation using Transformation runner.

3.5.1 Preparing data

1) Open *ecore* file by double-clicking on it. Right-click on the class, which is instantiated, and choose “Create dynamic instance ...” (see. Fig. 13). Choose the name of XMI file and press OK.

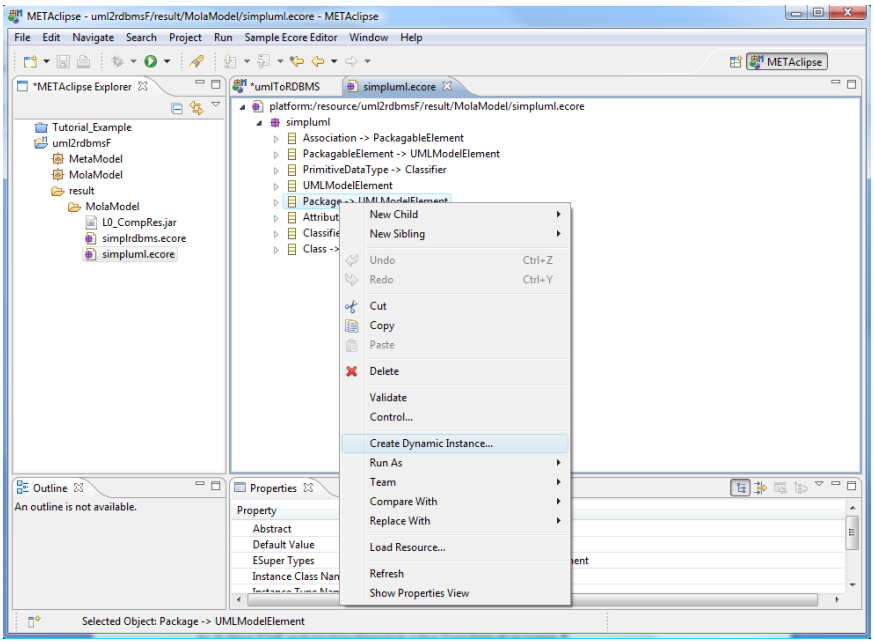

Figure 13 Dynamic instantiation of Ecore class

2) Use EMF Model Editor to create instances (see. Fig. 14).

