	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Measuring Model Repositories	Date 2006/08/08

# 1 ATL Transformation Example: Measuring Model Repositories

The Measuring Model Repositories example describes three transformations from a KM3 model to a Measure model, from a Measure model to a Measure model and from a Measure model to a Table model.

## 1.1 Transformation overview

The aim of this transformation is to collect measurement data from models and stored the resulting data in a generic table model.

Measurement can be performed on one KM3 metamodel or on the entire zoo of KM3 metamodels by keeping only global information and merging it with these of the other metamodels.

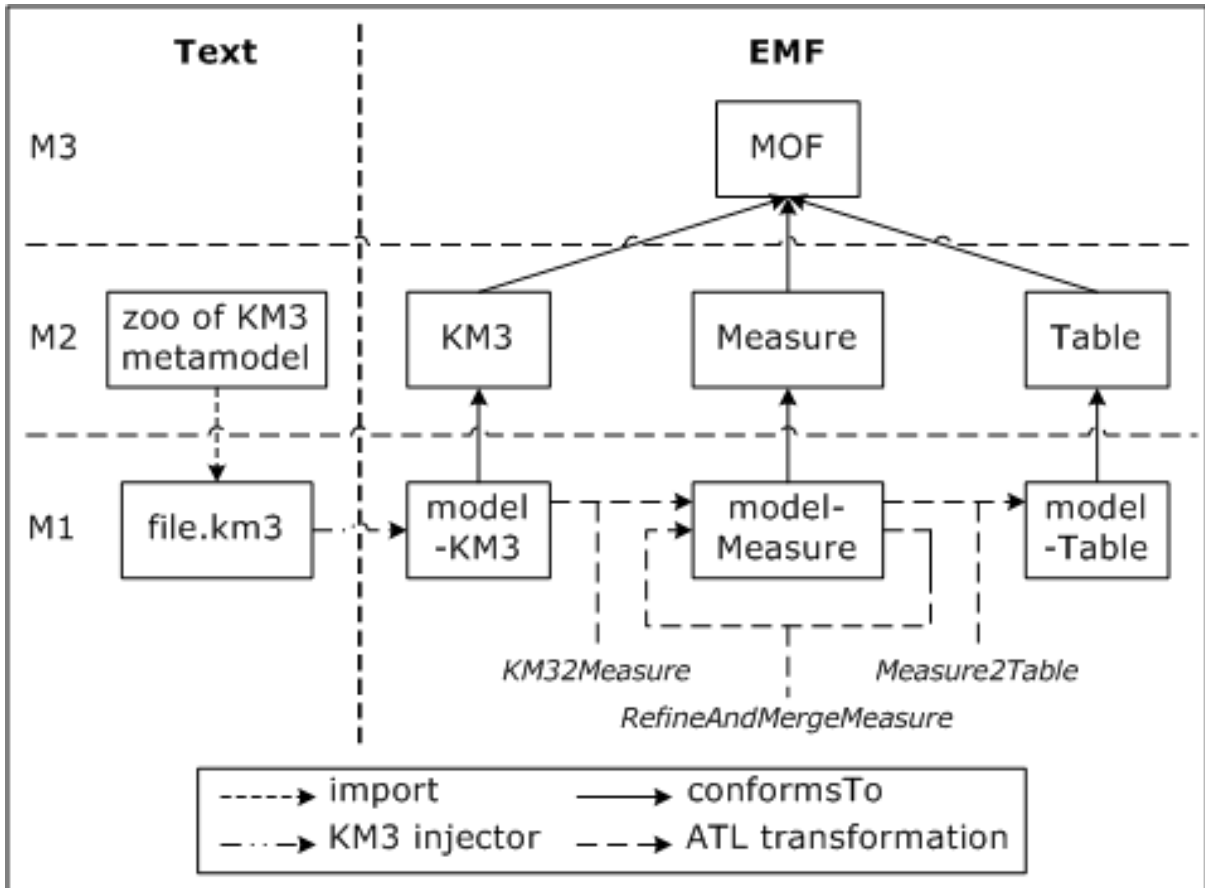



Figure 1: Overview of the transformation

	<b>ATL TRANSFORMATION EXAMPLE</b>	<b>Contributor</b> <b>Éric Vépa</b> <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	<b>Measuring Model Repositories</b>	<b>Date 2006/08/08</b>

## 1.2 Metamodels

### 1.2.1 KM3

The source metamodel for the Kernel MetaMetaModel KM3 will not be explained here.

### 1.2.2 Measure

The metamodel of Measure is described in Figure 2, and provided in Appendix A in KM3 format.

