Kent Modelling Framework Version – Tutorial

www.ukc.ac.uk/kmf

Stuart Kent and Octavian Patrascoiu,
Computing Laboratory,
University of Kent, Canterbury, UK.

Draft, 6th December 2002

Contents

Chapter 1

About KMF

The *Kent Modelling Framework* (KMF) is being developed to provide a set of tools to support model driven software development. At the core of KMF is *ToolGen*, a tool to generate modelling tools from the definition of modelling languages expressed as metamodels. *ToolGen* is supported by OCL4Java, a Java library which allows dynamic evaluation of OCL constraints; and XMI, a java implementation of the XMI standard. Tools generated using ToolGen use OCL4Java to provide built in support for checking well-formedness of models, amongst other things; they use XMI to write and read models in XMI format. XMI is also used by ToolGen to read in metamodels in standard UML 1.3. XMI 1.0 format. As KMF develops, we will be making available specific modelling tools built with the help of ToolGen.

Our vision is a set of tools that generate quality modelling tools from language definitions. There are many measures of quality. We prioritize usability, reliability and capability. Our focus is on providing capabilities not provided by other tools, such as well-formedness checking, simulation of partial models, learning models from examples and two-way model transformation.

There is still much to do.

1.1 About ToolGen

ToolGen generates a modelling tool from a modelling language. The generated tool provides some basic functionality, driven through a GUI, including: exploring populations; creation, viewing and editing of populations; evaluation of OCL constraints derived from the input metamodel; evaluation of any OCL expression over particular objects; output of populations as XMI and HUTN (Human Readable Text Notation).

The generated tool can be customized by writing Java to the generated API.

1.2 About OCL4Java

OCL4Java is a Java library that supports dynamic parsing and evaluation of OCL expressions. That is, you can provide an OCL expression as a String argument to an OCL evaluation method, together with a reference to a (Java) object that is to provide the context for that evaluation, and the method will then parse and evaluate that expression, returning an object representing the results of the evaluation.

OCL4Java is used in tools generated by ToolGen to check constraints in the metamodel over populations in the Tool, and to support dynamic evaluation of OCL expressions entered by the user through the GUI.

1.3 About XMI

XMI is a Java library that implements the XMI standard. Currently it supports XMI 1.0, but future versions will be updated to support XMI 1.x and XMI 2.0.

XMI is used in tools generated by ToolGen to write and read models in XMI format. XMI is also used by ToolGen to read in metamodels in standard UML 1.3. XMI 1.0 format.

Chapter 2

ToolGen

2.1 Getting Started

We assume that you have installed KMF as explained in readme.txt, in the zip archive downloaded.

An example metamodel is provided in the *VSML* (Very Simple Modelling Language) subdirectory of the examples directory. Two xmi files are provided, one generated Rational Rose and one from the Poseidon UML tool from Gentleware (download a free version from `www.gentleware.com`). They are called `vsmlFromRose.xmi` and `vsmlFromPoseidon.xmi`, respectively. The Rose and Poseidon sources are in `vsml.mdl` and `vsml.zargo`, respectively. The xmi was extracted from Rose 2000 Enterprise Edition using the Unysis xmi add-in for Rose. `vsmlFromPoseidon.xmi` was taken directly out of `vsml.zargo`, which is actually just a .zip archive.

Invoke ToolGen as described in the readme.txt that comes with the installation, or in Section 2.3 on page 6 in this document.

A window will appear. Click on the *Choose input file* button and select one either of `.xmi` files from the `examples/vsml/src` folder. Click on the *Generate Code* button. You'll see some information appearing in the main body of the window. There should be no errors.

In the same directory as the `.xmi` files you will find a directory called vsml. This contains the generated java code. `Vsml` is the top level package. Compile the code using `examples/vsml/src` as the root source folder. This will ensure that you also include packages `startup` and `lifecycle` in the compilation. You will need to add the following files to the classpath: `lib/javaClass.jar`, `lib/OCL4Java.jar`, `lib/xmlParserAPIs.jar`, `xercesImpl.jar` and `lib/XMI.jar`.

If you use the Eclipse development environment (www.eclipse.org) then you can just import the project from examples/vsml – we have left the .classpath and .project files there for this purpose.

Once compiled, launch `startup.LittleLibrary` in Java, remembering to include the files identified above in the classpath.