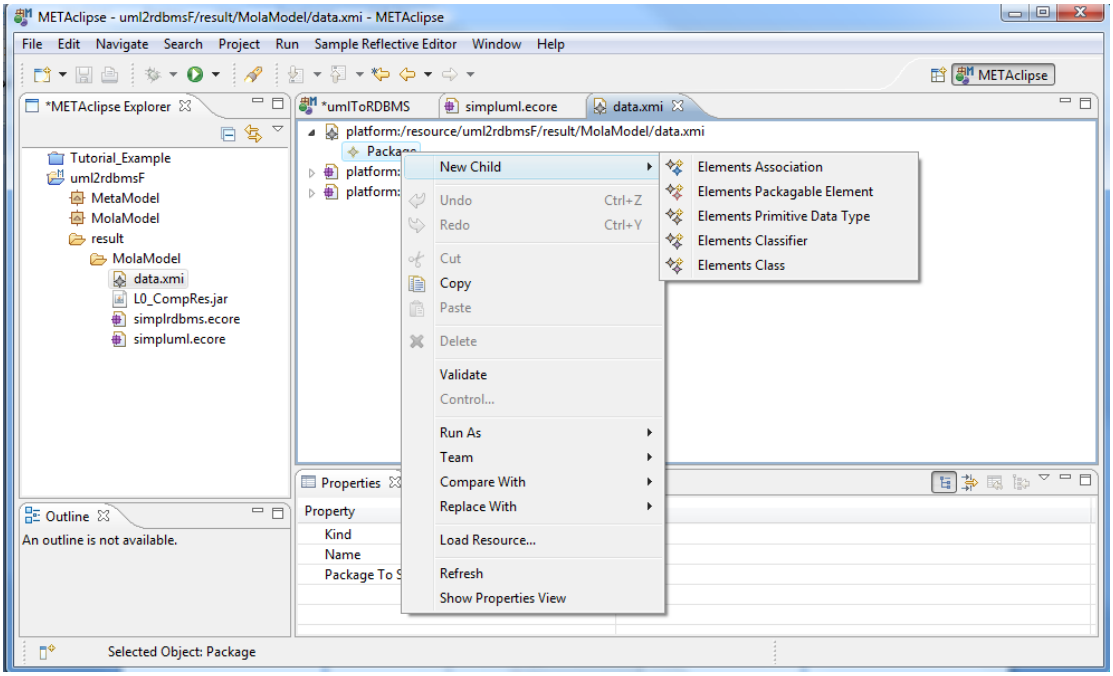

Figure 14 EMF Model Editor

3.5.2 Transforming data

1) Select “Run – Run Configurations ...” (see Fig. 15).

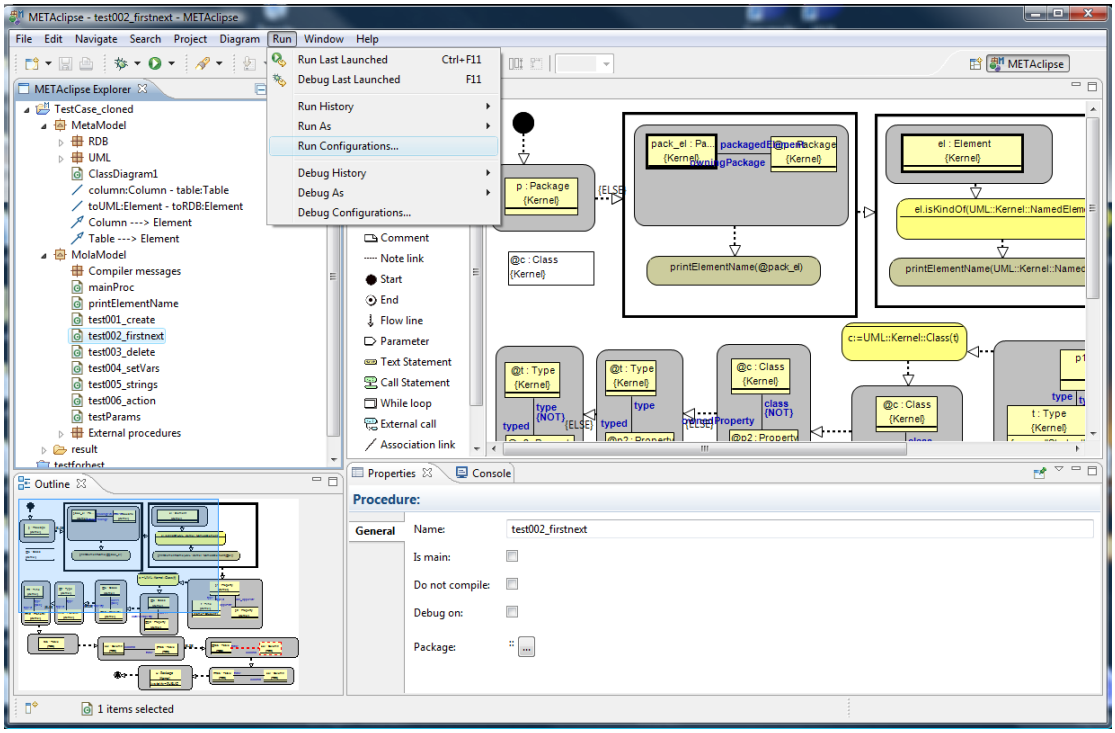

Figure 15 Opening Run configurations dialog

2) Create new launch configuration by selecting “Lx / MOLA Transformation” and pressing “New” button (see Fig. 16)