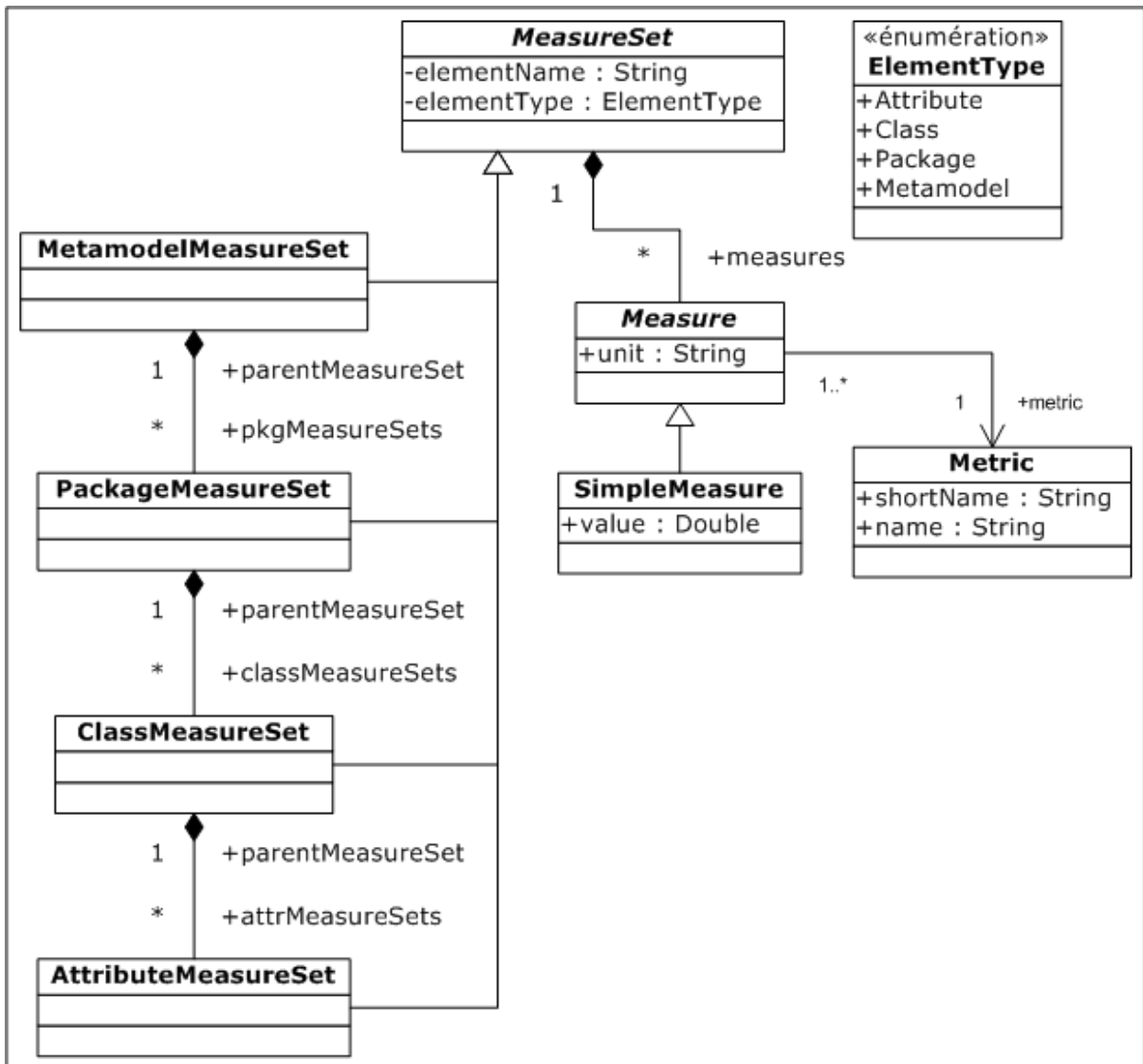



Figure 2: Measure metamodel

	<b>ATL TRANSFORMATION EXAMPLE</b>	<b>Contributor</b> <b>Éric Vépa</b> <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	<b>Measuring Model Repositories</b>	<b>Date 2006/08/08</b>

This metamodel offer the possibility of organizing sets of measure on different model elements (like metamodel, package, class or attribute). A set of measure owns a name and a type, from the model element concerned by the measurement.

A measure corresponds to a metric and has a unit. A simple measure also owns a value stored as a Double.

A metric is defined by a short and a long name. For instance, the metric corresponding to Number of Children will be represented with a short name "NOC" and the name "Number of Children".

### 1.2.3 Table

The target metamodel of Table is described in Figure 3, and provided in Appendix B in KM3 format.

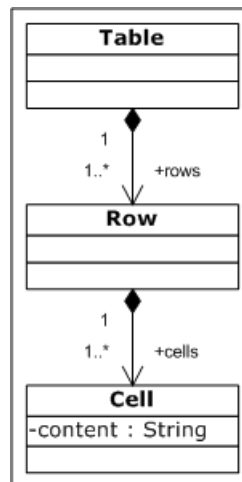



Figure 3: Table metamodel

Within this metamodel, a Table is associated with a Table element. Such an element is composed of several Rows that, in their turn, are composed of several Cells.

This metamodel is used to store the measurement data in different format. For instance, the header (first row) can be indicates that a table is for a certain type of presentation, etc.

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Measuring Model Repositories	Date 2006/08/08

### 1.3 Transformation from KM3 to Measure

#### 1.3.1 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 Metric element, with a short and long name, is created for each metric to measure.
- For each KM3 model element supported, the following elements are created:
  - A corresponding MeasureSet element, with a name and a type from the model element, chosen among the supported model elements, is created. MeasureSets are organized according to the hierarchy presents on the Measure metamodel.
  - Several SimpleMeasure elements, linked to a MeasureSet element, are created. A SimpleMeasure correspond to one of the Metric elements created for the whole model. A unit and a value are respectively given and calculated.

#### 1.3.2 ATL code

This ATL code for the KM32Measure transformation consists in 23 helpers and 6 rules.

The helper `divide` is used in case of a division by zero which returns zero and not NaN.

All the attribute helpers `allClasses`, `allAttributesInherited`, `allAttributes`, `allReferencesInherited` and `allReferences` returns a sequence of corresponding KM3 model element and are in several version, so they can be applied on different KM3 model elements.


The helpers `depthInheritanceTree` and `numberOfChildren` returns the value of the corresponding metric and can be applied on different KM3 elements.

The helper `metric` is used to found one of the Metric element created in the entrypoint rule for the whole model, by matching the short name of the metric.

The entrypoint rule `Metric()` allocates Metric elements. The rule creates Metric elements, with a short and long name, for each metric to measure.

The rules `MetamodelMeasureSet`, `PackageMeasureSet`, `ClassMeasureSet` and `AttributeMeasureSet` allocates a corresponding MeasureSet element for each corresponding KM3 model element supported. The rule creates a MeasureSet element which is composed of SimpleMeasure elements and can contains other MeasureSet elements.

The lazy rule `SimpleMeasure` allocates a SimpleMeasure. The rule created a SimpleMeasure element for one of the different Metric elements created in the entrypoint rule. The unit and value are given and calculated with the helpers.

	<b>ATL TRANSFORMATION EXAMPLE</b>	<b>Contributor</b> <b>Éric Vépa</b> <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	<b>Measuring Model Repositories</b>	<b>Date 2006/08/08</b>

```

--@name KM32Measure
--@version 1.0
--@domains measurement data, metrics, metamodel
--@authors Eric Vepa (eric.vepa <at> gmail.com)
--@date 2006/08/06
--@description This transformation is used to collect measurement data on a KM3
    metamodel. Some metrics are defined and measures are performed on the
    different model element and stored with the help of the Measure metamodel.

module KM32Measure; -- Module Template
create OUT : Measure from IN : KM3;

--@begin helper divide
--@comments returns a number even for a division by zero
helper context Real def : divide(divisor: Real) : Real =
    if divisor = 0
        then 0
        else self/divisor
    endif;
--@end helper divide

--@begin attribute helper allClasses
--@comments returns the sequence of all Class element of a Package or Metamodel
    element
helper context KM3!Package def : allClasses : Sequence(KM3!Class) =
    self.contents->select(c|c.ocllsTypeOf(KM3!Class));

helper context KM3!Metamodel def : allClasses : Sequence(KM3!Class) =
    self.contents->iterate(pkg; acc : Sequence(KM3!Class)=Sequence{}|
        acc->union(pkg.allClasses))->flatten();
--@end attribute helper allClasses


--@begin attribute helper allAttributesInherited
--@comments returns the sequence of all Attribute elements inherited of a Class
    , a Package or a Metamodel element
helper context KM3!Class def : allAttributesInherited : Sequence(KM3!Attribute)
    =
    if self.supertypes->isEmpty()
        then Sequence{}
        else self.supertypes->iterate(supertype; acc : Sequence(KM3!Attribute)=
            Sequence{}|
                acc->union(supertype.allAttributes))
        endif;

helper context KM3!Package def : allAttributesInherited : Sequence(KM3!
    Attribute) =
    self.allClasses->iterate(c; acc : Sequence(KM3!Attribute)=Sequence{}|
        acc->including(c.allAttributesInherited))->flatten();

helper context KM3!Metamodel def : allAttributesInherited : Sequence(KM3!
    Attribute) =
    self.allClasses->iterate(c; acc : Sequence(KM3!Attribute)=Sequence{}|
        acc->including(c.allAttributesInherited))->flatten();
--@end attribute helper allAttributesInherited

--@begin attribute helper allAttributes

