Developer documentation of VCE behavioral editor

Krzysztof Nirski

October 24, 2008

1 Overview

This documentation covers implementation details of VCE behavioral editor.

VCE is a software (in the form of set of Eclipse plugins) which purpose is to edit formal models graphically. It is splitted into two parts - structural and behavioral. Structural part is more or less complete[1], while behavioral is still in alpha state. As of today this two sets of plugins can work together in same Eclipse instance, nonetheless they are not cooperating. These two parts own their own resources, meaning that there is no reference for behavior of model in structural part and vice versa. There will be explained how to try to make them cooperate in one of the following chapters.

Basically, the behavior editor is inspired by UML state and activity diagrams [needs some more explanation]. It introduces concept of state (along with initial and final state) and metatransition which connects two states.

One of the important aspects of behavioral editor is a module based architecture of the core - its metamodel. Metamodel consists of two parts: the basic one defining prime concepts and the metatransition implementation (defining what kind of statements does it consist of). There are two different metatransition implementations. The first one (we will refer to it as ,,simple") allows only one statement (communication statement or event statement) per metatransition.

The semantic meaning of event statement is that id defines whole metatransition as internal behavior, which we only describe by label. On the other hand, communication statement can be either dispatch or collection of a message on particular channel, which in fact is infinite queue and enables communication between state machines.

1.1 Structured metatransition

Second metatransition implementation defines it as list of statements (in any amount and order): event (arbitrary operation), guard, assignment, message dispatch, exit. The metatransition itself becomes a block of structured sequential code. Exit is referencing target state, to which transistion is possible reaching that point in code. Guard is a control flow statement, which evaluates expression to determine whether execute embedded statement. Assignment evaluates expression and assigns its value to previously defined variable. Expression can be message collection in particular.

The important difference in comparison to simple metatransition is that it can include several outgoing links bound to states, i.e. in structured code

there are exit statements which can refer to different exit points.

2 Used tools

The main tools used in implementation are Eclipse Modeling Framework[2], Graphical Modeling Framework[3] and Graphical Editing Framework[4].

2.1 EMF and metamodel

EMF is a modeling framework and code generation facility, which is used to build tools based on data model. It introduces metamodel concept, which is a set of packages, classes, attributes and references like UML class diagram. It provides tools and runtime support to produce Java classes for the model, executing commands (own command stack) and a basic editor. Models can be specified using annotated Java code, UML, XML and other tools such as Rationnal Rose. In our case models are kept in ecore files (*.ecore) in XMI format.

2.2 GMF

A glue-framework which is meant to make cooperation between EMF and GEF easier. It is used to generate editor code from gmf descriptors: graphical definition descriptor (*.gmfgraph), tooling definition descriptor (*.gmftool), mapping descriptor (*.gmfmap), generator model based on mapping descriptor (*.gmfgen) and ecore metamodel. A process of assembling GMF application is shown on figure 1.

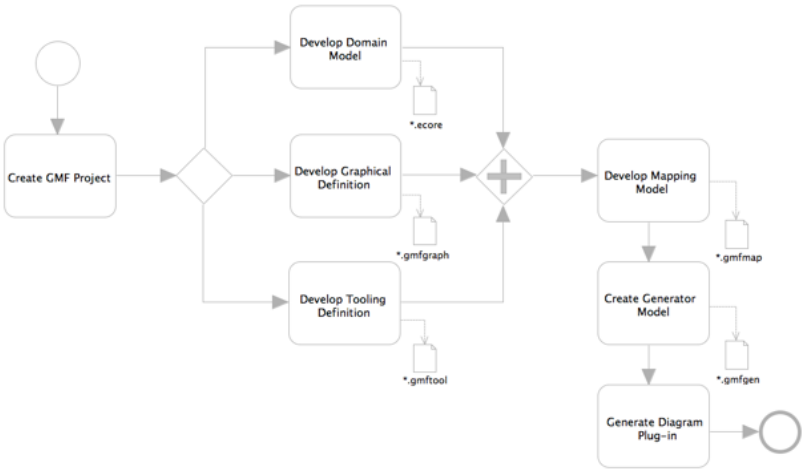

Figure 1: GMF workflow

2.3 GEF

A model-view-controller framework for Eclipse platform. The main concept extensively used by GMF to update ecore model is controller (an *EditPart class), although GEF can work with any model resource, or even without.

GEF consists of 2 plug-ins. The org.eclipse.draw2d plug-in provides a layout and rendering toolkit for displaying graphics. The developer can then take advantage of the many common operations provided in GEF and/or extend them for the specific domain.

Unfortunately, the payoff for flexibility is steep learning curve.

2.4 Documentation sources

- http://www.eclipse.org/articles/Article-Using%20EMF/using-emf.html - gives a good basic overview on EMF
- http://wiki.eclipse.org/GMF_Documentation_Index
- http://wiki.eclipse.org/index.php/GMF_Tutorial
- EclipseCons slides
- GMF developer guide in eclipse help system (unfortunately outdated and incomplete, there is alway tutorial from wiki included)

- newsgroups (in order of importance):
  - eclipse.modeling.gmf
  - eclipse.modeling.emf
  - eclipse.tools.gef
  - eclipse.tools.emf
  - eclipse.modeling.mdt
  - eclipse.modeling.mdt.ocl
  - eclipse.modeling.mdt.uml2
  - eclipse.modeling.mdt.uml2tools

2.4.1 Examples

The place to go for examples for GMF, GEF and EMF is Eclipse's CVS repository. Address: dev.eclipse.org , directory: /cvsroot/modeling. Login is anonymous, password is not required.

Mindmap example is explained in GMF tutorial. Taipan example is unfortunately not working properly - it has something to do with GMF version, but I am unsure what is the exact reason of it.

To go over GEF's features, check the Logic example.

3 Project execution environment

Having generator model, we can generate plugin code and run it as Eclipse Application (Eclipse will automatically load plugin code). In order to debug application just run it as Eclipse application in debug mode, and then you can change code while application is running, though you cannot add or delete methods and class-scoped variables. Now you can create new project (,,Empty EMF project") and then you can create new diagram files along with files containing model instance - you can do it either by choosing in wizard either ,,MetatransitionSimple Diagram" or ,,MetatransitionStructured" (see figure 2).

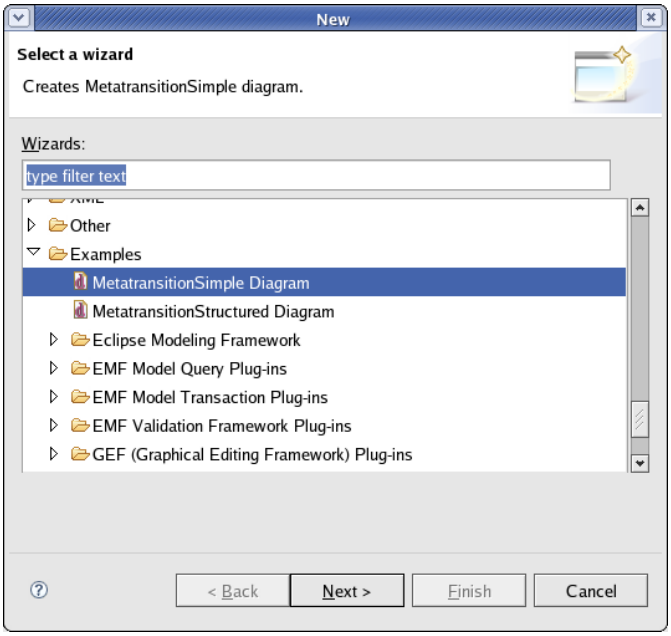

Figure 2: Eclipse wizard for creating resources

3.1 Dependencies

In order to execute diagram editor plugins it is needed have Eclipse 3.4.0 Ganymede SDK distribution installed as well as EMF 2.4, GMF 2.1 and GEF 3.4 plugin SDK bundles along with their own dependencies. Additionaly these plugins are needed: Ecore Tools 0.8, Eclipse MDT's UML2 Tools 0.8 and OCL 1.2 bundles. Set of plugins known to work:

- ca.ecliptical.emf.xpath_1.0.0
- com.ibm.icu_3.8.1.v20080530
- com.jcraft.jsch_0.1.37.v200803061811
- edu.cmu.sei.aadl.annex_1.4.2
- edu.cmu.sei.aadl.architecture_1.4.5
- edu.cmu.sei.aadl.flowanalysis_1.4.7
- edu.cmu.sei.aadl.help_1.5.4
- edu.cmu.sei.aadl.instance_1.5.4
- edu.cmu.sei.aadl.model.edit_1.4.3
- edu.cmu.sei.aadl.model.editor_1.4.5
- edu.cmu.sei.aadl.model_1.5.4
- edu.cmu.sei.aadl.parser_1.4.10
- edu.cmu.sei.aadl.properties.SEI_1.0.4
- edu.cmu.sei.aadl.resourcebudgets_1.4.6
- edu.cmu.sei.aadl.resourcemanagement_1.4.8
- edu.cmu.sei.aadl.security_1.4.4
- edu.cmu.sei.aadl.texteditor_1.4.5
- edu.cmu.sei.aadl.toMetaH_1.4.7
- edu.cmu.sei.aadl.unparser_1.4.5
- edu.cmu.sei.aadl_1.5.4
- edu.cmu.sei.osate-frontend.source_1.5.4