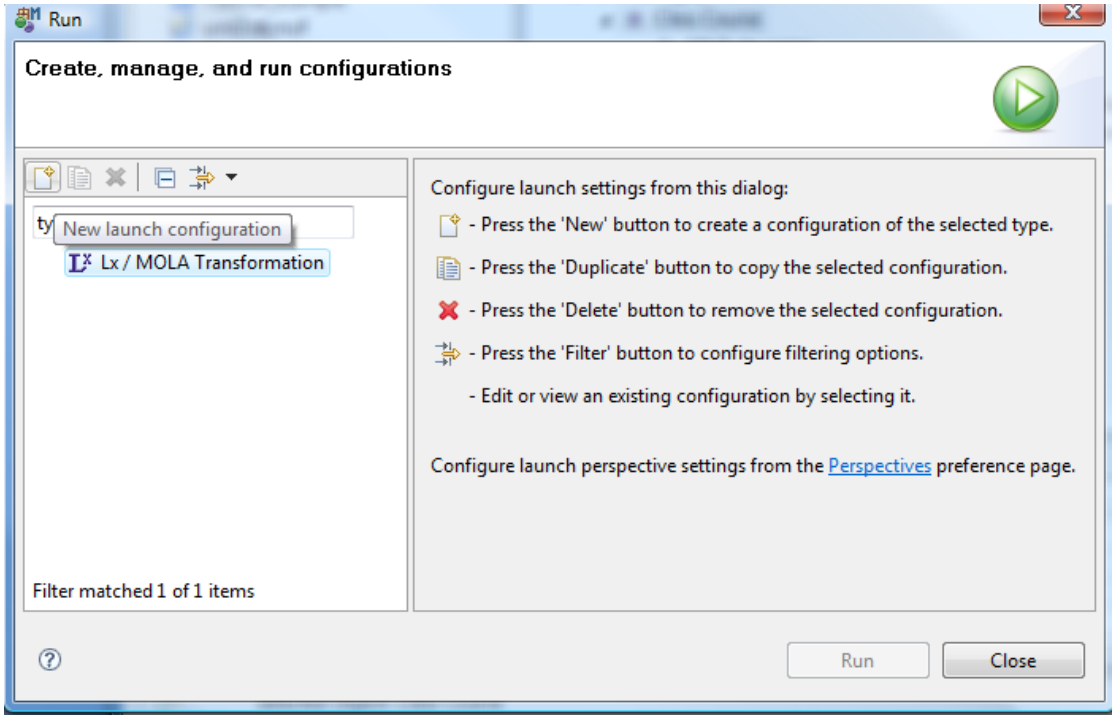

Figure 16 Creating new launch configuration

3) Choose transformation *jar* file, metamodel *ecore* files and model *xmi* files (see Fig. 17). Press “Run” to launch transformation. Model file should be reopened to see the result.