```

	<b>ATL TRANSFORMATION EXAMPLE</b>	<b>Contributor</b> <b>Éric Vépa</b> <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	<b>Measuring Model Repositories</b>	<b>Date 2006/08/08</b>

```

--@comments returns the sequence of all Attribute elements (locally defined and
    inherited) of a Class, a Package or a Metamodel element
helper context KM3!Class def : allAttributes : Sequence(KM3!Attribute) =
    self.structuralFeatures ->select(sf|sf.oclIsTypeOf(KM3!Attribute))->
        union(self.allAttributesInherited);

helper context KM3!Package def : allAttributes : Sequence(KM3!Attribute) =
    self.allClasses->iterate(class; acc : Sequence(KM3!Attribute)=Sequence{}|
        acc->union(class.allAttributes))->flatten();

helper context KM3!Metamodel def : allAttributes : Sequence(KM3!Attribute) =
    self.contents->iterate(pkg; acc : Sequence(KM3!Attribute)=Sequence{}|
        acc->union(pkg.allAttributes))->flatten();
--@end attribute helper allAttributes

--@begin attribute helper allReferencesInherited
--@comments returns the sequence of all Reference elements inherited of a Class
    , a Package or a Metamodel element
helper context KM3!Class def : allReferencesInherited : Sequence(KM3!Reference)
    =
    if self.supertypes->isEmpty()
        then Sequence{}
        else self.supertypes->iterate(supertype; acc : Sequence(KM3!Reference)=
            Sequence{}|
                supertype.allReferences)
        endif;

helper context KM3!Package def : allReferencesInherited : Sequence(KM3!
    Reference) =
    self.allClasses->iterate(c; acc : Sequence(KM3!Reference)=Sequence{}|
        acc->including(c.allReferencesInherited))->flatten();


helper context KM3!Metamodel def : allReferencesInherited : Sequence(KM3!
    Reference) =
    self.allClasses->iterate(c; acc : Sequence(KM3!Reference)=Sequence{}|
        acc->including(c.allReferencesInherited))->flatten();
--@end attribute helper allReferencesInherited

--@begin attribute helper allReferences
--@comments returns the sequence of all Reference elements (locally defined and
    inherited) of a Class, a Package or a Metamodel element
helper context KM3!Class def : allReferences : Sequence(KM3!Reference) =
    self.structuralFeatures ->select(sf|sf.oclIsTypeOf(KM3!Reference))->
        union(self.allReferencesInherited)->flatten();

helper context KM3!Package def : allReferences : Sequence(KM3!Reference) =
    self.allClasses->iterate(class; acc : Sequence(KM3!Reference)=Sequence{}|
        --@comments returns Reference element without opposite or which not have
            container opposite
        acc->union(class.allReferences ->select(ref|
            if ref.opposite.oclIsUndefined()
                then true
                else not ref.opposite.isContainer
            endif)))->flatten();

helper context KM3!Metamodel def : allReferences : Sequence(KM3!Reference) =
    self.contents->iterate(pkg; acc : Sequence(KM3!Reference)=Sequence{}|

```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Measuring Model Repositories	Date 2006/08/08

```

    acc->union(pkg.allReferences)->flatten();
--@end attribute helper allReferences

--@begin helper attributeInheritanceFactor
--@comments returns the value of the metric Attribute Inheritance Factor for a
    Class, Package or Metamodel element
helper context KM3!LocatedElement def : attributeInheritanceFactor() : Real =
    self.allAttributesInherited->size().divide(self.allAttributes->size());
--@end helper attributeInheritanceFactor

--@begin helper depthInheritanceTree
--@comments returns the value of the metric Depth Inheritance Tree for a Class,
    Package or Metamodel element
helper context KM3!Class def : depthInheritanceTree() : Real =
    if self.supertypes->isEmpty()
        then 0
        else 1+self.supertypes->iterate(supertype; maxDIT:Real=0|
            maxDIT.max(supertype.depthInheritanceTree()))
    endif;

helper context KM3!Package def : depthInheritanceTree() : Real =
    self.allClasses->iterate(c; maxDIT:Real=0|maxDIT.max(c.depthInheritanceTree()
    ));

helper context KM3!Metamodel def : depthInheritanceTree() : Real =
    self.allClasses->iterate(c; maxDIT:Real=0|maxDIT.max(c.depthInheritanceTree()
    ));
--@end helper depthInheritanceTree

--@begin helper numberOfChildren
--@comments returns the value of the metric Number of Children for a Class,
    Package or Metamodel element
helper context KM3!Class def : numberOfChildren() : Real =
    KM3!Class.allInstances()->select(c|c.supertypes->includes(self))->size();


helper context KM3!Package def : numberOfChildren() : Real =
    if self.allClasses->isEmpty()
        then 0
        else self.allClasses->collect(c|c.numberOfChildren()->sum()
    endif;

helper context KM3!Metamodel def : numberOfChildren() : Real =
    if self.allClasses->isEmpty()
        then 0
        else self.allClasses->collect(c|c.numberOfChildren()->sum()
    endif;
--@end helper numberOfChildren

--@begin helper metric
--@comments returns the Metric element which shortName is given
helper def : metric(shortName: String) : Measure!Metric =
    Measure!Metric.allInstances()->select(metric|metric.shortName=shortName)->
        first();
--@end helper metric

--@begin entrypoint rule Metrics

```

	<b>ATL TRANSFORMATION EXAMPLE</b>	<b>Contributor</b> <b>Éric Vépa</b> <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	<b>Measuring Model Repositories</b>	<b>Date 2006/08/08</b>

```

--@comments creates all Metric elements with a short and long name
entrypoint rule Metrics() {
  to
    --@comments corresponds to the metric : Total Number of Packages
    metricTNP:Measure!Metric (
      shortName <- 'TNP',
      name <- 'Total Number of Packages'
    ),
    --@comments corresponds to the metric : Total Number of Classes
    metricTNC:Measure!Metric (
      shortName <- 'TNC',
      name <- 'Total Number of Classes'
    ),
    --@comments corresponds to the metric : Total Number of Attributes
    metricTNA:Measure!Metric (
      shortName <- 'TNA',
      name <- 'Total Number of Attributes'
    ),
    --@comments corresponds to the metric : Total Number of Attributes
      Inherited
    metricTNAI:Measure!Metric (
      shortName <- 'TNAI',
      name <- 'Total Number of Attributes Inherited'
    ),
    --@comments corresponds to the metric : Attribute Inheritance Factor
    metricAIF:Measure!Metric (
      shortName <- 'AIF',
      name <- 'Attribute Inheritance Factor'
    ),
    --@comments corresponds to the metric : Depth Inheritance Tree
    metricDIT:Measure!Metric (
      shortName <- 'DIT',
      name <- 'Depth Inheritance Tree'
    ),
    --@comments corresponds to the metric : Number of Children
    metricNOC:Measure!Metric (
      shortName <- 'NOC',
      name <- 'Number of Children'
    ),
    --@comments corresponds to the metric : Total Number of Relationships
    metricTNR:Measure!Metric (
      shortName <- 'TNR',
      name <- 'Total Number of Relationships'
    ),
    --@comments corresponds to the metric : Total Number of Relationships
      Inherited
    metricTNRI:Measure!Metric (
      shortName <- 'TNRI',
      name <- 'Total Number of Relationships Inherited'
    )
}
--@end entrypoint rule Metrics


--@begin rule MetamodelMeasureSet
--@comments collect measurement data on a Metamodel element
rule MetamodelMeasureSet {