A window will appear, with a model in VSML preloaded. The model is, of course, a simple class model for a library system. There are three panes in the Window. The left hand pane has two tabs: in the *Elements* tab you can browse model elements categorized by metaclass in the metamodel; the *Lifecycle* tab gives you access to methods to e.g. construct new instances of metaclasses – this is beyond the scope of *Getting Started*. The right-hand pane has four tabs: we'll only consider the *View* tab in this section. The bottom pane is a console, where errors and results of OCL constraint checking are reported.

Select an element in the explorer tree under the *Elements* tab. Goto the edit menu and select `View Properties` option. A window for viewing the properties of the selected element will appear under the views tab (make sure it is in focus) in the right hand pane. All should be reasonably obvious.

At the bottom of this window is an area for evaluation of OCL expressions in the context of the object being viewed. Type in an appropriate OCL expression (e.g. `self.name` or `self.X` for any `X` that is a property of the model element) in the area above the `Evaluate` button, and see the result appear in the area below.

You can check all the OCL constraints defined on the metamodel for the model stored in the tool. Select `Evaluate All` from the `Tools` menu and watch the results appear in the console pane. Or, if you prefer, select one or more model elements in the element explorer and choose the `Evaluate` option; only the constraints for the selected elements will be evaluated.

You may like to try making some changes to the metamodel in Poseidon, and regenerating the code. This may require changes to the code in `startup` and `lifecycle` packages, to reflect the changes in the metamodel. `startup.LittleLibrary` can also be modified to pre-load a different population when the generated tool is launched.

If you want to use the tool without preloading a model, then it can be launched using `Vsml.startup.implementation.VsmlStartup`. You can load `LittleLibrary.xmi` (in `examples/vsml`) using the menu option `File | Open`, to populate it with the example that is preloaded by `startup.LittleLibrary`, or you can populate it yourself using lifecycle operations: click on *Lifecycle* tab in left pane, select an operation for one or more of the metaclasses, choose `Edit | Build Element` and a window will appear under the *Lifecycle* tab in the right hand pane which will allow you to create new model elements.

A fuller explanation of the features of the generated tool is provided in Section 2.4.2 on page 6. These include: editing; invocation of lifecycle operations, such as creation of new model elements; saving and loading populations to/from xmi; saving populations to HUTN (Human Usable Text Notation) format.

2.2 Input Metamodel

ToolGen is used to generate a tool from a metamodel. ToolGen currently accepts the metamodel in UML 1.3 XMI 1.0 format. We have tested ToolGen using xmi extracted from the Poseidon and Rose tools, as explained in Section 2.1.

2.2.1 Class Diagram

The generator recognizes class diagrams as normal; association ends are mapped like attributes. In addition, constraints are generated from multiplicities, and a bidirectionality constraint is generated from two-way associations.

2.2.2 Constraints

Constraints can be represented in two ways. For tools that support them directly (like Poseidon) constraints are represented as constraints owned by classes, and are entered as strings using standard OCL syntax (though there are one or two features that are not supported). Query operations may be referred to in constraints, and themselves given a body in OCL syntax. This will be evaluated in the generated code. If you look in the xmi from Poseidon, you will see that the body is represented as the body of the method associated with the operation.

For tools that don't support them directly (like Rational Rose) constraints may be represented in the specification of an operation with the stereotype `<<inv>>`. Query operations can be represented as operations with the stereotype `<<spec>>`; the body of the operation should appear in the specification part.

2.3 Generating Code

If you are using a Windows platform, invoke ToolGen using startToolGen.bat in the `bin` directory. Otherwise invoke `uk.ac.ukc.cs.kmf.toolgen.Main` adding the following files to the classpath: javaClass.jar, OCL4Java.jar, ToolGen.jar, UMLModel.jar, xerces.jar, XMIFramework.jar, all in the `lib` directory.

Once launched you will be presented with a single window. You must provide an input file; and may optionally provide license information, which is a simple xml file, and/or a package root (e.g. uk.ac.ukc.cs.kmf) for the generated code. The license information is used to populate the *About* window in the generated tool. Click on the appropriate buttons to locate the input file and license information file,and fill in the package root. Then click on the `Generate Code` button.

2.4 Generated Tool

2.4.1 Code Structure

2.4.2 Using the Tool

2.4.3 Customizing the Tool

2.5 Known Issues

Chapter 3

OCL4Java

3.1 Using the Library

3.2 Known Issues