- edu.cmu.sei.osate-plugins.source_1.5.2
- edu.cmu.sei.osate.core_1.4.8
- edu.cmu.sei.osate.pluginsupport_1.4.2
- edu.cmu.sei.osate.propertyview_1.4.7
- edu.cmu.sei.osate.reporter_1.4.2
- edu.cmu.sei.osate.ui_1.4.6
- edu.cmu.sei.osate.wizards_1.4.0
- edu.cmu.sei.osate.workspace_1.4.5
- edu.cmu.sei.timeweaver.binpacking_1.4.2
- fr.obeo.acceleo.bridge.ui_2.2.0.200801071903
- fr.obeo.acceleo.bridge_2.2.0.200801071903
- fr.obeo.acceleo.chain.edit_2.2.0.200801071903
- fr.obeo.acceleo.chain.editor_2.2.0.200801071903
- fr.obeo.acceleo.chain.ui_2.2.0.200801071903
- fr.obeo.acceleo.chain_2.2.0.200801071903
- fr.obeo.acceleo.ecore.ui_2.2.0.200801071903
- fr.obeo.acceleo.ecore_2.2.0.200801071903
- fr.obeo.acceleo.gen.debug.ui_2.2.0.200801071903
- fr.obeo.acceleo.gen.ui.help_2.2.0.200801071903
- fr.obeo.acceleo.gen.ui_2.2.0.200801071903
- fr.obeo.acceleo.gen_2.2.0.200801071903
- fr.obeo.acceleo.template.edit_2.2.0.200801071903
- fr.obeo.acceleo.template.gen_2.2.0.200801071903
- fr.obeo.acceleo.template_2.2.0.200801071903
- fr.obeo.acceleo.tools.ui_2.2.0.200801071903
- fr.obeo.acceleo.tools_2.2.0.200801071903
- fr.obeo.acceleo.uml13_2.2.0.200801071903
- fr.obeo.acceleo.uml14.mof_2.2.0.200801071903
- fr.obeo.acceleo.uml14.ui_2.2.0.200801071903
- fr.obeo.acceleo.uml14_2.2.0.200801071903
- javax.servlet.jsp_2.0.0.v200806031607
- javax.servlet_2.4.0.v200806031604
- javax.wsdl15_1.5.1.v200705290614
- javax.wsdl_1.4.0.v200706111329
- javax.xml.rpc_1.1.0.v200706111329
- javax.xml.soap_1.2.0.v200706111329
- javax.xml_1.3.4.v200806030440
- net.sourceforge.lpg.lpgjavaruntime_1.1.0.v200803061910
- org.apache.ant_1.7.0.v200803061910
- org.apache.axis_1.4.0.v200706191647
- org.apache.batik.bridge_1.6.0.v200805290154
- org.apache.batik.css_1.6.0.v200805290154
- org.apache.batik.dom.svg_1.6.0.v200805290154
- org.apache.batik.dom_1.6.0.v200805290154
- org.apache.batik.ext.awt_1.6.0.v200805290154
- org.apache.batik.parser_1.6.0.v200805290154
- org.apache.batik.pdf_1.6.0.v200806031500
- org.apache.batik.svggen_1.6.0.v200805290154
- org.apache.batik.transcoder_1.6.0.v200805290154
- org.apache.batik.util.gui_1.6.0.v200805290154
- org.apache.batik.util_1.6.0.v200805290154
- org.apache.batik.xml_1.6.0.v200805290154
- org.apache.commons.cli_1.0.0
- org.apache.commons.discovery_0.2.0.v200706111329
- org.apache.commons.el_1.0.0.v200806031608
- org.apache.commons.logging_1.0.4.v200701082340
- org.apache.commons.logging_1.0.4.v200706111724
- org.apache.commons.logging_1.0.4.v20080605-1930
- org.apache.fop_0.20.5
- org.apache.jasper_5.5.17.v200806031609
- org.apache.log4j_1.2.13.v200706111418
- org.apache.log4j_1.2.8.v200704240447
- org.apache.lucene.analysis_1.9.1.v20080530-1600
- org.apache.lucene_1.9.1.v20080530-1600
- org.apache.wsil4j_1.0.0.v200706111329
- org.apache.xerces_2.8.0.v200705301630
- org.apache.xerces_2.9.0.v200805270400
- org.apache.xml.resolver_1.1.0.v200705310020
- org.apache.xml.resolver_1.2.0.v200806030312
- org.apache.xml.serializer_2.7.1.v200806030322
- org.eclipse.draw2d.doc.isv_3.4.0.v20080606
- org.eclipse.draw2d.source_3.4.0.v20080115-33-7w3119163_
- org.eclipse.draw2d_3.4.0.v20080529
- org.eclipse.ecf.filetransfer_2.0.0.v20080611-1715
- org.eclipse.ecf.identity_2.0.0.v20080611-1715
- org.eclipse.ecf.provider.filetransfer_2.0.0.v20080611-1715

- org.eclipse.ecf_2.0.0.v20080611-1715
- org.eclipse.emf.activities_2.4.0.v200808251517
- org.eclipse.emf.ant_2.4.0.v200808251517
- org.eclipse.emf.cheatsheets_2.4.0.v200808251517
- org.eclipse.emf.codegen.ecore.ui_2.4.1.v200808251517
- org.eclipse.emf.codegen.ecore_2.4.1.v200808251517
- org.eclipse.emf.codegen.ui_2.4.0.v200808251517
- org.eclipse.emf.codegen_2.4.0.v200808251517
- org.eclipse.emf.common.ui_2.4.0.v200808251517
- org.eclipse.emf.common_2.4.0.v200808251517
- org.eclipse.emf.commonj.sdo_2.4.0.v200808251517
- org.eclipse.emf.converter_2.4.0.v200808251517
- org.eclipse.emf.databinding.edit_1.0.0.v200808251517
- org.eclipse.emf.databinding_1.0.0.v200808251517
- org.eclipse.emf.doc_2.4.1.v200808251517
- org.eclipse.emf.ecore.change.edit_2.4.0.v200808251517
- org.eclipse.emf.ecore.change_2.4.0.v200808251517
- org.eclipse.emf.ecore.edit_2.4.1.v200808251517
- org.eclipse.emf.ecore.editor_2.4.0.v200808251517
- org.eclipse.emf.ecore.sdo.doc_2.4.0.v200808251517
- org.eclipse.emf.ecore.sdo.edit_2.4.0.v200808251517
- org.eclipse.emf.ecore.sdo.editor_2.4.0.v200808251517
- org.eclipse.emf.ecore.sdo.source_2.4.0.v200808251517
- org.eclipse.emf.ecore.sdo_2.4.0.v200808251517
- org.eclipse.emf.ecore.xmi_2.4.1.v200808251517
- org.eclipse.emf.ecore_2.4.1.v200808251517
- org.eclipse.emf.ecoretools.diagram.search_0.7.0.v200806130939
- org.eclipse.emf.ecoretools.diagram.ui.outline_0.8.0.v200806130600
- org.eclipse.emf.ecoretools.diagram_0.8.0.v200806130600
- org.eclipse.emf.ecoretools.doc_0.8.0.v200806130600
- org.eclipse.emf.ecoretools.filters_0.8.0.v200806130600
- org.eclipse.emf.ecoretools.properties_0.8.0.v200806130600
- org.eclipse.emf.ecoretools.source_0.8.0.v200806130600
- org.eclipse.emf.ecoretools.tabbedproperties_0.8.0.v200806130600
- org.eclipse.emf.ecoretools_0.8.0.v200806130600
- org.eclipse.emf.edit.ui_2.4.1.v200808251517
- org.eclipse.emf.edit_2.4.1.v200808251517
- org.eclipse.emf.example.installer_1.0.0.v200808251517
- org.eclipse.emf.examples.generator.validator_1.1.0.v200808251517
- org.eclipse.emf.examples.library.edit_2.3.0.qualifier
- org.eclipse.emf.examples.library.editor_2.3.0.qualifier
- org.eclipse.emf.examples.library_2.3.0.qualifier
- org.eclipse.emf.examples.source_2.4.0.v200808251517
- org.eclipse.emf.examples_2.4.0.v200808251517
- org.eclipse.emf.exporter.html_2.4.0.v200808251517
- org.eclipse.emf.exporter_2.4.0.v200808251517
- org.eclipse.emf.importer.ecore_2.4.0.v200808251517
- org.eclipse.emf.importer.java_2.4.1.v200808251517
- org.eclipse.emf.importer.rose_2.4.0.v200808251517
- org.eclipse.emf.importer_2.4.1.v200808251517
- org.eclipse.emf.java.edit_2.4.0.v200808251517
- org.eclipse.emf.java.editor_2.4.0.v200808251517
- org.eclipse.emf.java_2.4.0.v200808251517
- org.eclipse.emf.mapping.ecore.editor_2.4.0.v200808251517
- org.eclipse.emf.mapping.ecore2ecore.editor_2.4.0.v200808251517
- org.eclipse.emf.mapping.ecore2ecore_2.4.0.v200808251517
- org.eclipse.emf.mapping.ecore2xml.ui_2.4.0.v200808251517
- org.eclipse.emf.mapping.ecore2xml_2.4.0.v200808251517
- org.eclipse.emf.mapping.ecore_2.4.0.v200808251517
- org.eclipse.emf.mapping.ui_2.4.0.v200808251517
- org.eclipse.emf.mapping.xsd2ecore.editor_2.4.0.v200808251517
- org.eclipse.emf.mapping.xsd2ecore_2.4.0.v200808251517
- org.eclipse.emf.mapping_2.4.0.v200808251517
- org.eclipse.emf.ocl.doc_1.1.101.v200807161725
- org.eclipse.emf.ocl.examples.interpreter_1.2.0.qualifier
- org.eclipse.emf.ocl.examples.source_1.2.0.v200805130238
- org.eclipse.emf.ocl.examples_1.2.0.v200805130238
- org.eclipse.emf.ocl.source_1.1.100.v200805130238-108Y7w311916241349
- org.eclipse.emf.ocl_1.1.100.v200805130238
- org.eclipse.emf.query.doc_1.2.0.v200805130238
- org.eclipse.emf.query.examples_1.2.0.v200805130238
- org.eclipse.emf.query.ocl.source_1.2.0.v200805130238-11-7w311916241349
- org.eclipse.emf.query.ocl_1.2.0.v200805130238
- org.eclipse.emf.query.source_1.2.0.v200805130238-11-7w311916241349
- org.eclipse.emf.query_1.2.0.v200805130238
- org.eclipse.emf.search.common.ui_0.7.0.v200806130939