```



**ATL TRANSFORMATION EXAMPLE****Contributor**  
**Éric Vépa**  
[eric.vepa@gmail.com](mailto:eric.vepa@gmail.com)**Measuring Model Repositories****Date 2006/08/08**

```
from
  mm:KM3!Metamodel
to
  mmMeasSet:Measure!MetamodelMeasureSet (
    elementName <- mm.contents->iterate(pkg; name:String='')|name +
      if pkg.name <> 'PrimitiveTypes'
        then pkg.name
        else ''
      endif),
    elementType <- #Metamodel,
    measures <- thisModule.SimpleMeasure('TNP','','',
      mm.contents->size()),
    measures <- thisModule.SimpleMeasure('TNC','','',
      mm.allClasses->size()),
    measures <- thisModule.SimpleMeasure('TNC','per Package',
      mm.allClasses->size().divide(mm.contents->size())),
    measures <- thisModule.SimpleMeasure('TNA','','',
      mm.allAttributes->size()),
    measures <- thisModule.SimpleMeasure('TNA','per Package',
      mm.allAttributes->size().divide(mm.contents->size())),
    measures <- thisModule.SimpleMeasure('TNA','per Class',
      mm.allAttributes->size().divide(mm.allClasses->size())),
    measures <- thisModule.SimpleMeasure('TNAI','','',
      mm.allAttributesInherited->size()),
    measures <- thisModule.SimpleMeasure('TNAI','per Package',
      mm.allAttributesInherited->size().divide(mm.contents->size())),
    measures <- thisModule.SimpleMeasure('TNAI','per Class',
      mm.allAttributesInherited->size().divide(mm.allClasses->size())),
    measures <- thisModule.SimpleMeasure('TNR','','',
      mm.allReferences->size()),
    measures <- thisModule.SimpleMeasure('TNR','per Package',
      mm.allReferences->size().divide(mm.contents->size())),
    measures <- thisModule.SimpleMeasure('TNR','per Class',
      mm.allReferences->size().divide(mm.allClasses->size())),
    measures <- thisModule.SimpleMeasure('TNRI','','',
      mm.allReferencesInherited->size()),
    measures <- thisModule.SimpleMeasure('TNRI','per Package',
      mm.allReferencesInherited->size().divide(mm.contents->size())),
    measures <- thisModule.SimpleMeasure('TNRI','per Class',
      mm.allReferencesInherited->size().divide(mm.allClasses->size())),
    measures <- thisModule.SimpleMeasure('AIF','per Class',
      mm.attributeInheritanceFactor()),
    measures <- thisModule.SimpleMeasure('DIT','','',
      mm.depthInheritanceTree()),
    measures <- thisModule.SimpleMeasure('DIT','per Package',
      if mm.contents->isEmpty()
        then 0
        else mm.contents->collect(c|c.depthInheritanceTree())->
          sum()/mm.contents->size()
      endif),
    measures <- thisModule.SimpleMeasure('DIT','per Class',
      if mm.allClasses->isEmpty()
        then 0
        else mm.allClasses->collect(c|c.depthInheritanceTree())->
          sum()/mm.allClasses->size()
      endif),
    measures <- thisModule.SimpleMeasure('NOC','','',
```

	<b>ATL TRANSFORMATION EXAMPLE</b>	<b>Contributor</b> <b>Éric Vépa</b> <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	<b>Measuring Model Repositories</b>	<b>Date 2006/08/08</b>

```

    mm.numberOfChildren()),
    measures <- thisModule.SimpleMeasure('NOC','per Package',
    mm.numberOfChildren().divide(mm.contents->size())),
    measures <- thisModule.SimpleMeasure('NOC','per Class',
    mm.numberOfChildren().divide(mm.allClasses->size())),
    pkgMeasureSets <- mm.contents
  )
}
--@end rule MetamodelMeasureSet

--@begin rule PackageMeasureSet
--@comments collect measurement data on a Package element
rule PackageMeasureSet {
  from
    pkg:KM3!Package
  to
    pkgMeasSet:Measure!PackageMeasureSet (
      elementName <- pkg.name,
      elementType <- #Package,
      measures <- thisModule.SimpleMeasure('TNC','',
        pkg.allClasses->size()),
      measures <- thisModule.SimpleMeasure('TNA','',
        pkg.allAttributes->size()),
      measures <- thisModule.SimpleMeasure('TNA','per Class',
        pkg.allAttributes->size().divide(pkg.allClasses->size())),
      measures <- thisModule.SimpleMeasure('TNAI','',
        pkg.allAttributesInherited->size()),
      measures <- thisModule.SimpleMeasure('TNAI','per Class',
        pkg.allAttributesInherited->size().divide(pkg.allClasses->size())),
      measures <- thisModule.SimpleMeasure('TNR','',
        pkg.allReferences->size()),
      measures <- thisModule.SimpleMeasure('TNR','per Class',
        pkg.allReferences->size().divide(pkg.allClasses->size())),
      measures <- thisModule.SimpleMeasure('TNRI','',
        pkg.allReferencesInherited->size()),
      measures <- thisModule.SimpleMeasure('TNRI','per Class',
        pkg.allReferencesInherited->size().divide(pkg.allClasses->size())),
      measures <- thisModule.SimpleMeasure('AIF','',
        pkg.attributeInheritanceFactor()),
      measures <- thisModule.SimpleMeasure('DIT','',
        pkg.depthInheritanceTree()),
      measures <- thisModule.SimpleMeasure('DIT','per Class',
        if pkg.allClasses->isEmpty()
        then 0
        else pkg.allClasses->collect(c|c.depthInheritanceTree()->
          sum()/pkg.allClasses->size()
        endif),
      measures <- thisModule.SimpleMeasure('NOC','',
        pkg.numberOfChildren()),
      measures <- thisModule.SimpleMeasure('NOC','per Class',
        pkg.numberOfChildren().divide(pkg.allClasses->size())),
      classMeasureSets <- pkg.allClasses
    )
}
--@end rule PackageMeasureSet

--@begin rule ClassMeasureSet


```



```
--@comments collect measurement data on a Class element
rule ClassMeasureSet {
  from
    class:KM3!Class
  to
    classMeasSet:Measure!ClassMeasureSet (
      elementName <- class.name,
      elementType <- #Class,
      measures <- thisModule.SimpleMeasure('TNA','','',
        class.allAttributes->size()),
      measures <- thisModule.SimpleMeasure('TNAI','','',
        class.allAttributesInherited->size()),
      measures <- thisModule.SimpleMeasure('TNR','','',
        class.allReferences->size()),
      measures <- thisModule.SimpleMeasure('TNRI','','',
        class.allReferencesInherited->size()),
      measures <- thisModule.SimpleMeasure('AIF','','',
        class.attributeInheritanceFactor()),
      measures <- thisModule.SimpleMeasure('DIT','','',
        class.depthInheritanceTree()),
      measures <- thisModule.SimpleMeasure('NOC','','',
        class.numberOfChildren()),
      attrMeasureSets <- class.allAttributes
    )
}
--@end rule ClassMeasureSet

--@begin rule AttributeMeasureSet
--@comments collect measurement data on a Attribute element
rule AttributeMeasureSet {
  from
    attr:KM3!Attribute
  to
    attrMeasSet:Measure!AttributeMeasureSet (
      elementName <- attr.name,
      elementType <- #Attribute
    )
}
--@end rule AttributeMeasureSet

--@begin lazy rule SimpleMeasure
--@comments stores a simple measure for the metric named 'shortName', the unit
'unit' and the value given
lazy rule SimpleMeasure {
  from
    shortName:String,
    unit:String,
    value:Real
  to
    simpleMeas:Measure!SimpleMeasure (
      metric <- thisModule.metric(shortName),
      unit <- unit,
      value <- value
    )
}
--@end lazy rule SimpleMeasure
```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Measuring Model Repositories	Date 2006/08/08

## 1.4 Transformation RefineAndMergeMeasure

### 1.4.1 Rules specification

These are the rules to refine and merge a Measure model with an other Measure model.