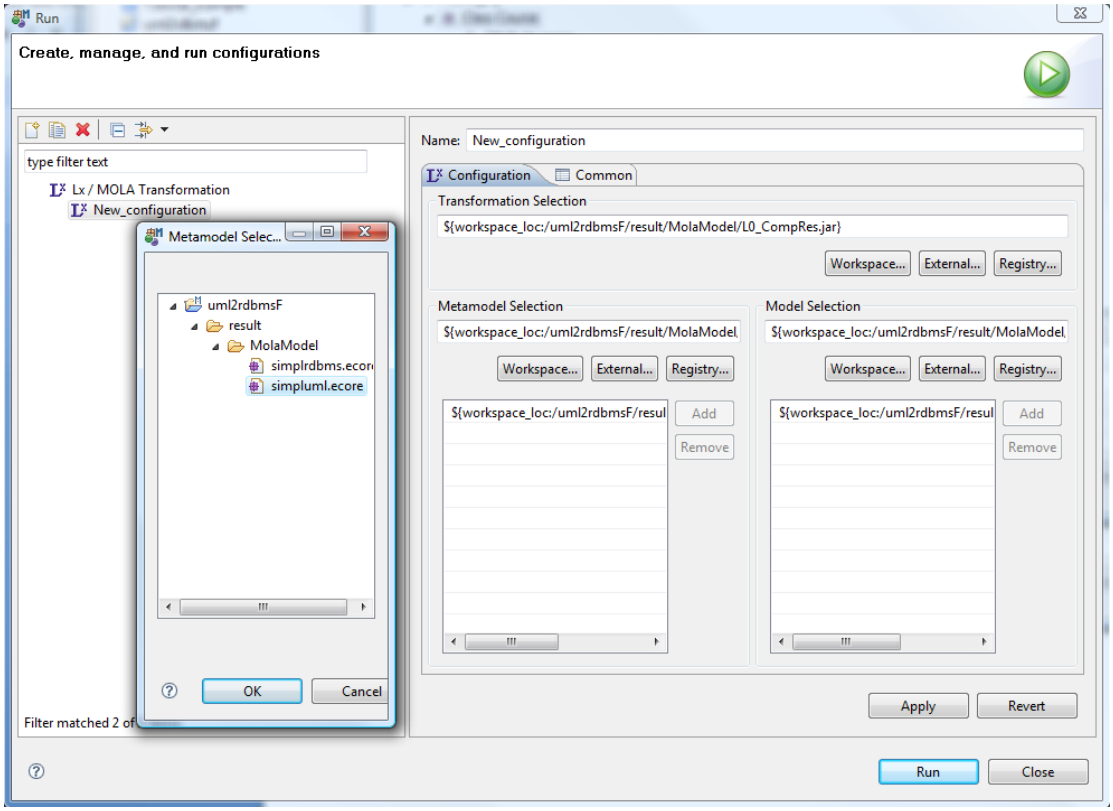

Figure 17 Configuring transformation runner

4 MOLA TDE

MOLA Transformation Development Environment (TDE) has been built on the basis of METAclipse tool building framework, which also has been developed by the University of Latvia, IMCS. METAclipse is a metamodel and transformation based tool building platform, which is specially fit for the support of complicated graphical domain specific languages, and MOLA is such a language. From the technical point of view, METAclipse is a set of Eclipse plugins which extend the functionality of standard Eclipse components EMF, GEF and partially, GMF. It contains advanced presentation engines, which support graphical diagram building, property editing and all other diagram and model related facilities. More precisely, the engines perform all the various visualisation and user interaction related tasks in a standard way typical to Eclipse environment, they do these jobs on the basis of a fixed presentation metamodel. However, the main functionality of a tool based on METAclipse is defined by transformations, which link the domain and presentation (visualisation) models in the tool, fill up property dialogs, and process the updated property values. In METAclipse framework these tool-specific transformations are built in MOLA.

