	ATL Transformation Example	Author Éric Vépa evepa@sodius.com
	KM3 to Measure	August 30th , 2007

1. ATL Transformation Example: KM3 to Measure

The KM3 to Measure example describes measurement on KM3 meta-models, using metrics defined as ATL libraries.

1.1. Transformation Overview

The aim of this transformation is to collect measurement data on KM3 meta-models.

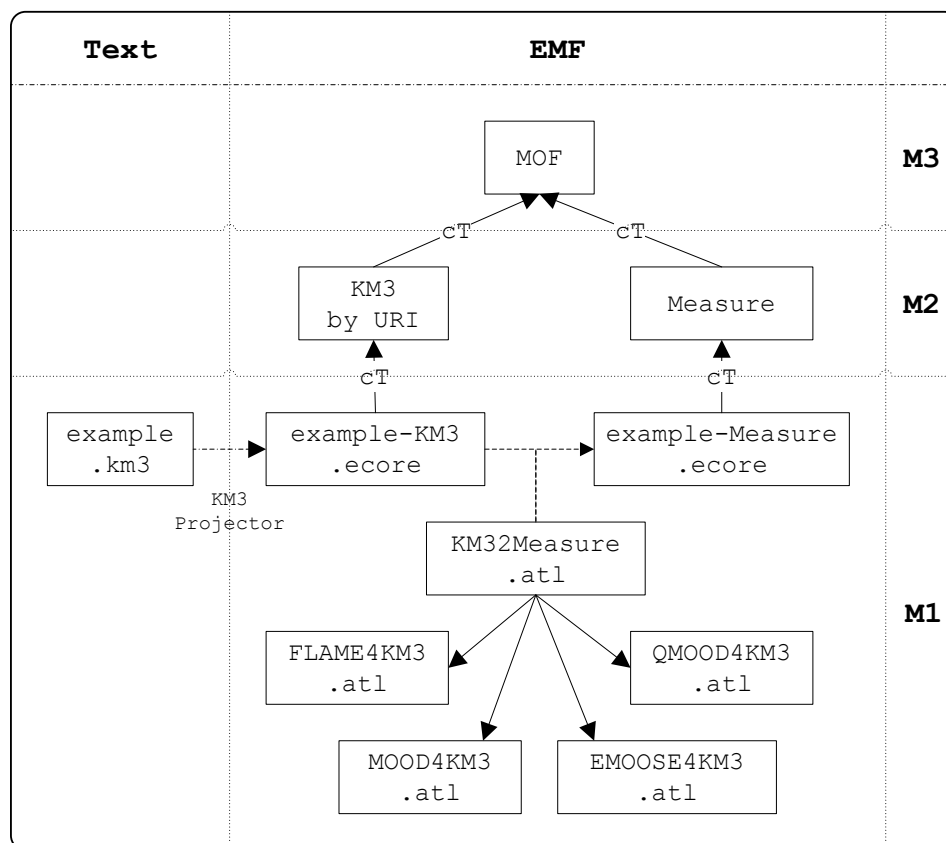


Figure 1: Overview of the transformation

KM3 meta-models can be measured with ATL transformations. A meta-model is selected from the zoo [3] in KM3 format, then injected as a KM3 model (with the predefined injector) and used as the input model of the transformation. The transformation input and output meta-model handlers are KM3 and Measure. The run of the transformation *KM32Measure* produces a collection of measurement data.

We obtain an output model of measures (which keeps the hierarchy of the meta-model). The metrics used in the transformation are implemented with ATL libraries and will be explained in an upcoming section.

2. Meta-models

2.1. KM3

The Kernel MetaMeta-model (KM3) meta-model is available in [2].

2.2. Measure

The Measure meta-model is used to stored the data collected after a model measurement.