- For each MetamodelMeasureSet element, the following elements are created:
  - A MetamodelMeasureSet element, with the same name, type and simple measures, is created. The pkgMeasureSets reference is not copied.
- For each SimpleMeasure element, the following elements are created:
  - A SimpleMeasure element, with the same unit, value and linked to the same Metric element, is created.
- For each Metric element, the following elements are created:
  - A Metric element, with the same short and long name, is created.
- The other elements are not copied.

### 1.4.2 ATL code

This ATL code for the RefineAndMergeMeasure transformation consists in 3 rules.


The rule RefineMetamodelMeasureSet allocates a MetamodelMeasureSet. The rule creates a MetamodelMeasureSet element with the same name, type and simple measures.

The lazy rules CopySimpleMeasure and CopyMetric allocate respectively a SimpleMeasure and a Metric. The rules creates a SimpleMeasure element ("copyMeas") and a Metric element ("copyMetric"). The attributes of these elements are copied without change.

```
--@name RefineAndMergeMeasure
--@version 1.0
--@domains measurement data, metrics, metamodel, merged data
--@authors Eric Vepa (eric.vepa <at> gmail.com)
--@date 2006/08/06
--@description This transformation is used to refine and merge measurement data
  on metamodels. We refine the first input model of measures by keeping only
  MetamodelMeasureSet. Next, we merge these sets of measure with the existing
  ones in the second input model. The result measurement data are for severals
  metamodels.

module RefineAndMergeMeasure; -- Module Template
create OUT : Measure from IN1 : Measure, IN2 : Measure;

--@begin rule RefineMetamodelMeasureSet
rule RefineMetamodelMeasureSet {
  from
    mmMeasSet : Measure ! MetamodelMeasureSet
  to
    refinedMmMeasSet : Measure ! MetamodelMeasureSet (
      elementType <- mmMeasSet.elementType ,
```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Measuring Model Repositories	Date 2006/08/08

```

    elementName <- mmMeasSet.elementName ,
    measures <- mmMeasSet.measures ->
    select(meas | meas.oclIsTypeOf(Measure!SimpleMeasure)) ->
      iterate(meas; acc:Sequence(Measure!SimpleMeasure)=Sequence{} |
        acc->append(thisModule.CopySimpleMeasure(meas)))
  )
}
--@end rule RefineMetamodelMeasureSet

--@begin lazy rule CopySimpleMeasure
lazy rule CopySimpleMeasure {
  from
    meas:Measure!SimpleMeasure
  to
    copyMeas:Measure!SimpleMeasure (
      metric <- thisModule.CopyMetric(meas.metric),
      unit <- meas.unit,
      value <- meas.value
    )
}
--@end lazy rule CopySimpleMeasure

--@begin lazy rule CopyMetric
lazy rule CopyMetric {
  from
    metric:Measure!Metric
  to
    copyMetric:Measure!Metric (
      shortName <- metric.shortName,
      name <- metric.name
    )
}
--@end lazy rule CopyMetric


```

## 1.5 Transformation from Measure to Table

### 1.5.1 Rules specification

These are the rules to stored the measurement data from a Measure model to a Table model.

- For each kind of MeasureSet element, the following elements are created:
  - A Table element, containing several Row element, is created.
  - A first Row element, linked to the Table element, and containing several Cell elements, is created.
  - A first Cell element, linked to the first Row element, is created. The content is set to the type of the MeasureSet element.
  - For each SimpleMeasure element of the first MeasureSet element, the following elements are created:
    - \* A Cell element, linked to the first Row element, is created. The content is set to the short name of the metric concatenated with the unit of the SimpleMeasure element.

	<b>ATL TRANSFORMATION EXAMPLE</b>	<b>Contributor</b> <b>Éric Vépa</b> <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	<b>Measuring Model Repositories</b>	<b>Date 2006/08/08</b>

- For each MeasureSet element, the following elements are created:
  - \* A Row element, linked to the Table element, and containing several Cell elements, is created.
  - \* A first Cell element, linked to the Row element, is created. The content is set to the name of the MeasureSet element.
  - \* For each SimpleMeasure element of the MeasureSet element, the following elements are created:
    - A Cell element, linked to the Row element, is created. The content is set to the value of the SimpleMeasure element.
- For desired Metric element, the following elements are created:
  - A Table element, containing several Row element, is created.
  - A first Row element, linked to the Table element, and containing two Cell elements, is created.
  - A first Cell element, linked to the first Row element, is created. The content is set to "Bar Chart" or "Pie Chart", depending on the representation desired.
  - A second Cell element, linked to the first Row element, is created. The content is set to the short name of the metric concatenated with a desired unit.
  - For each MeasureSet element, the following elements are created:
    - \* A Row element, linked to the Table element, and containing two Cell elements, is created.
    - \* A first Cell element, linked to the Row element, is created. The content is set to the name of the MeasureSet element.
    - \* A second Cell element, linked to the Row element, is created. The content is set to the value of the SimpleMeasure element corresponding to the desired metric (this value is represented as a percentage for a "Pie Chart" Table).

### 1.5.2 ATL code

This ATL code for the Measure2Table transformation consists in 6 helpers and 14 rules.


The helper metric is used to found the Metric element by his short name.

The helper simpleMeasures returns the sequence of all SimpleMeasure elements, of a MeasureSet, for the metric which name is given.

The helper valueNotNull determinates if the value for the metric which name is given is not null.

The helper canCreatePieChart verify if the data measurement for a metric which name is given are sufficient for creating a table for a SVG pie chart representation (at least one row with a non null value).

The entrypoint rule Table() called the different called and lazy rules which creates Table elements for different representations.

	<b>ATL TRANSFORMATION EXAMPLE</b>	<b>Contributor</b> <b>Éric Vépa</b> <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	<b>Measuring Model Repositories</b>	<b>Date 2006/08/08</b>

The called rule `AllMeasureSet2Table` allocates a `Table`. The rule creates a `Table` element composed of several `Row` elements.

The called rule `TablesForEntireZoo` called lazy rules that create a `Table` element, composed of several `Row` elements, for SVG bar and pie chart representations.

The lazy rules `MeasureSet2RowName`, `MeasureSet2RowValue`, `ChartHeaderRow`, `MeasureName2RowBar` and `MeasureName2RowSector` allocate a `Row`. These rules create a `Row` element composed of several `Cell` elements.

The lazy rules `MeasureSet2CellElementType`, `MeasureSet2CellElementName`, `Measure2CellName` and `SimpleMeasure2CellValue` allocate a `Cell`. These rules create a `Cell` element. The content of the `Cell` depends on the desired representation. It can be the type or the name of a `MeasureSet` element, the short or long name of a `Metric` element concatenated with the unit of a `SimpleMeasure` element. A value of a `SimpleMeasure` element, recalculated or not. Or a simple constant `String` as "Bar Chart" or "Pie Chart".