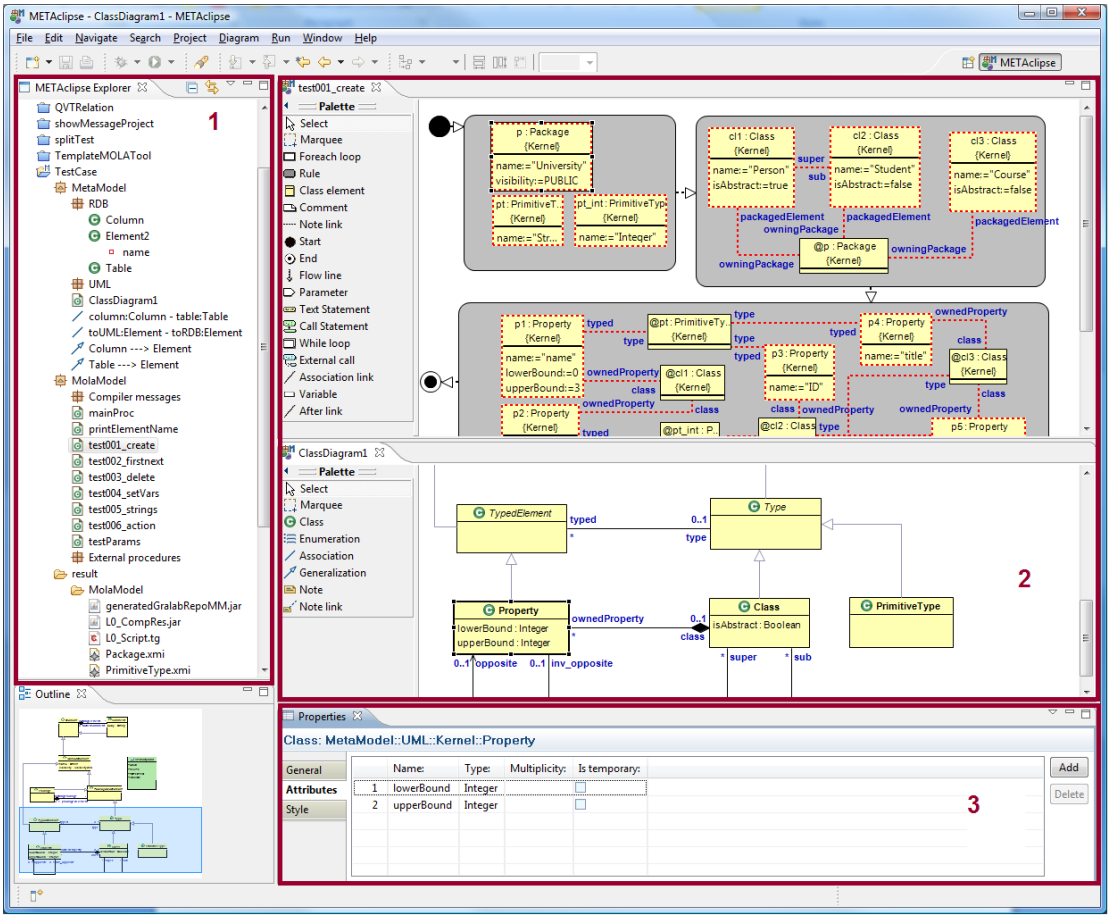

Figure 18 MOLA TDE

Since TDE has been built using Eclipse components, MOLA editors are similar to other tools built upon Eclipse. Therefore MOLA TDE uses the same workspace principle as any other Eclipse-based tool. MOLA workspace may contain any number of MOLA transformations. In fact, a single MOLA transformation is represented as an Eclipse project in MOLA TDE. It consists of metamodel and a set of MOLA procedures. Several editors in MOLA TDE are used to edit MOLA transformation:

- **METAclipse Explorer** (see Figure 18-1) is a tree-based editor which is used to manage MOLA transformations. It allows creating package hierarchy of metamodel and unit structure of MOLA transformation.

- **Graphical Editors** (see Figure 18-2) are graph-based editors which are used to edit metamodel elements (classes, associations, etc.) in class diagrams and MOLA elements (loops, rules, etc.) in MOLA procedures.

- **Property Editor** (see Figure 18-3) is a form-based editor which is used to manage property values of metamodel and MOLA elements selected in other editors.

There are two graphical editors in MOLA TDE – **Metamodel Editor** and **MOLA Editor**. They are used to edit class diagrams and MOLA diagrams (procedures) accordingly. The Metamodel Editor is described in Section 4.1 and MOLA Editor is described in Section 4.2.

Another important part of MOLA TDE is **MOLA Compiler**. It compiles MOLA transformation to C++ or Java code which can be run against mii_rep, JGralab or EMF repositories. MOLA Compiler is described in Section 4.3.

4.1 Metamodel Editor

4.1.1 Creating/Deleting Packages

Use METAclipse Explorer to create or delete a package. The tree node *MetaModel* represents a container for metamodels used by MOLA transformation. Packages can be created using context menu item *Add package* of any other package or *MetaModel* tree node. To delete a package use *Delete package* context menu item. All child elements of package are deleted too.

Note! Tree Editor may be used also to create and delete classes and enumerations. Use appropriate context menu items.

4.1.2 Creating/Deleting Class Diagrams

Use METAclipse Explorer to create or delete a class diagram. Class diagrams can be created using context menu item *Add class diagram* of any package or *MetaModel* tree node. To delete a class diagram use *Delete class diagram* context menu item.

Note! Deleting class diagram does not delete contained metamodel elements.

4.1.3 Creating/Deleting Class Diagram Elements

Use METAclipse Explorer to open class diagram. Double-click on class diagram node to open the appropriate diagram.

Use appropriate palette element to create a new metamodel element (class, enumeration, association, generalization or note). The new class or enumeration is added to package which contains the class diagram. Class attributes and enumeration literals can be added using appropriate context menu item or tab in the property editor.

Note! Class or enumeration can be moved to another package using *Package* property in the *General* tab.

Use *Delete from model* context menu item to permanently delete a metamodel element. It is removed from all diagrams where it appears.

Use *Delete from diagram* context menu item to remove a metamodel element (class, enumeration, association or generalization) representation from class diagram. A

Note! Metamodel element (class, enumeration, association or generalization) may appear in more than one class diagram. Use *Visualize in current diagram* context menu item on appropriate tree node in the METAclipse Explorer to add this metamodel element (class, enumeration, association or generalization) to the currently active class diagram. Use *Visualize class related* context menu item for class to add also associations and generalizations between this class and classes in the class diagram.

