KDM-to-UML2 Converter (MoDisco Tool Box)

1. Presentation

This tool is about converting KDM models into UML2 models in order to allow integrating KDM-compliant tools (i.e. discoverers) with UML2-compliant tools (e.g. modelers, model transformation tools, code generators, etc). The converter is implemented by an ATL model-to-model transformation taking as input a model conforming to the KDM metamodel and producing as output a model conforming to the UML2 metamodel.

KDM is an OMG specification for architecture-driven modernization. It is a common intermediate representation for existing software systems (i.e. legacy systems) and their operating environments. It can be used to model source code accurately but it is designed to remain above the procedural level.

UML is the standardized specification of a graphical language for object modeling. It is a general-purpose modeling language.

Both specifications are available from the OMG website:

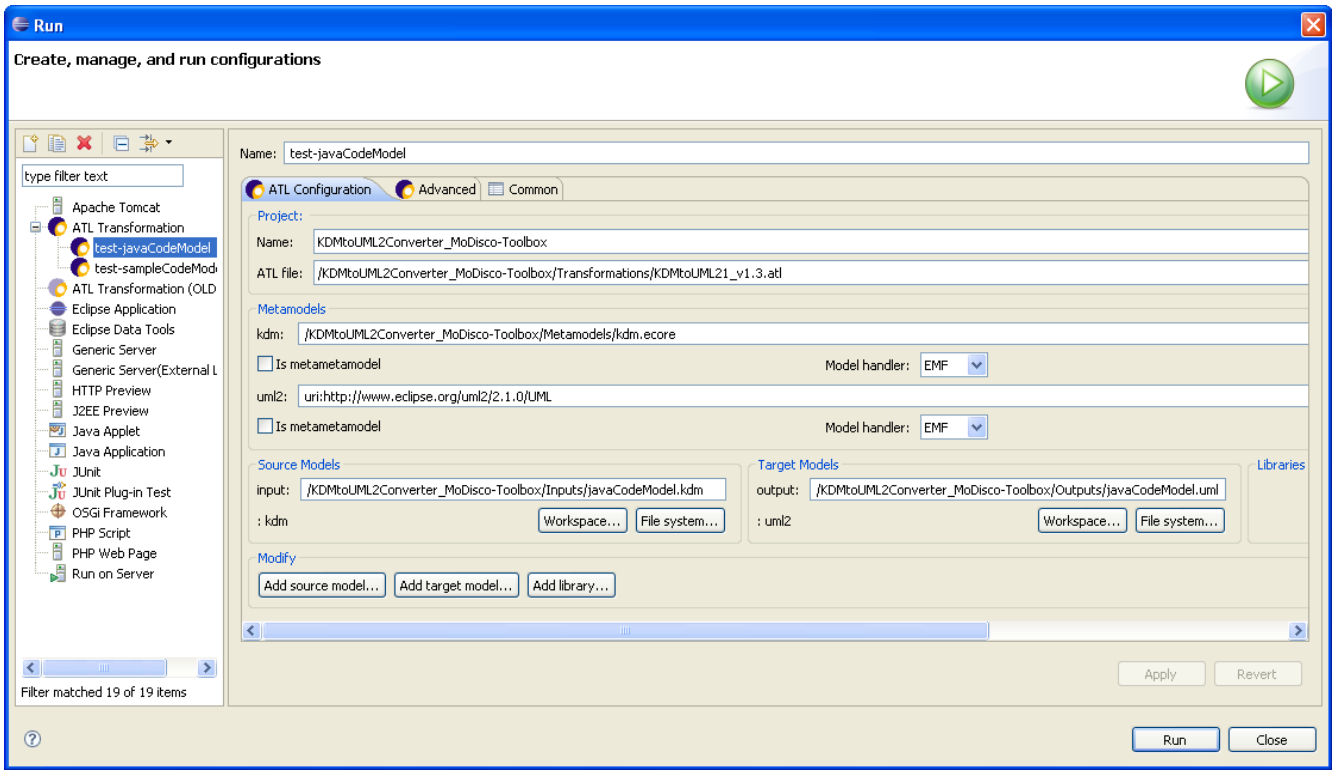
- KDM: http://www.omg.org/spec/KDM/1.0/
- UML 2.1: http://www.omg.org/spec/UML/2.1.2/

2. Installation

To use this converter under Eclipse (ATL plugins are required), simply follow these steps:

- Import the ATL project implementing the converter
- Use the existing launch configurations (provided tests) or create a new one and provide it with:
  - the ATL project and the given .atl file
  - the input metamodel: the given kdm.ecore file (select the EMF model handler)
  - the output metamodel: "*uri:http://www.eclipse.org/uml2/2.1.0/UML*" (also select the EMF model handler)
  - the source and target models

An example of an ATL launch configuration is show in the next screenshot:



3. *Structure of the transformation*

The transformation's rules properly create the structure of the UML model. All the top level packages are created in a single *Model* UML root element. Primitive types are also stored in a specific package in this root element. UML classes, interfaces, operations, inheritance relationships and parameters are retrieved from equivalent KDM model elements.

However KDM has been designed to build models as close to the source code as possible. Thus classes and interfaces may or may not be put in *CompilationUnit* KDM model elements: the transformation manages both cases.

Attributes and associations are specifically matched from the *MemberUnit* KDM model elements which are stored within classes or interfaces in the KDM model. The transformation then has to determine whether an *Association* UML model element is required or not. Because of the code-close architecture of KDM, there are some limitations in the transformation when it comes to *Association* UML model element matching.

4. Limitations

The transformation is unable to locate the bidirectional association between entities. For instance if we consider several links between two same entities in the KDM model, it is not possible to determine the corresponding UML associations that should be bidirectional. Thus only unidirectional *Associations* are created and two unidirectional *Associations* are created to represent a bidirectional one when needed.

There are 2 helpers which are defined in the transformation but currently not used:

- *isBidirectional(mu:kdm!MemberUnit):Boolean*
- *areTied(mu1:kdm!MemberUnit, mu2:kdm!MemberUnit):Boolean*

If you want to extend the transformation in order to determine the bidirectional associations, these helpers may be useful.

The transformation is also unable to proceed when there is a not-existing (or not-specified) type in a *MemberUnit* KDM model element. This is because it must create a *Property* UML model element in the class (or interface) specified as the equivalent type of the *MemberUnit* KDM model element.