The lazy rules `MeasureSets2SVGBarChart` and `MeasureSets2SVGPieChart` allocate a `Table`. These rules create a `Table` element which `Row` elements are composed of two `Cell` elements. These tables are used for the SVG representation of a metric with a bar or pie chart.


```
--@name Measure2Table
--@version 1.0
--@domains measurement data, metrics, metamodel, generic table representation
--@authors Eric Vepa (eric.vepa <at> gmail.com)
--@date 2006/08/06
--@description This transformation is used to represent measurement data on
metamodels as a generic table representation. Different kind of table are
created (different header row, number of columns, etc), depending on the
final representation (tabular HTML, SVG bar and pie chart, etc).

module Measure2Table; -- Module Template
create OUT : Table from IN : Measure;

--@begin helper metric
--@comments returns the Metric element which shortName is given
helper def : metric(shortName: String) : Measure!Metric =
  Measure!Metric.allInstances()->select(metric|metric.shortName=shortName)->
  first();
--@end helper metric

--@begin helper simpleMeasures
--@comments returns all the simple measures of a measure set for the metric
shortName given
helper context Measure!MeasureSet def : simpleMeasures(metricName: String) :
  Sequence(Measure!SimpleMeasure) =
  self.measures->select(m|m.oclIsTypeOf(Measure!SimpleMeasure))->
  select(meas|meas.metric.shortName = metricName);
--@end helper simpleMeasures

--@begin helper classMeasureSets
--@comments returns the sequence of all the ClassMeasureSet elements of a
MetamodelMeasureSet element
```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Measuring Model Repositories	Date 2006/08/08

```

helper context Measure!MetamodelMeasureSet def : classMeasureSets() : Sequence(
  Measure!ClassMeasureSet) =
  self.pkgMeasureSets->collect(pkg|pkg.classMeasureSets)->flatten();
--@end helper classMeasureSets

--@begin helper valueNotNull
--@comments returns true, if the value for the metric named 'metricName' is not
  null
helper context Measure!MeasureSet def : valueNotNull(metricName: String) :
  Boolean =
  self.simpleMeasures(metricName)->first().value <> 0;
--@end helper valueNotNull

--@begin helper canCreatePieChart
--@comments returns true if the data measurement on the metric named '
  metricName' are sufficient for creating a table for a SVG pie chart
  representation (at least one row with a non null value)
helper context Measure!PackageMeasureSet def : canCreatePieChart(metricName:
  String) : Boolean =
  if self.classMeasureSets->notEmpty()
    then self.classMeasureSets->exists(measSet|measSet.valueNotNull(metricName)
      )
    else false
  endif;

helper context Measure!MetamodelMeasureSet def : canCreatePieChart(metricName:
  String) : Boolean =
  if self.classMeasureSets->notEmpty()
    then self.classMeasureSets->exists(measSet|measSet.valueNotNull(
      metricName))
    else false
  endif;
--@end helper canCreatePieChart

--@begin entrypoint rule Tables
--@comments creates tables for different representations
entrypoint rule Tables() {
  using {
    --@comments only non empty measure sets are retained, then sorted by
      element name
    allMetamodelMeasuresSets : Sequence(Measure!MetamodelMeasureSet) =
      Measure!MetamodelMeasureSet.allInstances()->
        select(measSet|measSet.measures->notEmpty())->
          asSet()->sortedBy(measSet|measSet.elementName);
    allPackageMeasuresSets : Sequence(Measure!PackageMeasureSet) =
      Measure!PackageMeasureSet.allInstances()->
        select(measSet|measSet.measures->notEmpty())->
          asSet()->sortedBy(measSet|measSet.elementName);
    allClassMeasuresSets : Sequence(Measure!ClassMeasureSet) =
      Measure!ClassMeasureSet.allInstances()->
        select(measSet|measSet.measures->notEmpty())->
          asSet()->sortedBy(measSet|measSet.elementName);
    allAttributeMeasuresSets : Sequence(Measure!AttributeMeasureSet) =
      Measure!AttributeMeasureSet.allInstances()->
        select(measSet|measSet.measures->notEmpty())->
          asSet()->sortedBy(measSet|measSet.elementName);
  }
}