- org.eclipse.emf.search.common_0.7.0.v200806130939
- org.eclipse.emf.search.ecore.ocl.ui_0.7.0.v200806130939
- org.eclipse.emf.search.ecore.ocl_0.7.0.v200806130939
- org.eclipse.emf.search.ecore.ui_0.7.0.v200806130939
- org.eclipse.emf.search.ecore_0.7.0.v200806130939
- org.eclipse.emf.search.ocl.ui_0.7.0.v200806130939
- org.eclipse.emf.search.ocl_0.7.0.v200806130939
- org.eclipse.emf.search.ui_0.7.0.v200806130939
- org.eclipse.emf.search_0.7.0.v200806130939
- org.eclipse.emf.source_2.4.1.v200808251517
- org.eclipse.emf.transaction.doc_1.2.0.v200805130238
- org.eclipse.emf.transaction.examples_1.2.0.v200806051817
- org.eclipse.emf.transaction.source_1.2.1.v200807161719-2308s733I3E5B4B7J
- org.eclipse.emf.transaction.ui_1.2.0.v200805130238
- org.eclipse.emf.transaction_1.2.0.v200805130238
- org.eclipse.emf.validation.doc_1.2.1.v200807161729
- org.eclipse.emf.validation.examples.adapter_1.2.0.v200805130238
- org.eclipse.emf.validation.examples.general_1.2.0.v200805251807
- org.eclipse.emf.validation.examples.ocl_1.2.0.v200805130238
- org.eclipse.emf.validation.examples.source_1.2.0.v200805130238
- org.eclipse.emf.validation.examples_1.2.0.v200805280204
- org.eclipse.emf.validation.ocl.source_1.2.0.v200805130238-11-7w311916241349
- org.eclipse.emf.validation.ocl_1.2.0.v200805130238
- org.eclipse.emf.validation.source_1.2.0.v200805130238-35-9oA55S5L8G6FCT
- org.eclipse.emf.validation.ui.ide_1.2.0.v200805130238
- org.eclipse.emf.validation.ui_1.2.0.v200805130238
- org.eclipse.emf.validation_1.2.0.v200805170232
- org.eclipse.emf.workspace.doc_1.2.0.v200805130238
- org.eclipse.emf.workspace.source_1.2.1.v200807161719-2308s733I3E764D6E
- org.eclipse.emf.workspace.ui_1.2.0.v200805130238
- org.eclipse.emf.workspace_1.2.0.v200805130238
- org.eclipse.emf_2.4.0.v200808251517
- org.eclipse.gef.doc.isv_3.4.0.v20080606
- org.eclipse.gef.examples.ui.pde_3.4.0.v20080226
- org.eclipse.gef.source_3.4.0.v20080115-677-8082A5696H274A
- org.eclipse.gef_3.4.0.v20080526
- org.eclipse.gmf.bridge.ui.dashboard_2.0.0.v20080417-1610
- org.eclipse.gmf.bridge.ui_1.1.100.v20080417-1610
- org.eclipse.gmf.bridge_1.1.0.v20080528-1052
- org.eclipse.gmf.codegen.edit_2.1.0.v20080610-1132
- org.eclipse.gmf.codegen.ui_1.1.0.v20080512-1200
- org.eclipse.gmf.codegen_2.1.0.v20080610-1132
- org.eclipse.gmf.common_1.1.1.v20080610-1132
- org.eclipse.gmf.doc_1.2.0.v20080516-1143
- org.eclipse.gmf.ecore.editor_2.0.100.v20080610-1132
- org.eclipse.gmf.examples.ui.pde_1.0.200.v20080425-1959
- org.eclipse.gmf.examples_1.0.100.v20080425-1959
- org.eclipse.gmf.graphdef.codegen.ui_1.0.100.v20080425-1959
- org.eclipse.gmf.graphdef.codegen_2.0.100.v20080528-1052
- org.eclipse.gmf.graphdef.edit_2.0.100.v20080610-1132
- org.eclipse.gmf.graphdef_2.0.100.v20080528-1052
- org.eclipse.gmf.map.edit_2.1.0.v20080610-1132
- org.eclipse.gmf.map_2.1.0.v20080521
- org.eclipse.gmf.runtime.common.core_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.common.ui.action.ide_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.common.ui.action_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.common.ui.printing_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.common.ui.services.action_1.1.0.v20080507-2230
- org.eclipse.gmf.runtime.common.ui.services.dnd.ide_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.common.ui.services.dnd_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.common.ui.services.properties_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.common.ui.services_1.1.0.v20080612-1229
- org.eclipse.gmf.runtime.common.ui_1.1.0.v20080512-1200
- org.eclipse.gmf.runtime.diagram.core_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.diagram.ui.actions_1.1.0.v20080603-1553
- org.eclipse.gmf.runtime.diagram.ui.dnd_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.diagram.ui.geoshapes_1.1.0.v20080503-1740
- org.eclipse.gmf.runtime.diagram.ui.printing.render_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.diagram.ui.printing_1.1.0.v20080501-1739
- org.eclipse.gmf.runtime.diagram.ui.properties_1.1.0.v20080603-1553
- org.eclipse.gmf.runtime.diagram.ui.providers.ide_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.diagram.ui.providers_1.1.0.v20080503-1740
- org.eclipse.gmf.runtime.diagram.ui.render_1.1.0.v20080603-1553
- org.eclipse.gmf.runtime.diagram.ui.resources.editor.ide_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.diagram.ui.resources.editor_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.diagram.ui_1.1.0.v20080610-1132

- org.eclipse.gmf.runtime.draw2d.ui.render.awt_1.1.0.v20080603-1553
- org.eclipse.gmf.runtime.draw2d.ui.render_1.1.0.v20080507-2230
- org.eclipse.gmf.runtime.draw2d.ui_1.1.0.v20080610-1132
- org.eclipse.gmf.runtime.emf.clipboard.core_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.emf.commands.core_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.emf.core_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.emf.type.core_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.emf.type.ui_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.emf.ui.properties_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.emf.ui_1.1.0.v20080516-1748
- org.eclipse.gmf.runtime.gef.ui_1.1.0.v20080503-1740
- org.eclipse.gmf.runtime.notation.edit_1.1.0.v20080507-1326
- org.eclipse.gmf.runtime.notation.providers_1.1.0.v20080425-1959
- org.eclipse.gmf.runtime.notation_1.1.0.v20080507-1326
- org.eclipse.gmf.sdk_1.0.0.v20080425-1959
- org.eclipse.gmf.tooldef.edit_2.0.0.v20080610-1132
- org.eclipse.gmf.tooldef_2.0.0.v20080417-1610
- org.eclipse.gmf.tooling_2.1.0.v20080425-1959
- org.eclipse.gmf.validate_1.1.0.v20080603-1553
- org.eclipse.gmf.xpand.editor_1.0.0.v20080425-1959
- org.eclipse.gmf.xpand_1.1.0.v20080528-1052
- org.eclipse.gmf_1.0.0.v20080425-1959
- org.eclipse.jem.util_2.0.100.v200805140020
- org.eclipse.jface.databinding_1.2.0.I20080515-2000a
- org.eclipse.jface.text_3.4.0.v20080603-2000
- org.eclipse.jface_3.4.0.I20080606-1300
- org.eclipse.jsch.core_1.1.100.I20080604
- org.eclipse.jsch.ui_1.1.100.I20080415
- org.eclipse.ltk.core.refactoring_3.4.0.v20080603-2000
- org.eclipse.ltk.ui.refactoring_3.4.0.v20080605-1800
- org.eclipse.m2m.atl.adt.builder_2.0.0.v200801311241
- org.eclipse.m2m.atl.adt.debug_2.0.0.v200801311241
- org.eclipse.m2m.atl.adt.editor_2.0.0.v200801311241
- org.eclipse.m2m.atl.adt.perspective_2.0.0.v200801311241
- org.eclipse.m2m.atl.adt.wizard_2.0.0.v200801311241
- org.eclipse.m2m.atl.compilers.atl2006_2.0.0.v200801311241
- org.eclipse.m2m.atl.doc_2.0.0.v200801311241
- org.eclipse.m2m.atl.drivers.emf4atl_2.0.0.v200801311241
- org.eclipse.m2m.atl.drivers.uml24atl_2.0.0.v200801311241
- org.eclipse.m2m.atl.engine.vm_2.0.0.v200801311241
- org.eclipse.m2m.atl.engine_2.0.0.v200801311241
- org.eclipse.m2m.atl.ocl.core_2.0.0.v200801311241
- org.eclipse.m2m.atl.service.core_2.0.0.v200801311241
- org.eclipse.m2m.atl.source_2.0.0.v200801311241
- org.eclipse.ocl.doc_1.2.1.v200807161725
- org.eclipse.ocl.ecore_1.2.1.v200807301850
- org.eclipse.ocl.source_1.2.1.v200807161725-3419oA55S5L9G8SC_
- org.eclipse.ocl.uml.source_1.2.1.v200807161725-1107w311918272836
- org.eclipse.ocl.uml_1.2.1.v200807161725
- org.eclipse.ocl_1.2.1.v200807161725
- org.eclipse.uml2.codegen.ecore.ui_1.4.0.v200805131030
- org.eclipse.uml2.codegen.ecore_1.4.0.v200805201126
- org.eclipse.uml2.common.edit_1.4.0.v200805131030
- org.eclipse.uml2.common_1.4.0.v200805131030
- org.eclipse.uml2.diagram.activity_0.8.0.v200806112132
- org.eclipse.uml2.diagram.clazz_0.8.0.v200806112132
- org.eclipse.uml2.diagram.codegen.edit_0.8.0.v200806112132
- org.eclipse.uml2.diagram.codegen_0.8.0.v200806112132
- org.eclipse.uml2.diagram.common_0.8.0.v200806112132
- org.eclipse.uml2.diagram.component_0.8.0.v200806112132
- org.eclipse.uml2.diagram.csd_0.8.0.v200806112132
- org.eclipse.uml2.diagram.def_0.8.0.v200806112132
- org.eclipse.uml2.diagram.deploy_0.8.0.v200806112132
- org.eclipse.uml2.diagram.examples.clazz_0.8.0.v200806112132
- org.eclipse.uml2.diagram.examples.profile_0.8.0.v200806112132
- org.eclipse.uml2.diagram.parser_0.8.0.v200806112132
- org.eclipse.uml2.diagram.profile_0.8.0.v200806112132
- org.eclipse.uml2.diagram.source_0.8.0.v200806112132
- org.eclipse.uml2.diagram.statemachine_0.8.0.v200806112132
- org.eclipse.uml2.diagram.usecase_0.8.0.v200806112132
- org.eclipse.uml2.diagram_0.8.0.v200806112132
- org.eclipse.uml2.search.common.ui_0.7.0.v200806130939
- org.eclipse.uml2.search.ocl.ui_0.7.0.v200806130939
- org.eclipse.uml2.search.ocl_0.7.0.v200806130939
- org.eclipse.uml2.search.ui_0.7.0.v200806130939
- org.eclipse.uml2.search_0.7.0.v200806130939