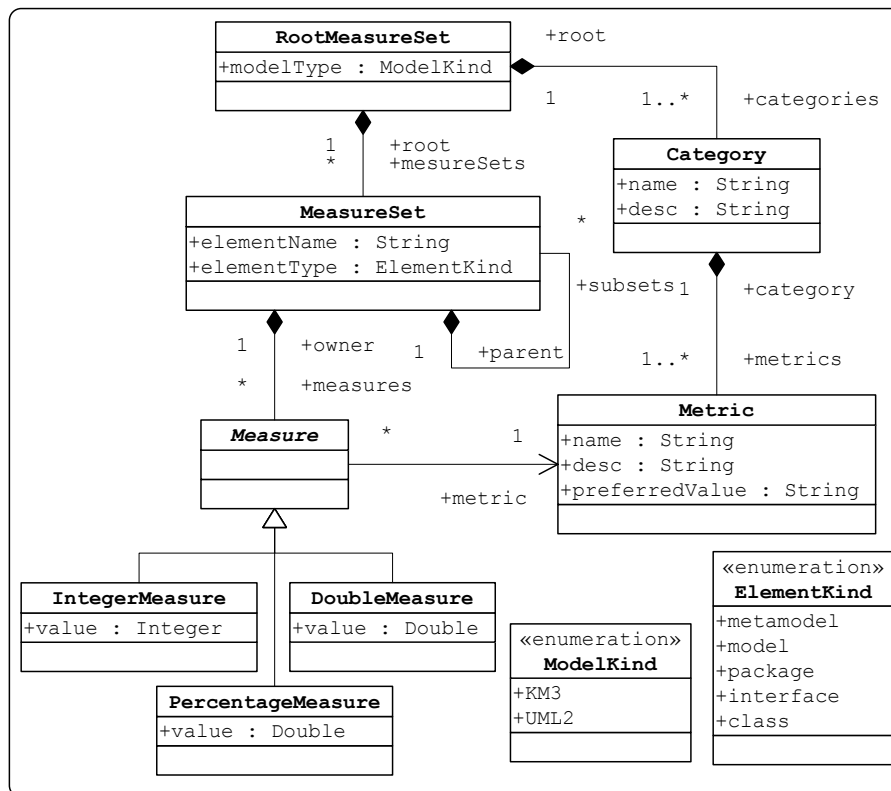




Figure 2: Measure meta-model

A measure model is in the following way made up: the root is a set of measure (*RootMeasureSet*) which contains information on the type of measured model (*modelType* among *KM3* or *UML2*), a set of categories of metric and sets of measure for each model element measured. A category (*Category*) corresponds to a metric set with a *name* and a description (*desc*) (an acronym and its definition). A category gathers one or more metric (*Metric*) also defined with a *name* and a description. A default predicate is also associated (*preferredValue*), it is the desired values for the metric (for example $\neq 0$ or > 75). A set of measure (*MeasureSet*) described measurements performed on a model element (*elementName*) of a given type (*elementType* among *meta-model*, *model*, *package*, *interface*

 	ATL Transformation Example	Author Éric Vépa evepa@sodius.com
	KM3 to Measure	August 30th , 2007

or *class*). The set of measure are structured between them, for example a set of measure on a package will contain the set of measure of the classes that this package contains. A measure (*Measure*) is associated to a metric and is declined in several versions. Measures with an integer, real or percentage value (respectively *IntegerMesasure*, *DoubleMeasure* and *PercentageMeasure*).

3. Transformation from KM3 to Measure



3.1. Rules specification

These are the rules to collect measurement data from a KM3 model to a Measure model.

- For the whole model, the following elements are created:
 - A *RootMeasureSet* element is created with:
 - A type of model measured (*modelType* set to #KM3).
 - For each category implemented, the following elements are created:
 - A *Category* element with :
 - A name and a description.
 - The created *Category* element is linked to the *RootMeasureSet*.
 - For each metric implemented for a category, the following elements are created:
 - A *Metric* element with:
 - A name and a description.
 - A mandatory preferred value.
 - The created *Metric* element is linked to a *Category* element.

The measure level determinates the metrics and categories that are registered.

- For each *Package* element, the following elements are created:
 - A *MeasureSet* element with the name and the type of the *Package* element measured.
 - The created *MeasureSet* element is linked to the *MeasureSet* created for his owner *Package* element.
 - If the *Package* element contains *Class* elements, the following elements are created:

 	ATL Transformation Example	Author Éric Vépa evepa@sodius.com
	KM3 to Measure	August 30th , 2007

- An *IntegerMeasure*, *DoubleMeasure* or *PercentageMeasure* element, for each *Metric* element created and defined for package level.
 - If the *Package* element is a root package :
 - It is linked to the *RootMeasureSet*.
- For each *Class* element, the following elements are created:
 - A *MeasureSet* element with the name and the type of the *Class* element measured.
 - The created *MeasureSet* element is linked to the *MeasureSet* created for his owner *Package* or *Class* (nested classifier) element.
 - An *IntegerMeasure*, *DoubleMeasure* or *PercentageMeasure* element, for each *Metric* element created and defined for class level.

3.2. ATL code

This ATL code for the *KM32Measure* transformation consists in 4 helpers and 9 rules.

The transformation uses the metrics libraries defined in section 4.

The attribute helper *measureLevel* is used to define the type of model elements measured. For example, at package level (*#package*), only metrics defined for packages will be used. At class level (*#class*), both packages and classes metrics will be used.

The two maps *CategoryByName* and *MetricByName* are used to register the categories of metrics and the metrics implemented.


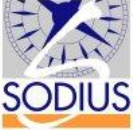
The entrypoint rule *Metrics* is used to fill the two previous maps, before processing measures. The metrics and categories registered depend on the measure level.