4.1.4 Importing Ecore Metamodels

Use **ECore Metamodel Import Wizard** (*File* | *Import ...* |*Metamodel Import* | *Ecore Metamodel Import)* to import Ecore metamodel into MOLA TDE.

Note! Use *METAclipse* | *Visualize invisible elements* context menu item on project node to visualize imported metamodel in the METAclipse Explorer tree.

4.2 MOLA Editor

4.2.1 Creating/Deleting MOLA Packages (Units)

Use METAclipse Explorer to create or delete a MOLA package. The tree node *MolaModel* represents a container for MOLA procedures. MOLA packages can be created using context menu item *Add package* of any other MOLA package or *MolaModel* tree node. To delete a package use *Delete package* context menu item. All child elements of package are deleted too.

Note! Package can be set as MOLA unit using *Is unit* property.

4.2.2 Creating/Deleting MOLA Procedures

Use METAclipse Explorer to create or delete a MOLA procedure. MOLA procedures can be created using context menu item *Add procedure* of any package or *MolaModel* tree node. To delete a MOLA procedure use *Delete procedure* context menu item.

Note! Use *Add external procedure* context menu item on the *External procedures* tree node to create an external procedure. The actual implementation of external procedure should be added to *UserCodePlaceHolde.cpp* or *java* file depending on target platform.

4.2.3 Editing MOLA Procedures

Use METAclipse Explorer to open MOLA diagram. Double-click on MOLA procedure node to open the appropriate diagram.

Use appropriate palette element to create a new MOLA element.

Use property editor to set properties of MOLA elements.

4.2.4. Importing/Exporting MOLA Procedures

Use *Export* context menu item on MOLA procedure node or MOLA package node to export procedure or package to XML file.

Note! Use this option on *MolaModel* node to export all MOLA transformations.

Use *Import* context menu item on MOLA package to import MOLA procedures into the MOLA transformation.

Note! The metamodel fragment used by imported MOLA procedures should be a subset of metamodel used by MOLA transformation in this project.

4.3 MOLA Compiler

MOLA compiler creates the executable file of transformation depending on the target platform. Use *Compile All* | *Compile to* * context menu item on *MolaModel* tree node to compile whole MOLA transformation.

4.3.1 Using MOLA to mii_rep (C++) Compiler

Use *Compile to C++* to compile MOLA transformation against **mii_rep** repository. The C++ project is created in the project folder *.mola\MolaModel\compres\comp_result_MII.* Open the *L0_CompRes_wgb_mii.bpr* file using **Borland Turbo C++ Explorer**. You can edit the *userCodePlaceHolder.cpp* file (add external procedure definitions). Build the project – the results are *L0_CompRes_wgb_mii.dll* file (executable) and *metamodel.xml* file (repository definition).

4.3.2 Using MOLA to JGralab Compiler

Use *Compile to JGralab* to compile MOLA transformation against **JGralab** repository. The results are *generatedGralabRepoMM.jar* (repository implementation classes), *L0_Script.tg* (repository definition) and *L0_CompRes.jar* (executable). These files are stored in the project folder *result\MolaModel*.

Note! If *result* folder does not appear in the METAclipse Explorer, then use *File* | *Refresh* menu item.

4.3.3 Using MOLA to EMF Compiler

Use *Compile to EMF* to compile MOLA transformation against **EMF** repository. The results are set of **.ecore* files (repository definition) and *L0_CompRes.jar* (executable). These files are stored in the project folder *result\MolaModel*.

Note! If *result* folder does not appear in the METAclipse Explorer, then use *File* | *Refresh* menu item.

4.3.4 Part of MOLA Language Recognized by Compiler

Currently the MOLA compiler supports:

A MOLA procedure may contain statements: *start*, *end*, *rule*, *text statement*, *call statement* (also *external*), *for-each loop* (**while loop** is not supported). Statements may be connected with control flows accordingly to their semantics.