- org.eclipse.uml2.uml.ecore.exporter_2.2.0.v200805131030
- org.eclipse.uml2.uml.ecore.importer_2.2.0.v200805131030
- org.eclipse.uml2.uml.edit_2.2.0.v200805131030
- org.eclipse.uml2.uml.editor_2.2.0.v200805131030
- org.eclipse.uml2.uml.resources_2.2.0.v200805131030
- org.eclipse.uml2.uml_2.2.0.v200805141133
- org.eclipse.uml2_2.1.0.v200805131030
- org.eclipse.uml2tools_0.8.0.v200806112132
- org.eclipse.xsd.cheatsheets_2.3.0.v200808251517
- org.eclipse.xsd.doc_2.4.0.v200808251517
- org.eclipse.xsd.ecore.converter_2.4.0.v200808251517
- org.eclipse.xsd.ecore.exporter_2.4.0.v200808251517
- org.eclipse.xsd.ecore.importer_2.4.0.v200808251517
- org.eclipse.xsd.edit_2.4.0.v200808251517
- org.eclipse.xsd.editor_2.4.0.v200808251517
- org.eclipse.xsd.example.installer_1.0.0.v200808251517
- org.eclipse.xsd.example_2.4.0.v200808251517
- org.eclipse.xsd.mapping.editor_2.4.0.v200808251517
- org.eclipse.xsd.mapping_2.4.0.v200808251517
- org.eclipse.xsd.source_2.4.1.v200808251517
- org.eclipse.xsd_2.4.1.v200808251517
- org.mortbay.jetty_5.1.14.v200806031611
- org.objectweb.asm_3.1.0.v200803061910
- org.sat4j.core_2.0.0.v20080602
- org.sat4j.pb_2.0.0.v20080602
- org.uddi4j_2.0.5.v200706111329
- org.w3c.css.sac_1.3.0.v200805290154
- org.w3c.dom.smil_1.0.0.v200806040011
- org.w3c.dom.svg_1.1.0.v200806040011

4 How to use the editor

This chapter covers the simple metatransition diagram editor.

After creating diagram with Eclipse's wizard you should see empty editor like on figure 3. Along with main window GMF is providing palette view, properties view and graphical outline view. Unfortunately GMF does not provide tree outline view, it has to be implemented manually (the one on figure is a basic implementation, without possibility to delete or drag and drop elements to diagram).

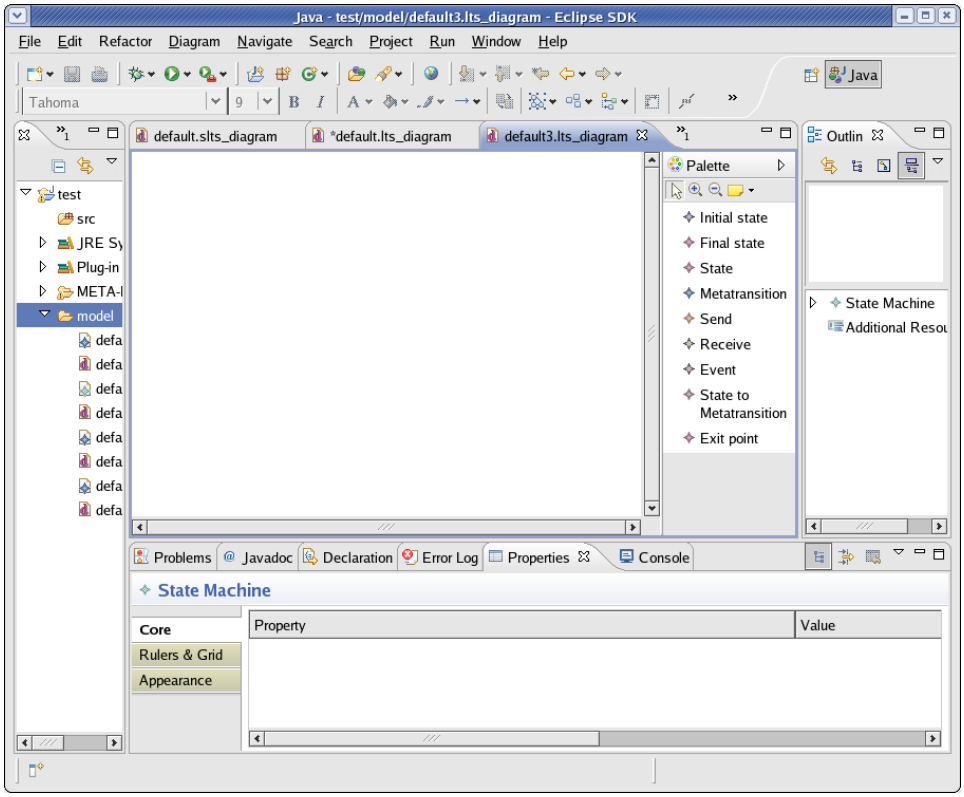

Figure 3: Empty diagram editor

4.1 Palette

You can use palette to create new elements with mouse or as an alternative it can be done with keyboard shortcut which activates palette tool (for example Ctrl+I for state). A simple example of state machine created with editor can be seen on figure 4

Palette consist of:

- Initial state

  The entry of state machine. There can be only one initial state. Graphical representation is taken from UML state diagram.

- Final state

  Exit from state machine. Final state is named, and there can be several

of them or none. Graphical representation is taken form UML state diagram.

- State

  Intermediate named state machine's state. Graphical representation is taken from UML state diagram.

- Metatransition

  Transition between states, which has to contain a statement.

- Send

  This is communication statement, the form of input on diagram editor is "channel_name ! message". It is also possible to edit it through properties view.

- Receive

  Same as above, except the format is "channel_name ? message".

- Event

  Statements indicating that internal action is taken while transiting to another state.

- State to metatransition

  Link between state and metatransition.

- Exit point

  Link between metatransition and state.

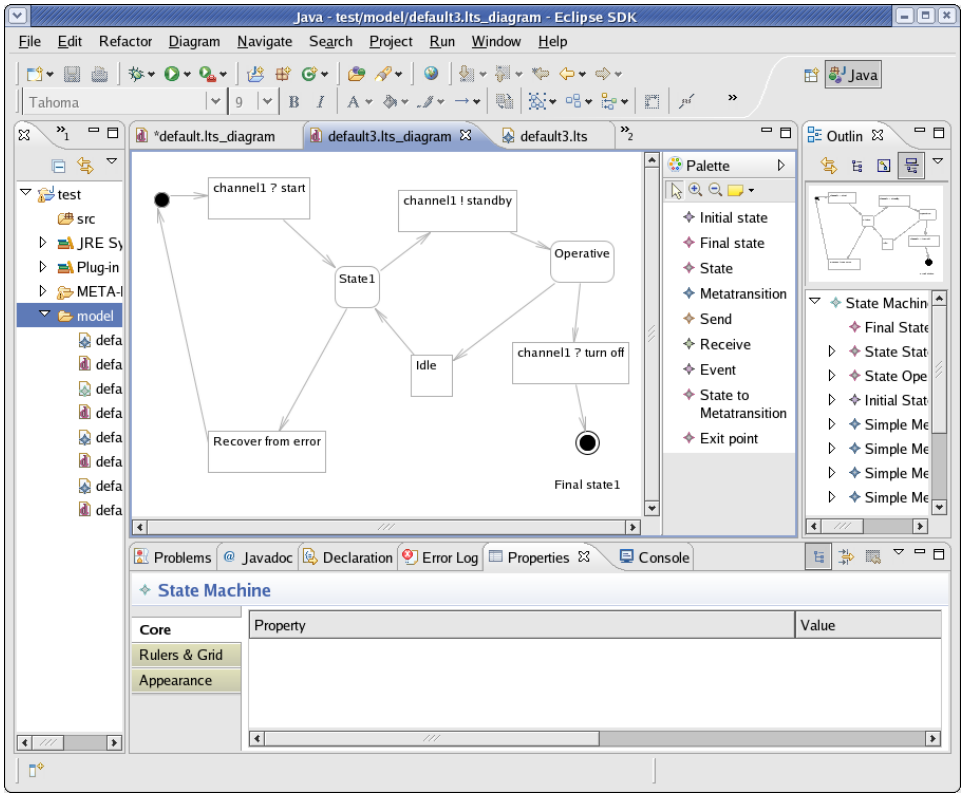

Figure 4: Example diagram

Another way of inserting state connected through metatransition with existing state is to use context menu of existing element and choose ,,Create metatransition with state". A state with name ,,State[n]" will appear as well as empty metatransition.

4.2 Copy and paste

Copying and pasting elements ought to be done by selecting them by mouse and dragging with ctrl key pressed. Unfortunately, ctrl+c and ctrl+v functionality is not working well for reasons explained in one of the next chapters [not ready yet].

4.3 Channels

To manage channels (which can referenced from communication statements, press ctrl+l. A window will be shown (see figure 5).

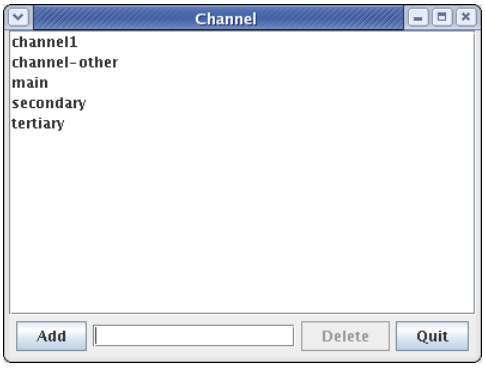

Figure 5: Channel managment window

This window allows to add and delete channels, which can be referenced by communication statements. Channel can be selected through properties view of communication element.

What is important, it is possible to share channels between diagrams by selecting ,,Load resource...” from context menu in EMF resource editor as shown on figure 6. By dint of it, a channel from ecore file can be referenced in communication element in another diagram. In ecore editor referenced file with its contents will be shown (figure 7).