The rule *Package2MeasureSet* is called if the package or class level is enabled. If the package contains some classes, measures will be performed for the metrics defined for package level.

The rule *Class2MeasureSet* is called if the class level is enabled. Measures are performed for each metrics defined for class level.

The called rules *Category*, *Metric* and *MetricWithPreferredValue* are used in the entrypoint rule to register the implemented categories and metrics with mandatory preferred value.

The called rules *IntegerMeasure*, *DoubleMeasure* and *PercentageMeasure* store the value for a metric given.

 	ATL Transformation Example	Author Éric Vépa evepa@sodius.com
	KM3 to Measure	August 30th , 2007

4. Metrics Libraries

4.1. FLAME for KM3 Library

4.1.1. FLAME (Formal Library for Aiding Metrics Extraction)

The functions of this library are defined in OCL language in [4] and [5] for the UML 1.3 meta-model. They have been adapted to fit with the KM3 meta-model. The functions that deal with operation, overridden feature and visibility feature have been omitted. Additional functions on references have been implemented.

Attributes have been mapped as KM3 attributes and KM3 containment references.

Additional helpers on KM3 references (non containment) have been implemented.



4.1.2. ATL code

This ATL code for the *FLAME4KM3* library consists in 52 helpers.

4.2. MOOD for KM3 Library

4.2.1. MOOD (Metrics for Object-Oriented Design) and MOOD2

Name	MOOD::AIF - Attributes Inheritance Factor
Informal definition	Quotient between the number of inherited attributes in all classes of the package and the number of available attributes (locally defined plus inherited) for all classes of the current package.
Name	MOOD::RIF - References Inheritance Factor
Informal definition	Quotient between the number of inherited references in all classes of the package and the number of available references (locally defined plus inherited) for all classes of the current package.
Name	MOOD::CCF - Class Coupling Factor
Informal definition	Quotient between the actual number of coupled class-pairs within the package and the maximum possible number of class-pair couplings in the package. This coupling is the one not imputable to inheritance.
Name	MOOD::ICF - Internal Coupling Factor
Informal definition	Quotient between the number of coupling links where both the client and supplier classes belong to the current package and the total number of coupling links originating in the current package.
Name	MOOD2::IIF - Internal Inheritance Factor
Informal definition	Quotient between the number of inheritance links where both the base and derived classes belong to the current package and the total number of inheritance links originating in the current package.

 	ATL Transformation Example	Author Éric Vépa evepa@sodius.com
	KM3 to Measure	August 30th , 2007

4.2.2. ATL code

This ATL code for the *MOOD4KM3* library consists in 5 helpers.

The implemented metrics from the MOOD and MOOD2 sets only depend on the FLAME functions and are list above. These metrics are defined for package level.

4.3. EMOOSE for KM3 Library



4.3.1. MOOSE (Metrics for Object-Oriented Software Engineering) and EMOOSE (Extended MOOSE)

Name	MOOSE::WRC - Weighted References per Class
Informal definition	The sum of complexities of the references in the current class. If all reference complexities are considered to be unique, WRC is equal to the number of references. The authors do not propose any algorithm for calculating the complexities of references. As such, in the formalization, the complexities were considered unitary.
Name	MOOSE::DIT - Depth of Inheritance Tree
Informal definition	The length of the longest path of inheritance from the current class to the root of the tree.
Name	MOOSE::NOC - Number Of Children
Informal definition	The number of classes that inherit directly from the current class.
Name	MOOSE::CBO - Coupling Between Objects
Informal definition	The number of other classes that are coupled to the current one. Two classes are coupled when references declared in one class use references or instance variables defined by the other class. Or used as a type or in reference by other classes.
Name	EMOOSE::SIZE2
Informal definition	Number of local attributes and references defined in the class. The metric SIZE 1 is code dependant so not adapted to our problem.

4.3.2. ATL code

This ATL code for the *EMOOSE4KM3* library consists in 5 helpers.



The implemented metrics from the MOOSE and EMOOSE sets only depend on the FLAME functions and are list above. These metrics are defined for class level.