A MOLA procedure may contain every kind of the *referencable elements* (*variable*, *parameter*, *class element*)

MOLA constructions (*text statement*, *class element*, *call statement*) may contain *expressions*. The MOLA compiler does not support **set expressions** and **emptiness verification** operations on attribute value. The **left part of a relation** in a constraint may not be the *toEnum()* function and the *NULL* constant. **The negative integer constants** are also not supported.

There are other constraints that are not listed here, but they are rarely used constructions. Therefore, they will be implemented in the next versions of MOLA compiler.

5 MOLA TEE

5.1 Transformation Runner

Use *Run* | *Run Configurations ...* menu item to invoke **Transformation Runner**. Create *Lx/ MOLA Transformation* launch configuration. Fill required fields in the configuration form.

- Transformation Selection – executable (L0_CompRes.jar)
- Metamodel Selection – repository definition for EMF (*.ecore) or repository implementation classes for JGralab (*generatedGralabRepoMM.jar)* .
- Model Selection – model definition for EMF (*.xmi) or repository definition for JGralab (*.tg)

Press *Run* to launch transformation.

5.2 Repository Browser

Use repository browser to view model and execute MOLA transformations. The main advantage of this tool is the capability to open all supported repository files.

Appendix A – Error Messages of the MOLA Compiler

1 - "The parameter has no type specification!"

The type has not been set for a parameter of an external procedure.

2 - "The parameter has no type specification!"

The type has not been set for a parameter of a MOLA procedure.

3 - "The variable has no type specification!"

The type has not been set for a variable.

4 - "The function toEnum used in a non-enumeration expression!"

The function toEnum used in a non-enumeration expression.

5 - "Too much parameters in a call of the toEnum!"

A call of the function toEnum in an expression has more than one call parameter.

6 - "No parameters in a call of the toEnum!"

A call of the function toEnum in an expression has no call parameters.

7 - "There are multiple identical outgoing flows from the flowend!"

The graphical statement has more then one outgoing control flows of the same type. (More than one ELSE or non-ELSE flow)

8 - "Too much parameters in a call of the *function_name* function!"

A call of a one-argument function (toInteger, size, toString, toUpper, toLower, toBoolean) in an expression has more than one call parameter.

9 - "No parameters in a call of the *function_name* function!"

A call of a one-argument function (toInteger, size, toString, toUpper, toLower, toBoolean) in an expression has no call parameter.

10 - "Not enough parameters in a call of the indexOf function!"

A call of the function indexOf in an expression has less than two call parameters.

11 - "Too much parameters in a call of the indexOf function!"

A call of the function indexOf in an expression has more than two call parameters.

12 - "Not enough parameters in a call of the substring function!"

A call of the function substring in an expression has less than two call parameters.

13 - "Too much parameters in a call of the substring function!"

A call of the function substring in an expression has more than three call parameters.

14 - "Expression formation error in a call of the toString function!"

15 - "String expression in a call of the toString function!"

A string expression is supplied as a call parameter to the toString function.

16 - "Class-typed expression in a call of the toString function!"

A class-typed expression is supplied as a call parameter to the toString function.

17 - "A simple expression contains prohibited symbol (>,<,>=,<=,=,<>,.,comma(,),and,or,not)!"

Keywords and, or, not or symbols >,<,<=,>=,=,<> is used in an expression (in an assignment or a call parameter). Also a wrong usage of the comma (,) or the dot (.) in any expression may cause this error.

18 – "A type mismatch in an expression by attribute specification "*attribute_name*" !"

The attribute of a wrong type is used. Only the name of the attribute is displayed in the error message (also when a pointer with dot is specified before it)

19 - "A type mismatch in an expression by referencable element specification

"*referencable_element_name*" !"

A variable, parameter or class element used in an expression of an incompatible type.

20 - "An unknown identifier "*identifier*" !"

The unknown identifier is used in non-enumeration expression.

21 – "An unknown enumeration literal "*literal*" !"

The unknown identifier is used in an enumeration expression.

22 - "No left side in a relation!"

The left side of a relation (to the left from <,>,...) is missing in a constraint.

23 - "No right side in a relation!"

The right side of a relation (to the right from <,>,...) is missing in a constraint.

24 - "Bracket error in a Boolean expression!"

There are bracket mismatch (no enclosing or opening bracket or …) in an expression.

25 - "OR has no right operand!"