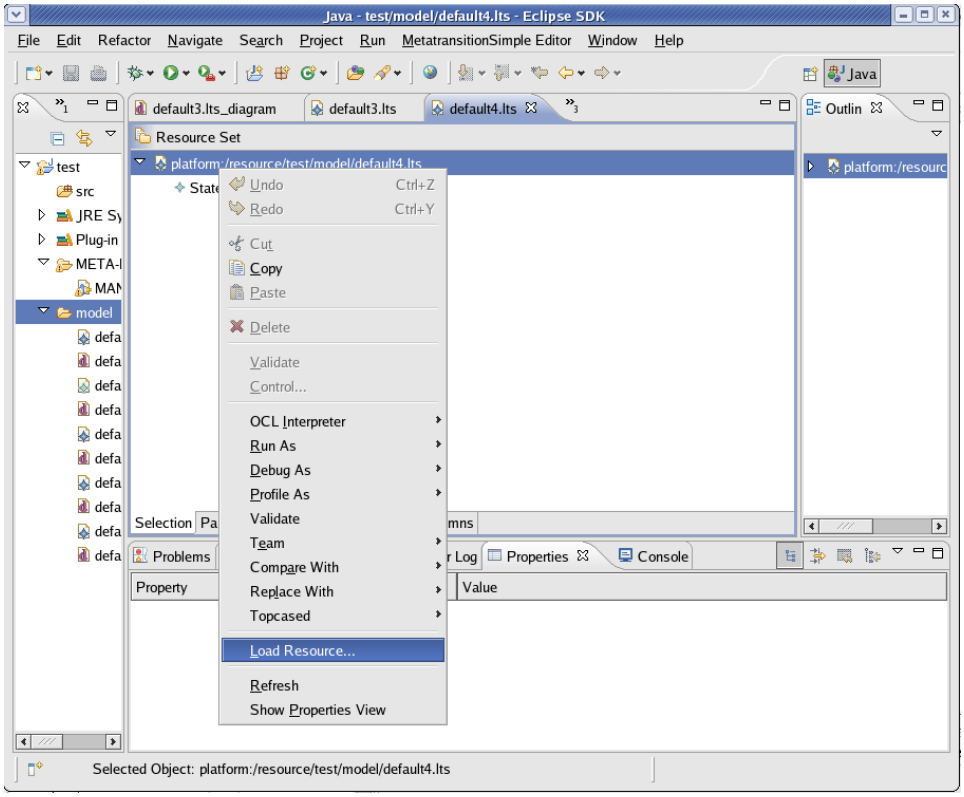

Figure 6: Loading resources (in this case channels) to another ecore file

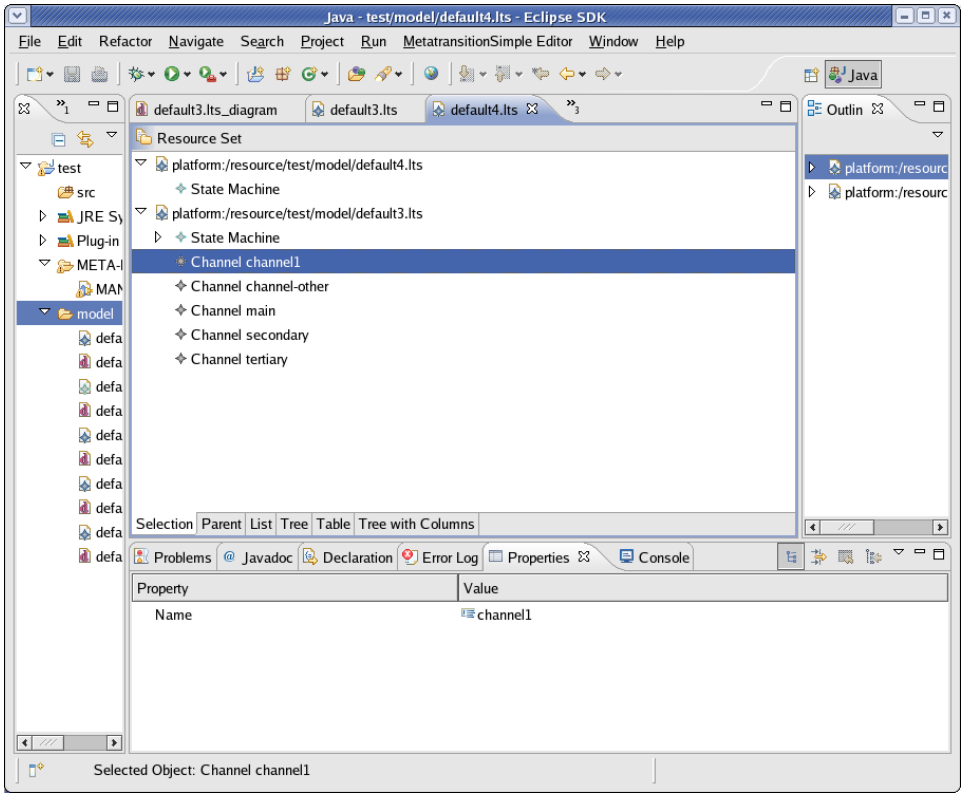

Figure 7: Now we can use channels from another ecore file

4.4 Validation

In order to validate the model, select ,,Validate" from menu ,,Diagram". If any constraint is violated, there will be message in Problems view (messages are not customized, this is one of the TODO's - see last chapter). You can see example of violated constraint on figure 8

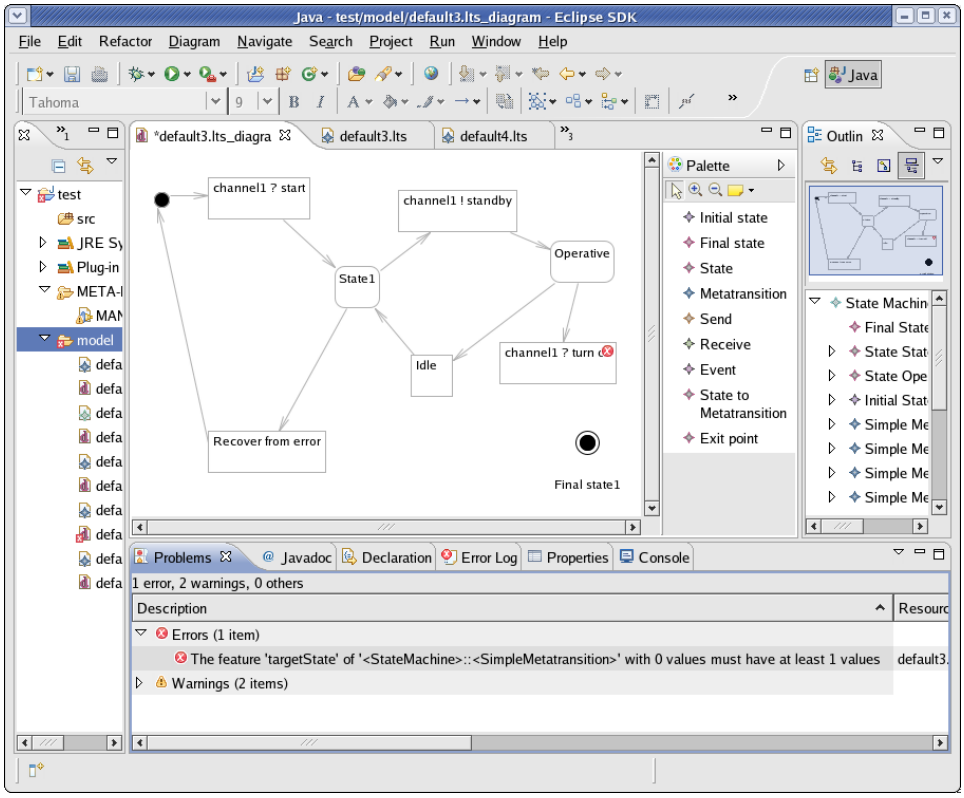

Figure 8: One constraint is violated

5 Eclipse plugins description

Project consists of 9 eclipse plugins, i.e.:

- `model.behavior.statemachine`

  This is where metamodel files are kept along with diagrams. There is also template directory, which can be used to keep JET files (JET is a templating scripting language which can be used to automatically modify code generated by EMF, there is tutorial available for it[5]). These JET scripts are not used right now, but it is possible to declare to use them in generator model files (*.genmodel). Scripts that are present in the directory (*.javajet) can be used to enable OCL constraint method stubs generation.

In `src` directory there is code generated by EMF - model interfaces and implementation classes.

- `model.behavior.statemachine.common.diagram`

  This is main development project. There are some common classes used by both editors, which are used by both diagram editors. There will be a detailed description of the classes in one of the following chapters.

- `model.behavior.statemachine.gmf`

  Project consists of GMF descriptors. It is worth mentioning that mapping descriptors use some resources from UML2Tools project (that is a part of Model Development Tools[6] . Generator models are modified after generation from mapping descriptors. These descriptors will de described in details in one of the following chapters.

Following 6 plugins contain main runtime code of editors. The `*.diagram` projects depend on `model.behavior.statemachine.common.diagram`

- `model.behavior.statemachine.simple.diagram`

  This is where generated code is kept. Packages that contain custom classes are named metatransitionSimple.diagram.custom.* . There are also some mofifications to original code as well as to plugin descriptors (plugin.xml and MANIFEST.MF). Methods that are changed in relation to original generated ones are annotated with `@generated NOT`. Changes and additions to plugin.xml descriptor are indicated by absence of `<?gmfgen generated="true"?>` line inside tags.

  Depends on model.behavior.statemachine, model.behavior.statemachine.simple.edit and model.behavior.statemachine.common.diagram.

- `model.behavior.statemachine.simple.edit`

  These are helper classes for EMF generated tree-editor. None of them are modified after generation.

- `model.behavior.statemachine.simple.editor`

  Generated EMF editor classes. No modifications there, although it would be good idea to modifiy in order to be abble to create multiple root elements in resource file (i.e. main state machine item along with channels, importance of this will be explained later).

These plugins are less developed than previous ones, but projects have more or less the same contents.

- `model.behavior.statemachine.structured.diagram`

  Structured metatransition diagram editor generated code. See `simple.diagram`.

- `model.behavior.statemachine.structured.edit`

  See `simple.edit`. No modified code.

- `model.behavior.statemachine.structured.editor`

  See `simple.editor`. No modified code.

6 Implementation

This chapter will cover the details of new and customized generated classes and how they affect behavior of application. It is considered to be a supplement for javadoc documentation.

6.1 model.behavior.statemachine

6.1.1 statemachine.ecore[diag]

Basic metamodel, which should be used along with another one implementing AbstractMetatransition class.

