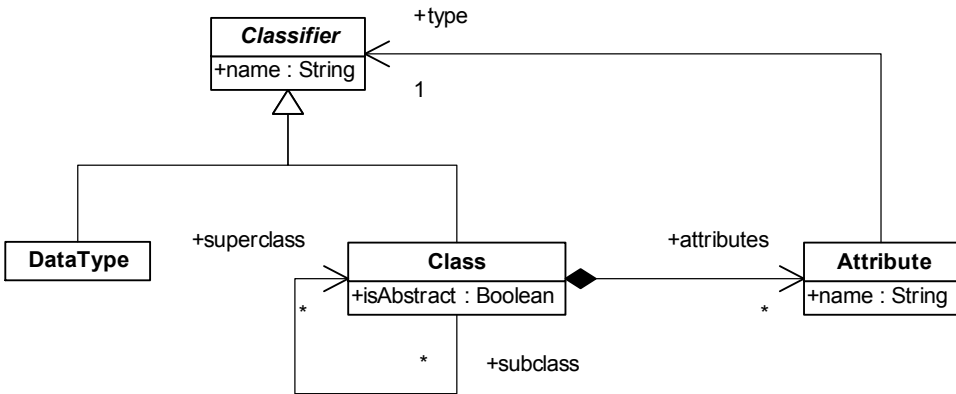


Context of this work



- The present courseware has been elaborated in the context of the MODELWARE European IST FP6 project (<http://www.modelware-ist.org/>).
- Co-funded by the European Commission, the MODELWARE project involves 19 partners from 8 European countries. MODELWARE aims to improve software productivity by capitalizing on techniques known as Model-Driven Development (MDD).
- To achieve the goal of large-scale adoption of these MDD techniques, MODELWARE promotes the idea of a collaborative development of courseware dedicated to this domain.
- The MDD courseware provided here with the status of open source software is produced under the EPL 1.0 license.

1. Description of the Exercise

Short Name: UMLClassFlattening
Full Name: Removing UML Inheritance Hierarchies in Class Diagrams
Short Description: This exercise requires writing a transformation program that takes an UML class diagram as an input and generates a new diagram in which only the leaves of the inheritance hierarchies are kept. Inheritance hierarchies in the source model are flattened by moving all the attributes of superclasses to the leaf classes. Before flattening a check for the presence of cycles in the inheritance graph is performed. The result is a report that contains information about eventual cycles. This report takes the form of a model that conforms to a metamodel describing defects.
Tools: The required transformation may be implemented in any model transformation language. The description of the exercise in this document assumes that the metamodels will be expressed in KM3 [2] and ATL [3] will be used for writing the transformation program. ATL engine can be downloaded from the GMT web site [3].
<p>Source Metamodels:</p> <p>The source metamodel is a simplified version of the UML metamodel standardized by OMG [1]. It is shown in Figure 1.</p>  <pre> classDiagram class Classifier { +name : String } class DataType class Class { +isAbstract : Boolean } class Attribute { +name : String } Classifier < -- DataType Classifier < -- Class Classifier "1" --> "1" Attribute : +type Class "*" --> "*" Class : +subclass Class "*" *-- "*" Attribute : +attributes Class --> Classifier : +superclass </pre> <p>The diagram shows a metamodel with four classes: Classifier, DataType, Class, and Attribute. Classifier is the base class, with DataType and Class as subclasses. Classifier has an attribute +name : String. Class has an attribute +isAbstract : Boolean. Attribute has an attribute +name : String. There are three relationships: 1 Classifier to 1 Attribute (association named +type), Class to Class (self-association named +subclass), and Class to Attribute (aggregation named +attributes). Additionally, there is a directed association from Class to Classifier named +superclass.</p>
Figure 1 Simplified UML metamodel

Target Metamodels:

The result of the cycles check is a model that conforms to the following metamodel.

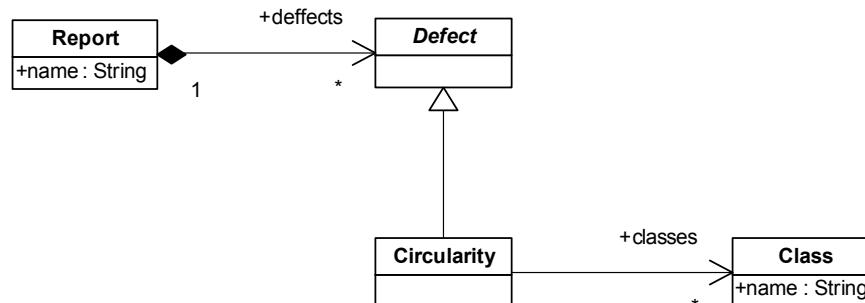


Figure 2 Metamodel of defects report

Instances of this metamodel contain reports about defects found in a given UML model. Currently only one defect is supported: presence of cycles in inheritance hierarchies. For every detected cycle an ordered list of the classes that form the cycle is given.

Tasks:

1. Write a transformation program that checks an input UML model for cycles in the inheritance hierarchies. The program produces an output model conforming to the metamodel in Figure 2. For every detected cycle an instance of class Circularity is created. It refers to the classes that form the cycle. If no cycles are found the report is empty (i.e. does not contain defects).
2. Write a transformation program that transforms an input UML model to a new UML model without inheritance relations. For this purpose:
 - Keep only the leaf classes in the output model.
 - Include the attributes inherited from the superclasses in the subclasses. This rule is applied recursively through the inheritance hierarchy.
 - Apply the rule that the attributes in a subclass override the attributes with same name in the superclass.
3. Test the first transformation program (Task 1) by feeding it with a model that contains cycles.
4. Test the second transformation program (Task 2) with a sample UML model.

Guidelines: For Task 1 use the idea from the classical algorithm for topological sort over graphs.

References

1. OMG Unified Modeling Language (UML), version 1.4 (formal/03-03-01), 2002, <http://www.omg.org/technology/documents/formal/uml.htm>
2. KM3 User Manual. <http://eclipse.org/gmt>
3. The ATL Development Tools. <http://eclipse.org/gmt>