An OR-expression has no right operand.

26 - "OR has no left operand!"

An OR-expression has no left operand.

27 - "AND has no right operand!"

An AND-expression has no right operand.

28 - "AND has no left operand"

An AND-expression has no left operand.

29 - "NOT has no operand!"

A NOT-expression has no operand.

30 - "An expression ends with the dot!"

There is an incorrect usage of dot (‘.’) symbol in an expression – it is the last symbol in the expression.

31 - "The keyword SELF used not in a class element!"

The keyword ‘self’ is used not in class element (e.g. in text statement or call statement). Note, ‘self’ keyword can be used in a class element only.

32 - "An unknown identifier "*identifier*" before a dot!"

An identifier before a dot (‘.’) symbol is unknown. It must be a valid pointer name.

33 -"A referencable element "*referencable_element_name*" before a dot is not class-typed (pointer)!"

An identifier before a dot (‘.’) symbol is referencable element name, but not pointer. There is no dot operand for primitive-typed or enumeration elements.

34 -"An identifier "*identifier*" after a dot is not attribute specification!"

An identifier after a dot (‘.’) symbol is not the attribute specification (current version of compiler does not support set-expressions, therefore only attribute specifications are allowed)

35 - "Type mismatch by attribute specification "*attribute_specification*" !"

An attribute specification is used in expression of different type.

36 - "Incorrect syntax of the "*method_name*" method!"

37 - "Unknown type specification "*identifier*" in a isTypeOf() method!"

An argument in the ’isTypeOf’ method is not a valid type specification.

38 - "Type mismatch by downcast "*type_specification*" !"

A type specified in the downcast is not compatible with type of expression.

39 -"Unknown type specification "*identifier*" in a isKindOf() method!"

An argument in the ’isKindOf’ method is not a valid type specification.

40 -"Missing a pointer in a downcast "*identifier*" !"

41 - "An unknown identifier "*identifier*" in downcast!"

42 - "The type of the referencable element "*identifier*" is not a class!"

43 - "Type mismatch in a downcast by referencable element "*identifier*" !"

44 - "Missing right bracket in a downcast "*identifier*" !"

45 - "Type mismatch by function "*identifier*" !"

46 - "Type mismatch by constant "*identifier*" !"

47 - "Type mismatch by operator "*identifier*" !"

48 - "Bracket error!"

49 - "Right bracket missing in isTypeOf/isKindOf method!"

50 - "There are more than one main procedure!"

51 - "No called procedure set!"

52 - "The call parameter "*identifier*" and the actual parameter have incompatible types!"

53 - "Not enough call parameters!"

A call statement has less call parameters than specified by the called procedure. A call statement has wrong numbering of call parameters.

54 - "Too many call parameters!"

55 - "No loop variable in the loophead"

56 - "A delete or normal association link is to a create class element!"

57 - "A rule is empty!"

58 - "A class element has no class specified!"

59 - "An association link has no association specified!"

60 - "A loop variable is reference!"

61 - "A NOT-element must be a normal class element!"

62 - "A NOT-link must be a normal association link!"

63 - "No loophead in a foreach loop!"

64 - "Nothing selected to compile!"

65 - "An unknown identifier *identifier* in a constraint!"

66 – "No incoming control flow!"

67 – "The alternative(ELSE) outgoing control flow is not permitted!"

68 – "No outgoing control flow!"

69 – "The incoming control flow not permitted to loophead!"

70 – "No identifier after arrow(->) in an expression!"

71 – "An unknown identifier *"identifier"* after arrow(->) in an expression!"

72– "An identifier *"identifier"* used in the isEmpty or notEmpty method is not property name!!"

73 – "The opening bracket missing after isEmpty or notEmpty method!!"

74 – "The closing bracket missing after isEmpty or notEmpty method!"

75 – "There is no loophead in a while-loop!"

76 - "Assigned element not set!"

Assignment created in a text statement, but variable has not been set. Active, empty row is found in the grid.

77 - "Assigned property not set!"

Assignment created in a class element, but property has not been set. Active, empty row is found in the grid.

78 - "Class element with non-class type!"

79 - "Creating instance of abstract class!"

1001 - "MODEL CONSISTENCE ERROR!!!!!!"