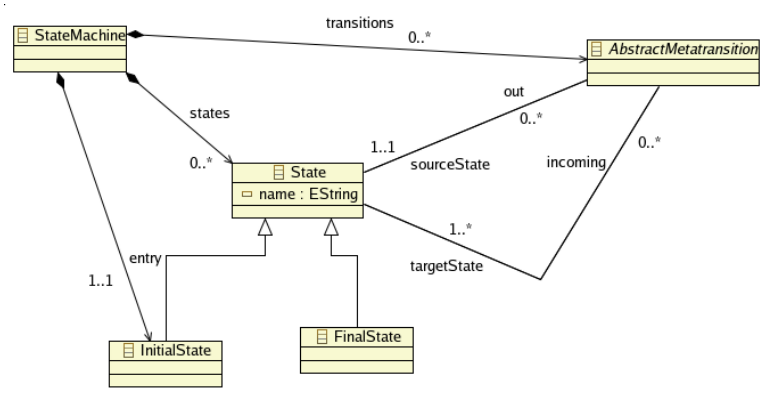

Figure 9: State machine metamodel

The original idea was to create template inrastructure for metamodels. That would mean using one file which references the basic metamodel for statemachine, and metatransition specialization metamodel. It is impossible to include one ecore metatransition file in another one, so after discussion it was decided to only keep reference to basic metamodel in concrete metatransition metamodel and use it as a base for further development of an diagram editor.

There are some constraints which cannot be implemented on model level, because they can differ between metatransition specializations requirements.

- There is one and only one initial state allowed in model.
- There can be any amount of final states.
- Metatransition should always have a target state. Whether it can have multiple targets it depends on concrete metatransition specialization.
- Additional constraints are specified on diagram editor level.

6.1.2 metatransitionSimple.ecore[diag]

Metamodel of simple metatransition which can include only one statement - event (inner action) described by text label or communication statement (dispatch or collection of message on previously defined channel).

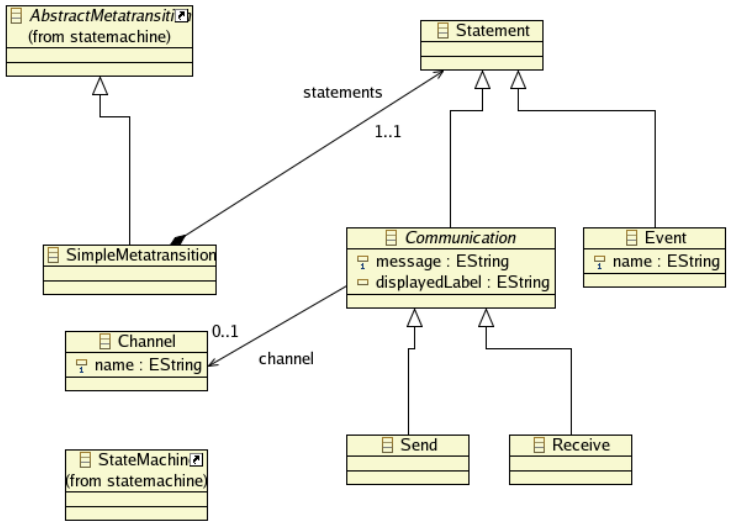

Figure 10: Simple metatransition metamodel

Communication class has two attributes: `message` and `displayLabel`. `message` is simply a name of message, while `displayLabel` is introduced to solve issue with display of contained elements (in this case channel). Communication classes are using overridden getter and setter for this attribute to display and alter custom label consisting of channel name, special symbol (question mark for collection and exclamation mark for dispatch of message) and message name. Setter methods for channel reference and message attribute are overridden in order to notify view after alteration (MVC design pattern convention).

6.1.3 metatransitionStructured.ecore[diag]

Structured metatransition is meant to contain list of statements similar to ones found in programming languages, so in general it acts as sequential code. Variables and channels are defined using special editor implemented in Swing (because of lack of possibility to handle multiple root elements in editors generated in EMF and GMF) - it will be described later in the chapter.

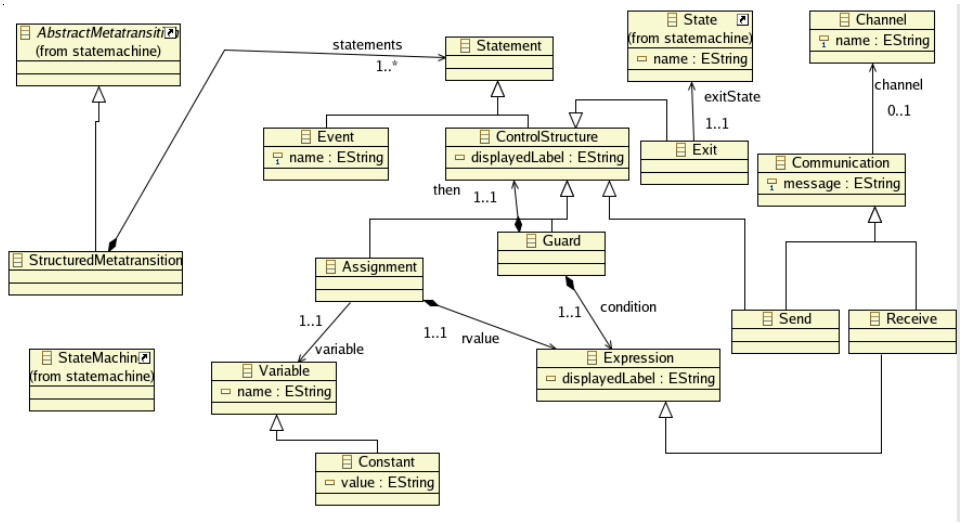

Figure 11: Structured metatransition metamodel

6.1.4 metatransition*.genmodel

Changes from original generated file:

- Property Type of displayedLabel property of all containing classes is changed from ,,Editable" to ,,None" in order to not to display this property in Properties view for Communication element.
- Edit and Editor plugins directory names (along with plugin IDs) changed to follow the pattern model.behavior.statemachine.{simple,structuredi}.{edit,editor} .

6.1.5 metatransitionSimple.common.ExtendedCommunicationImpl

This class extends CommunicationImpl which is default generated implementation of Communication. Overridden methods:

- `String getDisplayedLabel()`

  Builds label from referenced channel's name, delimiter character and message attribute. If channel is null, instead of it's name ,,noChannel" label is provided. If message name is empty, ,,noMessage" is provided.

- `void setDisplayedLabel(String)`

  Tokenizes label with respect to delimiter given in constructor. If tokenization fails, sets channel to null, and whole displayedLabel as message. Otherwise sets second token as message tries to identify channel. If it fails, sets channel to null.

- `void setChannel(Channel)`

  Extends original setter with notification execution that is supposed to be intercepted by handleNotificationEvent method in appropriate controller (*EditPart) class. :

- `void setMessage(String)`

  Works similar to above.

New constructor was added:

- `ExtendedCommunicationImpl(char)`

  Sets delimiter for tokenizing displayed label.

6.1.6 metatransitionSimple.impl.{SendImpl,ReceiveImpl}

These classes are inheriting from ExtendedCommunicationImpl (see chapter 6.1.5). The only thing that is different is delimiter character set in constructor. This is question mark for ReceiveImpl and exclamation mark for SendImpl. Other than that, these classes are nearly identical (minor difference - eStaticClass - is forced by EMF reflective API that needs to be implemented).

6.2 model.behavior.statemachine.gmf

This project consists of GMF-specific XML descriptors used to generate diagram editor code. All files are in `model` directory.

6.2.1 metatransitionSimple.gmfgraph

Graphical definition descriptor. Canvas - the root element consists of figure gallery, compartments, nodes and labels definitions.

Figure gallery contains figure descriptors, which describe graphical representation of them. The structure of description is hierarchical and it can include various elements like shape, layout, border, fonts, colors, size constraints.

Elements of gallery:

- `metatransitionFigure` - a simple rectangle.
- `finalStateFigure` - like in UML specification (black dot inside a circle).
- `finalStateName_external` - a workaround for external final state label.
- `arrowNameFigure` - currently unused, it can be used along with connection to label it.

An example of descriptor content (`finalStateFigure`):

```
<descriptors
    name="finalStateFigure">
  <actualFigure
      xsi:type="gmfgraph:Ellipse"
      name="ActivityFinalFigure">
    <layout
        xsi:type="gmfgraph:CustomLayout"
        qualifiedClassName="org.eclipse.uml2.diagram.common.draw2d.CenterLayout"/>
    <foregroundColor
        xsi:type="gmfgraph:ConstantColor"
        value="black"/>
    <maximumSize dx="23" dy="23"/>
    <minimumSize dx="23" dy="23"/>
    <preferredSize dx="23" dy="23"/>
    <children
        xsi:type="gmfgraph:Ellipse"
        name="ActivityFinalFigure_inner">
      <backgroundColor
          xsi:type="gmfgraph:ConstantColor"
          value="black"/>
      <maximumSize dx="15" dy="15"/>
      <minimumSize dx="15" dy="15"/>
      <preferredSize dx="15" dy="15"/>
    </children>
  </actualFigure>
</descriptors>
```

Figure descriptors used by nodes, diagram labels, compartments and (not in this case) connections definitions.

6.2.2 metatransitionSimple.gmftool

Palette definition descriptor. Nothing much here, just one palette with appropriate creation tools (described in chapter 4.1).

6.2.3 metatransitionSimple.gmfmap

Mapping descriptor. This is the main descriptor which binds the three models we have so far: the domain, the graphical definition, and the tooling definition. This is a key model to GMF development and will be used as input to a transformation step which will produce our final model, the generation model.

In ,,canvas mapping" element it is needed to specify domain model (package from ecore file) and root diagram element respectively (that is, the class which directly or indirectly contains all the classes that are supposed to be used in diagram). In our case StateMachine class is the root diagram element. Other than that aprropriate reference to palette and canvas ought to be set.

The first level of mapping are ,,top node reference" and ,,link mapping" elements. Every top node reference can specify containment feature - this is why state class as well as metatransition class are contained by root diagram element.

For node mapping that can be created as a child of top node reference, it is important to set domain element (from ecore metamodel), diagram node (from graphical definition descriptor) and tool (a creation tool from palette descriptor) properties.

The most complex node mapping is metatransition. It includes compartment which consists of three child references (statements): Event, Receive and Send. Event has featured label mapping which references name of event, communication elements are using displayedLabel attribute which was described before (chapter 6.1.5).

There are two link mappings: state to metatransition and metatransition to state. The second one has constraint defined - it can be created if and only if metatransition has not any outgoing links already created (see chapter 6.4.6).

Initial state mapping and links mapping are using canvas figures taken from UML2Tools project[6]. All descriptors that become handy at some point are stored in platform:/plugin/org.eclipse.uml2.diagram.def/.

6.2.4 metatransitionSimple.gmfgen

Generation model. To create it you have to use context menu and select ,,Create generation model", and choose aprropriate metamodel.

To generate editors In order to generate rcp application, you should select ,,RCP application" option in last wizard step.