```






```
do {
  --@comments creates tables for each kind of non empty measure set [
    Metamodel][Zoo][TabularHTML]
  if allMetamodelMeasuresSets ->notEmpty()
    then thisModule.AllMeasureSet2Table(allMetamodelMeasuresSets)
    else OclUndefined
  endif;
  if allPackageMeasuresSets ->notEmpty()
    then thisModule.AllMeasureSet2Table(allPackageMeasuresSets)
    else OclUndefined
  endif;
  if allClassMeasuresSets ->notEmpty()
    then thisModule.AllMeasureSet2Table(allClassMeasuresSets)
    else OclUndefined
  endif;
  if allAttributeMeasuresSets ->notEmpty()
    then thisModule.AllMeasureSet2Table(allAttributeMeasuresSets)
    else OclUndefined
  endif;
  --@comments creates tables for metrics on one metamodel and for SVG bar
  chart representation [Metamodel][SVGBarChart]
  for (pkgMeasSet in allPackageMeasuresSets ->select(pkgMeasSet|pkgMeasSet.
    classMeasureSets ->notEmpty())) {
    thisModule.MeasureSets2SVGBarChart(pkgMeasSet.classMeasureSets,'TNAI');
    thisModule.MeasureSets2SVGBarChart(pkgMeasSet.classMeasureSets,'TNR');
    thisModule.MeasureSets2SVGBarChart(pkgMeasSet.classMeasureSets,'TNRI');
    thisModule.MeasureSets2SVGBarChart(pkgMeasSet.classMeasureSets,'AIF');
    thisModule.MeasureSets2SVGBarChart(pkgMeasSet.classMeasureSets,'DIT');
    thisModule.MeasureSets2SVGBarChart(pkgMeasSet.classMeasureSets,'NOC');
  }
  --@comments creates tables for metrics on one metamodel and for SVG pie
  chart representation [Metamodel][SVGPieChart]
  for (pkgMeasSet in allPackageMeasuresSets) {
    if pkgMeasSet.canCreatePieChart('TNA')
      then thisModule.MeasureSets2SVGPieChart(pkgMeasSet.classMeasureSets,'
        TNA')
      else OclUndefined
    endif;
    if pkgMeasSet.canCreatePieChart('TNAI')
      then thisModule.MeasureSets2SVGPieChart(pkgMeasSet.classMeasureSets,'
        TNAI')
      else OclUndefined
    endif;
    if pkgMeasSet.canCreatePieChart('TNR')
      then thisModule.MeasureSets2SVGPieChart(pkgMeasSet.classMeasureSets,'
        TNR')
      else OclUndefined
    endif;
    if pkgMeasSet.canCreatePieChart('TNRI')
      then thisModule.MeasureSets2SVGPieChart(pkgMeasSet.classMeasureSets,'
        TNRI')
      else OclUndefined
    endif;
    if pkgMeasSet.canCreatePieChart('AIF')
      then thisModule.MeasureSets2SVGPieChart(pkgMeasSet.classMeasureSets,'
        AIF')
      else OclUndefined
    endif;
  }
}
```



```
endif;
if pkgMeasSet.canCreatePieChart('DIT')
  then thisModule.MeasureSets2SVGPieChart(pkgMeasSet.classMeasureSets,'
    DIT')
  else OclUndefined
endif;
if pkgMeasSet.canCreatePieChart('NOC')
  then thisModule.MeasureSets2SVGPieChart(pkgMeasSet.classMeasureSets,'
    NOC')
  else OclUndefined
endif;
}
--@comments creates tables for metrics on the entire zoo of metamodels and
  for SVG bar and pie chart representation [Zoo][SVGBarChart][SVGPieChart]
if allPackageMeasuresSets->isEmpty()
  then thisModule.TablesForEntireZoo(allMetamodelMeasuresSets)
  else OclUndefined
endif;
}
}
--@end entrypoint rule Tables

--@begin called rule AllMeasureSet2Table
--@comments creates a table for all measure sets of one kind
rule AllMeasureSet2Table(allMeasSet:Sequence(Measure!MeasureSet)) {
  to
  globalTable:Table!Table (
    rows <- thisModule.MeasureSet2RowName(allMeasSet->first()),
    rows <- allMeasSet->iterate(measSet; acc:Sequence(Table!Row)=Sequence{}|
      acc->including(thisModule.MeasureSet2RowValue(measSet)))
  )
}
--@end called rule AllMeasureSet2Table

--@begin called rule TablesForEntireZoo
--@comments creates tables for metrics on the entire zoo of metamodels and for
  SVG bar and pie chart representation [Zoo][SVGBarChart][SVGPieChart]
rule TablesForEntireZoo(allMmMeasSet:Sequence(Measure!MetamodelMeasureSet)) {
  do {
    --@comments creates tables for metrics on the entire zoo of metamodels and
      for SVG bar chart representation [Zoo][SVGBarChart]
    thisModule.MeasureSets2SVGBarChart(allMmMeasSet,'TNP');
    thisModule.MeasureSets2SVGBarChart(allMmMeasSet,'TNC');
    thisModule.MeasureSets2SVGBarChart(allMmMeasSet,'TNA');
    thisModule.MeasureSets2SVGBarChart(allMmMeasSet,'TNAI');
    thisModule.MeasureSets2SVGBarChart(allMmMeasSet,'TNR');
    thisModule.MeasureSets2SVGBarChart(allMmMeasSet,'TNRI');
    thisModule.MeasureSets2SVGBarChart(allMmMeasSet,'AIF');
    thisModule.MeasureSets2SVGBarChart(allMmMeasSet,'DIT');
    thisModule.MeasureSets2SVGBarChart(allMmMeasSet,'NOC');
    --@comments creates tables for metrics on the entire zoo of metamodels and
      for SVG pie chart representation [Zoo][SVGPieChart]
    thisModule.MeasureSets2SVGPieChart(allMmMeasSet,'TNP');
    thisModule.MeasureSets2SVGPieChart(allMmMeasSet,'TNC');
    thisModule.MeasureSets2SVGPieChart(allMmMeasSet,'TNA');
    thisModule.MeasureSets2SVGPieChart(allMmMeasSet,'TNAI');
    thisModule.MeasureSets2SVGPieChart(allMmMeasSet,'TNR');
```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Measuring Model Repositories	Date 2006/08/08

```


    thisModule.MeasureSets2SVGPieChart(allMmMeasSet,'TNRI');
    thisModule.MeasureSets2SVGPieChart(allMmMeasSet,'AIF');
    thisModule.MeasureSets2SVGPieChart(allMmMeasSet,'DIT');
    thisModule.MeasureSets2SVGPieChart(allMmMeasSet,'NOC');
}
}
--@end called rule TablesForEntireZoo

--@begin unique lazy rule MeasureSet2RowName
--@comments creates a row with the type and all the names of the metrics of a
    MeasureSet element
unique lazy rule MeasureSet2RowName {
    from
        measSet:Measure!MeasureSet
    to
        rowName:Table!Row (
            cells <- thisModule.MeasureSet2CellElementType(measSet),
            cells <- measSet.measures->
                select(meas|meas.oclIsTypeOf(Measure!SimpleMeasure))->
                    iterate(meas; acc:Sequence(Table!Cell)=Sequence{}|
                        acc->including(thisModule.Measure2CellName(meas)))
        )
}
--@end unique lazy rule MeasureSet2RowName

--@begin lazy rule MeasureSet2RowValue
--@comments creates a row with the name and all the values of the simple
    measures of a MeasureSet element
lazy rule MeasureSet2RowValue {
    from
        measSet:Measure!MeasureSet
    to
        rowValue:Table!Row (
            cells <- thisModule.MeasureSet2CellElementName(measSet),
            cells <- measSet.measures->
                select(meas|meas.oclIsTypeOf(Measure!SimpleMeasure))->
                    iterate(meas; acc:Sequence(Table!Cell)=Sequence{}|
                        acc->including(thisModule.SimpleMeasure2CellValue(meas)))
        )
}
--@end lazy rule MeasureSet2RowValue

--@begin unique lazy rule MeasureSet2CellElementType
--@comments creates a cell with the type of a MeasureSet element
unique lazy rule MeasureSet2CellElementType {
    from
        measSet:Measure!MeasureSet
    to
        cellType:Table!Cell (
            content <- if measSet.elementType = #Attribute
                then 'Attribute'
            else if measSet.elementType = #Class
                then 'Class'
            else if measSet.elementType = #Package
                then 'Package'
            else if measSet.elementType = #Metamodel
                then 'Metamodel'
        )
}

```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Measuring Model Repositories	Date 2006/08/08

```

                else 'UnknowModelElement'
            endif
        endif
    endif
endif
)
}
--@end unique lazy rule MeasureSet2CellElementType


--@begin lazy rule MeasureSet2CellElementName
--@comments creates a cell with the name of a MeasureSet element
lazy rule MeasureSet2CellElementName {
    from
        measSet:Measure!MeasureSet
    to
        cellName:Table!Cell (
            content <- measSet.elementName
        )
}
--@end lazy rule MeasureSet2CellElementName

--@begin lazy rule Measure2CellName
--@comments creates a cell with the name of the metric and the unit of a
Measure element
lazy rule Measure2CellName {
    from
        meas:Measure!Measure
    to
        cellName:Table!Cell (
            content <- meas.metric.shortName +
                if meas.unit->size() <> 0
                    then ' ' + meas.unit
                    else ''
                endif
        )
}
--@end lazy rule Measure2CellName

--@begin lazy rule SimpleMeasure2CellValue
--@comments creates a cell with a the value of a SimpleMeasure element
lazy rule SimpleMeasure2CellValue {
    from
        meas:Measure!SimpleMeasure
    to
        cellValue:Table!Cell (
            content <- meas.value.toString()
        )
}
--@end lazy rule SimpleMeasure2CellValue

--@begin lazy rule ChartHeaderRow
--@comments creates a header row for a SVG chart representation
lazy rule ChartHeaderRow {
    from
        firstCellContent:String,
        metricName:String,