 	ATL Transformation Example	Author Éric Vépa evepa@sodius.com
	KM3 to Measure	August 30th , 2007

4.4. QMOOD for KM3 Library

4.4.1. QMOOD (Quality Model for Object-Oriented Design)

Name	QMOOD::DSC - Design Size in Classes
Informal definition	Count of the total number of classes in the design.
Name	QMOOD::NOH - Number of Hierarchies
Informal definition	Count of the number of class hierarchies in the design.
Name	QMOOD::NIC - Number of Independent Classes
Informal definition	Count of the number of Classes that are not inherited by any Class in the design.
Name	QMOOD::NSI - Number of Single Inheritance
Informal definition	Number of Classes (sub classes) that use inheritance in the design.
Name	QMOOD::NMI - Number of Multiple Inheritance
Informal definition	Count of the number of instances of multiple inheritance in the design.
Name	QMOOD::NNC - Number of Internal Classes
Informal definition	Count of the number of internal classes defined for creating generalization-specialization structures in class hierarchies of the design.
Name	QMOOD::NAC - Number of Abstract Classes
Informal definition	Count of the number of classes that have been defined purely for organizing information in the design.
Name	QMOOD::NLC - Number of Leaf Classes
Informal definition	Count of the number of leaf classes in the hierarchies of the design.
Name	QMOOD::ADI - Average Depth of Inheritance
Informal definition	The average depth of inheritance of classes in the design. It is computed by dividing the summation of maximum path lengths to all classes by the number of classes. The path length for a class is the number of edges from the root to the class in an inheritance tree representation.
Name	QMOOD::AWI - Average Width of Inheritance
Informal definition	The average number of children per class in the design. The metric is computed by dividing the summation of the number of children over all classes by the number of classes in the design
Name	QMOOD::ANA - Average Number of Ancestors
Informal definition	The average number of classes from which a class inherits information.
Name	QMOOD::MAA - Measure of Attribute Abstraction
Informal definition	The ratio of the number of attributes inherited by a class to the total number of attributes in the class.
Name	QMOOD::MRA - Measure of Reference Abstraction
Informal definition	The ratio of the number of references inherited by a class to the total number of references in the class.
Name	QMOOD::MOA - Measure of Aggregation
Informal definition	Count of the number of data declarations whose types are user defined


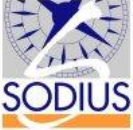
 	ATL Transformation Example	Author Éric Vépa evepa@sodius.com
	KM3 to Measure	August 30th , 2007

	classes.
Name	QMOOD::MRM - Modeled Relationship Measure
Informal definition	Measure of the total number of attribute and parameter based relationships in a class.
Name	QMOOD::NOA - Number of Ancestors
Informal definition	Counts the number of distinct classes which a class inherits.
Name	QMOOD::NOR - Number of References
Informal definition	Count of all the references defined in a class.
Name	QMOOD::NOD - Number of Attributes
Informal definition	Number of attributes in the class.
Name	QMOOD::NAD - Number of Abstract Data Types
Informal definition	Number of user defined objects used as attributes in the class and which are necessary to instantiate an object instance of the (aggregate) class.
Name	QMOOD::CSM - Class Size Metric
Informal definition	Sum of the number of references and attributes in the class.
Name	QMOOD::DCC - Direct Class Coupling
Informal definition	Count of the different number of classes that a class is directly related to. The metric includes classes that are directly related by attribute declarations and references.
Name	QMOOD::MCC - Maximum Class Coupling
Informal definition	This metric not only includes classes that are directly related to a class by attributes and references, but also classes that are indirectly related through the directly related classes.
Name	QMOOD::DAC - Direct Attribute Base Coupling
Informal definition	This metric is a direct count of the number of different class types that are declared as attribute references inside a class.
Name	QMOOD::DRC - Direct Reference Base Coupling
Informal definition	This metric is a direct count of the number of different class types that are declared as references inside a class.
Name	QMOOD::CCD - Class Complexity Based on Data
Informal definition	Computes complexity based upon the number of components (attributes) that are defined in the class. All component declarations are resolved to the basic primitives (integers, doubles and characters). The metric value is a count of the number of primitives.

4.4.2. ATL code

This ATL code for the *QMOOD4KM3* library consists in 25 helpers.

The implemented metrics from the QMOOD set only depends on the FLAME functions and are list above. These metrics are defined both for package and class levels.

 	ATL Transformation Example	Author Éric Vépa evepa@sodius.com
	KM3 to Measure	August 30th , 2007

5. References

- [1] ATLAS (ATLantic dAta Systems) Official Webpage: <http://www.sciences.univ-nantes.fr/lina/ATLAS/>
- [2] The Kernel MetaMetaModel (KM3) Manual:
<http://www.eclipse.org/gmt/am3/km3/doc/KernelMetaMetaModel%5Bv00.06%5D.pdf>
- [3] The Atlantic Zoo: <http://www.eclipse.org/gmt/am3/zoos/atlanticZoo/>
- [4] Baroni, A.L.: *Formal Definition of Object-Oriented Design Metrics*. Master Thesis, Vrije University, Brussel, Belgium, 2002.
- [5] Baroni, A.L. and Abreu, F.B.: *A Formal Library for Aiding Metrics Extraction*. In: Workshop on Object-Oriented Reengineering (ECOOP'03), Darmstadt, Germany, July 2003.