Changes in comparison to standard generated file:

- Editor Generator

  Changed file extensions.
  Changed Validation Provider Priority to something else than lowest in order to enable validation.
  Changed Plugin ID to model.behavior.statemachine.simple.diagram .

6.2.5 metatransitionStructured.*

Work done on these descriptors is mostly equivalent to the above, except there is Guard child reference in metatransition structured compartment.

6.3 model.behavior.statemachine.common.diagram

6.3.1 statemachine.common.diagram.actions package

- `AbstractEnableTool`

  This is base class for enabling palette tool actions which can be used in cooperation with `org.eclipse.ui.bindings` Eclipse plugin extension point to create keyboard bindings. Unfortunately, there is no inversion of control in eclipse plugin system, therefore it is needed to create separate class for each tool.

  In order to be used as action in eclipse plugin, the class implements `IObjectActionDelegate` interface.

- `ActionUtils`

  Utility class which is a singleton and keeps editing domain reference and resource (model instance) reference used by Enable*Tool and BasicCustomListModel classes.

  Methods:

  - `void initialize(ISelection)`
    Initializes editing domain and resource variables on the basis of current selection.

- `TransactionalEditingDomain getDiagramEditDomain()` and `Resource getResource()`
  Getters for references.
- `void activateTool(String)`
  Sets active tool. Uses ToolRegistry to determine instance of appropriate creation tool.

6.3.2 statemachine.common.diagram.actions.enabletool package

This package contains concrete classes that are extending AbstractEnableTool.

6.3.3 statemachine.common.diagram.gui package

- `BasicCustomListModel`

  This is class extending AbstractListModel known from swing GUI framework. It manipulates on model instance and provides access to all elements of given EClass in model. Uses EMF command stack to edit model.In the projects it is used with Channel and Variable (structured metatransition diagram editor) elements.

  Implements ICustomListModel.

- `ICustomListModel`

  Interface providing additional methods to edit list of model elements.

  Extends javax.swing.ListModel from Java standard library.

- `{Create,Remove}EObjectCommand`

  These classes are wrappers to EMF commands and are used by BasicCustomListModel class.

- `ItemWindow`

  A simple swing window used to display and edit (add and delete) contents of list model imlementing ICustomListModel interface.

6.3.4 statemachine.common.diagram.handler package

- `CustomHandlerProvider`

  This is only used to store CustomHandlerProvider. It can be used by declaring appropriate global action handler in plugin descriptor (plugin.xml) of diagram editor projects.

- `UndoableOperationToGefCommandBridge`

  A bridge to GEF command stack. It will convert any IUndoableOperation to GEF command.

- `CustomCopyPasteHandler`

  Fork of DiagramGlobalActionHandler class. The `ICommand getCommand()` method is modified to return appropriate EMF copy and paste commands. Unfortunately there is bug there and copying and pasting do not work.

  There are some more methods to look at: `boolean can{Copy,Paste,Cut}`.

  There is another way to enable EMF clipboard in GMF diagram editor (coping objects as strings) which I have not tried[8].

6.3.5 statemachine.common.diagram.resource package

This package containts modified Resource and ResourceFactory classes with enabled UUID support to guarantee uniqueness of EMF object.

6.3.6 statemachine.common.diagram.util package

- `ToolsRegistry`

  A simple registry used to store ToolEntry references. Based on HashMap from standard library.

- `CreateCommandUtil`

  Utility class with static methods that are providing unique names for EMF objects.

6.4 model.behavior.statemachine.simple.diagram

6.4.1 metatransitionSimple.diagram.custom.action package

These classes are custom actions (IObjectActionDelegate implementations) that can be bound to some user interface element in plugin descriptor.

- `MetatransitionCreateAction`

  - `void selectionChange(IAction, ISelection)`
    Retrieves controller class for state.

- `void run(IAction)`

  The main method of class. It creates coumpound command from requests of creating metatransition and a link to it from selected state. Then it executes the command on GEF's command stack.

- `MetatransitionWithStateCreateAction`

  This class is partly similar to the one above. The difference is that in run method after executing command that creates metatransition connected to selected state another compound command is created and invoked. It consists of two request: to create a new basic state and a link to it.

6.4.2 metatransitionSimple.diagram.custom.gui package

Only one class there - ShowChannelWindow, which is an implementation of AbstractShowWindow abstract class. It creates and shows ItemWindow with BasicListModel customized with Channel EClass.

6.4.3 metatransitionSimple.diagram.custom.outline package

Class MetatransitionSimpleNavigator is basic implementation of EMF tree in outline view (unfortunately GMF itself does not provide that kind of view). Ecore tools package implementation is used - that should be the point of further research. You can check this view in action while editing metamodel diagrams in Eclipse.

The other class (TreeOutlinePage) is acting as a bridge to simple navigator implementation. It is used as adapter by diagram editor plugin (details: further part of chapter).

6.4.4 metatransitionSimple.diagram.edit.commands package

*CreateCommand classes are modified. The method doDefaultElementCreation is overriden to initialize name with CreateCommandUtil class from common.diagram project.

6.4.5 metatransitionSimple.diagram.part

MetatransitionSimpleDiagramEditor.getAdapter(Class) method modified in order to adapt customized outline view (TreeOutlinePage, see chapter 6.4.3).

Private static classes MetatransitionSimplePaletteFactory. {NodeToolEntry,LinkToolEntry} are modified to add self references to registry while constructing.

6.4.6 Constraints

- class metatransitionSimple.diagram.edit.policies. MetatransitionSimpleBaseItemSemanticEditPolicy

  Method canExistAbstractMetatransitionTargetState_3002 implements constraint that allows of creation links from metatransition to state only if metatransition hasn't got any outgoing links before.

6.4.7 Plugin descriptor (plugin.xml)

List of extension points that were added/edited:

- org.eclipse.gmf.runtime.common.ui.services.action.globalActionHandlerProviders

```
<GlobalActionHandlerProvider
      class="statemachine.common.diagram.handler.CustomHandlerProvider"
      id="MetatransitionSimpleRender">
   <Priority name="Lowest"/>
   <ViewId id="metatransitionSimple.diagram.part.MetatransitionSimpleDiagramEditorID">
      <ElementType class="org.eclipse.gmf.runtime.diagram.ui.editparts.IGraphicalEditPart">
         <GlobalActionId actionId="cut"/>
         <GlobalActionId actionId="copy"/>
         <GlobalActionId actionId="paste"/>
      </ElementType>
   </ViewId>
</GlobalActionHandlerProvider>
```

  This uses custom handler to provide customized cut/copy/paste commands.

- org.eclipse.emf.ecore.extension_parser

```
<extension point="org.eclipse.emf.ecore.extension_parser">
  <parser type="lts"
    class="statemachine.common.diagram.resource.CopyPasteEnabledResourceFactory" />
</extension>
```

  Custom resource factory instead of default.

- org.eclipse.ui.bindings

```
<extension point="org.eclipse.ui.bindings">
  <key commandId="metatransitionSimple.enableStateTool"
      sequence="M1+I" schemeId="org.eclipse.ui.defaultAcceleratorConfiguration"/>
  <!-- <key commandId="metatransitionSimple.enableMetatransitionTool"
      sequence="M1+I" schemeId="org.eclipse.ui.defaultAcceleratorConfiguration"/> -->
  <key commandId="metatransitionSimple.showChannelWindow"
      sequence="M1+L" schemeId="org.eclipse.ui.defaultAcceleratorConfiguration"/>
  <key commandId="metatransitionSimple.createMetatransition"
      sequence="M1+A" schemeId="org.eclipse.ui.defaultAcceleratorConfiguration"/>
  <key commandId="metatransitionSimple.createMetatransitionWithState"
      sequence="M1+4" schemeId="org.eclipse.ui.defaultAcceleratorConfiguration"/>
</extension>
```

Keyboard bindings for commands.

- org.eclipse.ui.commands

```
<extension point="org.eclipse.ui.commands">
  <category name="metatransitionSimple"
      description="Commands related to simple metatransition diagram editor"
      id="metatransitionSimple"/>
  <command categoryId="metatransitionSimple"
      description="Activates state creation tool"
      id="metatransitionSimple.enableStateTool" name="Enable state tool">
  </command>
  <command categoryId="metatransitionSimple"
      description="Shows channel window"
      id="metatransitionSimple.showChannelWindow"
      name="Show channels window">
  </command>
  <command categoryId="metatransitionSimple"
      description="Creates metatransition linked to selected state"
      id="metatransitionSimple.createMetatransition"
      name="Create metatransition">
  </command>
  <command categoryId="metatransitionSimple"
      description="Creates metatransition with output state linked to selected state"
      id="metatransitionSimple.createMetatransitionWithState"
      name="Create metatransition with state">
  </command>
</extension>
```

Custom commands (actions) definitions which can be used by menus, shortcuts, etc.

- org.eclipse.ui.popupMenus

```
<extension point="org.eclipse.ui.popupMenus">
  <objectContribution
      adaptable="false"
      id="metatransitonSimple.diagram.ui.objectContribution.StateEditPart1"
      objectClass="metatransitionSimple.diagram.edit.parts.StateEditPart">
  <action
      class="metatransitionSimple.diagram.custom.actions.MetatransitionCreateAction"
      definitionId="metatransitionSimple.createMetatransition"
      enablesFor="1"
      id="metatransitionSimple.createMetatransitionID"
      label="Create metatransition">
  </action>
  ...
```

Object contribution points for controller (EditPart) of metamodel element which is concerned. In action tag the main attribute is ,,class" which names a class implementing IObjectActionDelegate.

7 Possible integration with VCE component architecture editor

VCE component architecture editor is built on top of Topcased 1.x framework[7], which is a different solution (from GMF) to solve the same problem. In order There is a new 2.x update to Topcased, but it still uses the same code generation mechanism as 1.x. [to be completed]

8 Previous work

There was some effort done before to develop a more sophisticated metamodel. It is not used in current diagram editor implementation, because GMF and EMF tools are imposing some restrictions in metamodel usage. These are:

- There is only one root element allowed in model instance.
- All elements that are meant to be put in the diagram as nodes have to be contained by root element.
- Complicated referenced and contained elements initialization.
- One-to-one diagram node and creation tool binding.
- One-to-one diagram node and metamodel class binding.

Some of classes on metamodel diagrams are duplicating Pablo's architecture metamodel classes, but this can be easily merged[1].