```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Measuring Model Repositories	Date 2006/08/08


```

    unit:String
  to
  headerRow:Table!Row (
    cells <- headerCell,
    cells <- metricNameCell
  ),
  headerCell:Table!Cell (
    content <- firstCellContent
  ),
  metricNameCell:Table!Cell (
    content <- thisModule.metric(metricName).name +
      ' ' + unit
  )
}
--@end lazy rule ChartHeaderRow

--@begin lazy rule MeasureSets2SVGBarChart
--@comments creates a table for a SVG bar chart representation and for one
metric
lazy rule MeasureSets2SVGBarChart {
  from
    measSets:Sequence(Measure!MeasureSet),
    metricName:String
  to
    barDiagTable:Table!Table (
      rows <- thisModule.ChartHeaderRow('Bar Chart',metricName,
        if measSets->first().oclIsTypeOf(Measure!AttributeMeasureSet)
        then 'per Attribute'
        else if measSets->first().oclIsTypeOf(Measure!ClassMeasureSet)
        then 'per Class'
        else if measSets->first().oclIsTypeOf(Measure!PackageMeasureSet)
        then 'per Package'
        else 'per Metamodel'
        endif
      endif),
      rows <- measSets->iterate(measSet; acc:Sequence(Table!Row)=Sequence{}|
        acc->including(thisModule.MeasureName2RowBar(measSet,metricName)))
    )
}
--@end lazy rule MeasureSets2SVGBarChart

--@begin lazy rule MeasureName2RowBar
--@comments creates a row, for a bar of the SVG bar chart representation, with
the name and the value of the model element for one metric
lazy rule MeasureName2RowBar {
  from
    measSet:Measure!MeasureSet,
    metricName:String
  to
    rowValue:Table!Row (
      cells <- thisModule.MeasureSet2CellElementName(measSet),
      cells <- thisModule.SimpleMeasure2CellValue(measSet.simpleMeasures(
        metricName)->first())
    )
}
--@end lazy rule MeasureName2RowBar


```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Measuring Model Repositories	Date 2006/08/08

```

--@begin lazy rule MeasureSets2SVGPieChart
--@comments creates a table for a SVG pie chart representation and for one
metric
lazy rule MeasureSets2SVGPieChart {
  from
    measSets:Sequence(Measure!MeasureSet),
    metricName:String
  using {
    sumSectors : Real = measSets->
      collect(measSet|measSet.simpleMeasures(metricName)->
        first().value)->sum();
  }
  to
    pieDiagTable:Table!Table (
      rows <- thisModule.ChartHeaderRow('Pie Chart',metricName,
        if measSets->first().oclIsTypeOf(Measure!AttributeMeasureSet)
        then 'per Attribute'
        else if measSets->first().oclIsTypeOf(Measure!ClassMeasureSet)
        then 'per Class'
        else if measSets->first().oclIsTypeOf(Measure!PackageMeasureSet)
        then 'per Package'
        else 'per Metamodel'
        endif
      endif
    ),
    rows <- measSets->iterate(measSet; acc:Sequence(Table!Row)=Sequence{}|
      --@comments creates a sector for non null value
      if measSet.valueNotNull(metricName)
      then acc->including(thisModule.MeasureName2RowSector(measSet,
        metricName,sumSectors))
      else acc
    endif)
  )
}
--@end lazy rule MeasureSets2SVGPieChart
--
--@begin lazy rule MeasureName2RowSector
--@comments creates a row, for a sector of the SVG pie chart representation,
with the name and the value of the model element for one metric
lazy rule MeasureName2RowSector {
  from
    measSet:Measure!MeasureSet,
    metricName:String,
    sumSectors:Real
  to
    rowValue:Table!Row (
      cells <- thisModule.MeasureSet2CellElementName(measSet),
      cells <- cellSector
    ),
    cellSector:Table!Cell (
      content <- (measSet.simpleMeasures(metricName)->
        first().value/sumSectors*100).toString()
    )
  )
}
--@end lazy rule MeasureName2RowSector

```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Measuring Model Repositories	Date 2006/08/08

## A Appendix: Measure metamodel in KM3 format

```

-- @name Measure
-- @version 1.0
-- @domains measurement data, metrics, metamodel
-- @authors Eric Vepa (eric.vepa <at> gmail.com)
-- @date 2006/08/06
-- @description This metamodel is a representation of measurement data on
metamodels.

--@begin package Measure
package Measure {
  --@begin abstract class MeasureSet
  --@comments defines an abstract set of measures on a named model element of a
certain type
  abstract class MeasureSet {
    attribute elementName : String;
    attribute elementType : ElementType;
    reference measures [*] ordered container : Measure oppositeOf measureSet;
  }
  --@end abstract class MeasureSet


  --@begin enumeration ElementType
  --@comments defines the possible types for a model element
  enumeration ElementType {
    literal Attribute;
    literal Class;
    literal Package;
    literal Metamodel;
  }
  --@end enumeration ElementType

  --@begin class MetamodelMeasureSet
  --@comments defines a set of measures on a metamodel
  class MetamodelMeasureSet extends MeasureSet {
    reference pkgMeasureSets [*] ordered container : PackageMeasureSet
oppositeOf parentMeasureSet;
  }
  --@end class MetamodelMeasureSet

  --@begin class PackageMeasureSet
  --@comments defines a set of measures on a package
  class PackageMeasureSet extends MeasureSet {
    reference classMeasureSets [*] ordered container : ClassMeasureSet
oppositeOf parentMeasureSet;
    reference parentMeasureSet : MetamodelMeasureSet oppositeOf pkgMeasureSets;
  }
  --@end class PackageMeasureSet

  --@begin class ClassMeasureSet
  --@comments defines a set of measures on a class
  class ClassMeasureSet extends MeasureSet {

```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Measuring Model Repositories	Date 2006/08/08

```

reference attrMeasureSets [*] ordered container : AttributeMeasureSet
    oppositeOf parentMeasureSet;
reference parentMeasureSet : PackageMeasureSet oppositeOf classMeasureSets;
}
--@end class ClassMeasureSet

--@begin class AttributeMeasureSet
--@comments defines a set of measures on an attribute
class AttributeMeasureSet extends MeasureSet {
    reference parentMeasureSet : ClassMeasureSet oppositeOf attrMeasureSets;
}
--@end class AttributeMeasureSet

--@begin abstract class Measure
--@comments defines an abstract measure for a certain metric, with a unit and
    contained by a measure set
abstract class Measure {
    reference metric : Metric;
    attribute unit : String;
    reference measureSet : MeasureSet oppositeOf measures;
}
--@end abstract class Measure

--@begin class SimpleMeasure
--@comments defines a simple measure with a value stored as a Double
class SimpleMeasure extends Measure {
    attribute value : Double;
}
--@end class SimpleMeasure

--@begin class Metric
--@comments defines a metric with a short and a complete name
class Metric {
    attribute shortName : String;
    attribute name : String;
}
--@end class Metric
}
--@end package Measure

--@begin package PrimitiveTypes
package PrimitiveTypes {
    datatype String;
    datatype Boolean;
    datatype Integer;
    datatype Double;
}
--@end package PrimitiveTypes

```


## B Appendix: Table metamodel in KM3 format

```

-- @name Table
-- @version 1.1
-- @domains spreadsheet

```



	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Measuring Model Repositories	Date 2006/08/08

```

-- @authors David Touzet (david.touzet@univ-nantes.fr)
-- @date 2005/04/12
-- @description This is a very basic abstract Table metamodel, which may be
  easily mapped to existing table representations (XHTML, ExcelML etc). Within
  this metamodel, a Table is associated with a Table element. Such an element
  is composed of several Rows that, in their turn, are composed of several
  Cells.

package Table {

  class Table {
    reference rows[1-]* ordered container : Row;
  }

  class Row {
    reference cells[1-]* ordered container : Cell;
  }

  class Cell {
    attribute content : String;
  }
}

package PrimitiveTypes {
  datatype String;
}

```