8.1 Component behavior definition

The basic idea is to define components behavior with services. A service is an unit which consists of:

- Service methods with their own state machines and return values, which are bound to component's exposed interfaces.
- Local methods which are similar, but can be used only within service.
- Invocation queue - requests for method calls are coming into this queue. There can be several ways to get elements from it.
- Run Activity - a scheduler which distributes invocations of service methods. Has its own state logic.
- Variables - global and local.

8.2 Metamodel

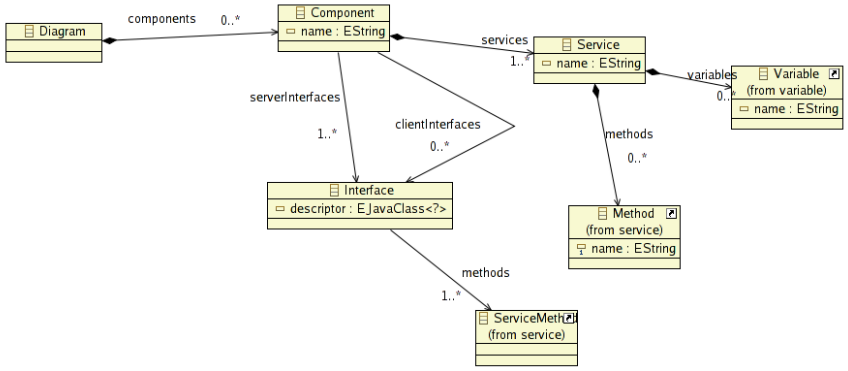

Figure 12: Root package

The root package should be possibly a merging point (with GCM architecture metamodel, see above).

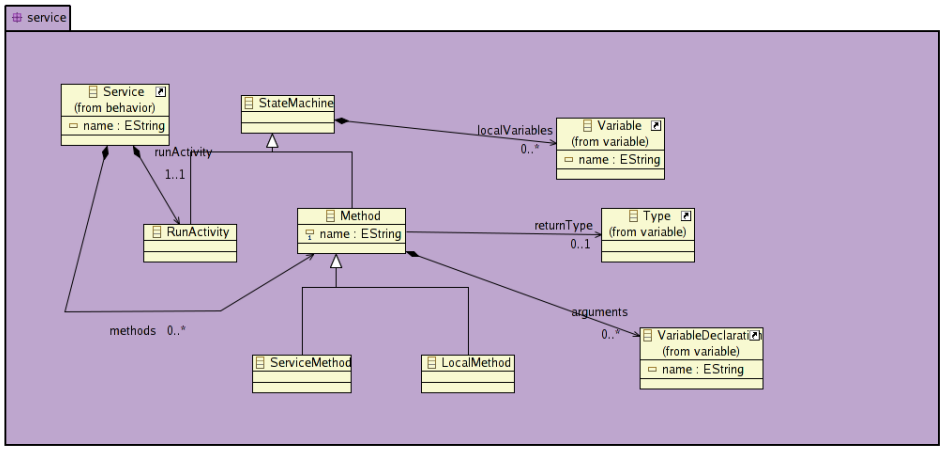

Figure 13: Service package

Every method has its logic defined by state machine. Every method can

have its own local variables.

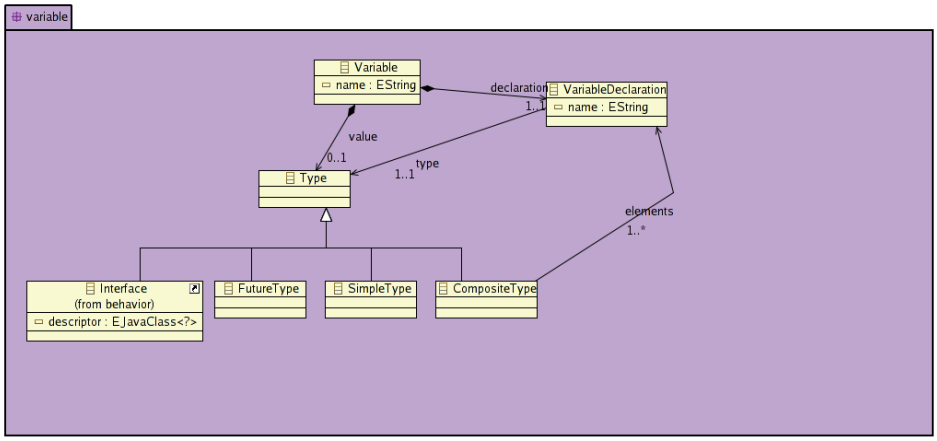

Figure 14: Variable package

Varaiables can be of simple or composite type. Inside composite type only some of contained variables may be relevant to considered model.

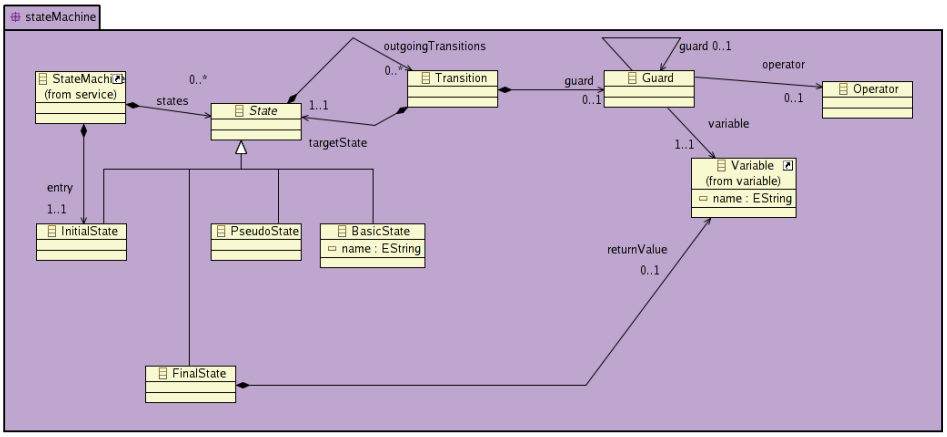

Figure 15: State machine package

State machine metamodel definition is not up to date on this metamodel diagram, although the idea of guard determining whether it is possible to do transition could be taken into consideration once again.

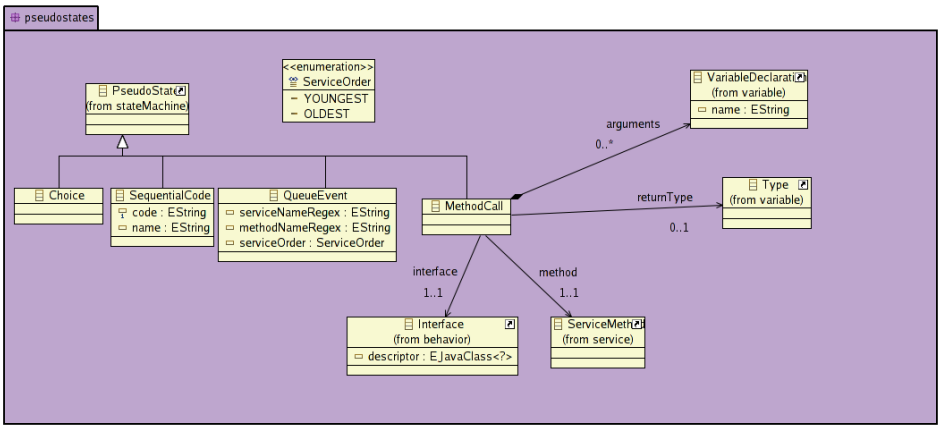

Figure 16: Pseudostates package

Roughly speaking, pseudostates are corresponding to structures introduced in structured metatransition metamodel.

There is also a simple project in repository with a germ of diagram of Component-Service-Methods structure editor implementation. It is made in pure declarative way (there are no changes to generated code - and, in fact, it is better to keep as much original generated code as possible). PUT NAME from repository HERE.

9 TODO list

- Fix copy and paste by ctrl+c ctrl+v (possibly by using EMF save to string and restore from string features).
- Complete validation implementation (customized messages - the place to start is metatransitionSimple.util.MetatransitionSimpleValidator.getResourceLocator() method in model.behavior.statemachine.simple.simple.diagram project).
- Complete structured metatransition implementation. This can be probably done the same way as extension of communication class6.1.5 by using custom implementation of ,,fake" attribute that can manipulate on references and contained objects. See metamodel.
- Set some reasonable keyboard shortcuts for tool activation and other actions.

- Extend existing implementation of tree outline view.
- Possibly (re)implement constraints in OCL language. There were some attemps to include OCL constraints interpretation into plugin code. In model.behavior.statemachine project's descriptor, try to use some variation of:

```
<extension
      point="org.eclipse.emf.validation.constraintProviders"
      id="oclProvider">
      <constraintProvider
      class="org.eclipse.emf.validation.examples.ocl.OCLConstraintProvider">
         <package namespaceUri="http://metatransitionSimple/1.0" />
         <ocl path="model/metatransitionSimple.ocl" />
      </constraintProvider>
</extension>
```

- Prepare RCP version of diagram editors. This was already done without too much effort, since GMF 2.1 provides mechanism to generate standalone RCP application code. The problem is that it cannot be asily integrated with project explorer view (see newsgroup posts concerning project navigator, especially [CommonNavigator][RCP][GMF]... ones). There are also some issues with validation in RCP mode.
- Complete plugin descriptor of diagram in order to include all necessary actions using implemented classes (see chapter 6.3).
- Integrate metamodel with Pablo's GCM architecture metamodel, so that for every component user can specify its behavior.

  Probably the best way to do it would be creating shared editing domain and separate plugins for different diagram editors. For details check [9].
- In more distant future - exporting behavior description to other formats (possibly XML-based as well).

  For this task, I would start with XML transformation tools like XSLT (see Apache Xalan project). The other option is to use Import/Export feature in similar way that Mikolaj Baranowski did.

References

[1] Future reference for Mikolaj's and Pablo's documentation.

[2] Homepage of EMF: http://www.eclipse.org/modeling/emf/

[3] Homepage of GMF: http://www.eclipse.org/modeling/gmf/

[4] Homepage of GEF: http://www.eclipse.org/gef/

[5] Tutorial on JET: http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html

[6] Homepage of MDT: http://www.eclipse.org/modeling/mdt/

[7] Homepage of Topcased: http://topcased.gforge.enseeiht.fr/

[8] http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.gmf.doc/tutorials/msl/clipboardTutorial.html

[9] http://wiki.eclipse.org/GMF_Tips#Sharing_single_EditingDomain_instance_across_several_diagrams