	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa eric.vepa@gmail.com
INRIA	Table to SVGBarChart	Date 2006/08/04

1 ATL Transformation Example: Table to SVGBarChart

The Table to SVGBarChart example describes a transformation from a Table model to a SVG file containing several bar chart representations.

1.1 Transformation overview

The aim of this transformation is to generate an SVG file from the input data contained in a Table model. This file can next be read with an SVG viewer or latest Internet navigator like Mozilla Firefox 1.5.

The generation of the output SVG file is realised by a first transformation from Table to SVG. Next, an extraction to an SVG file is necessary. This is done by applying a transformation from SVG to XML and the use of the XML extractor to obtained an XML file, which will be renamed into an .svg file.

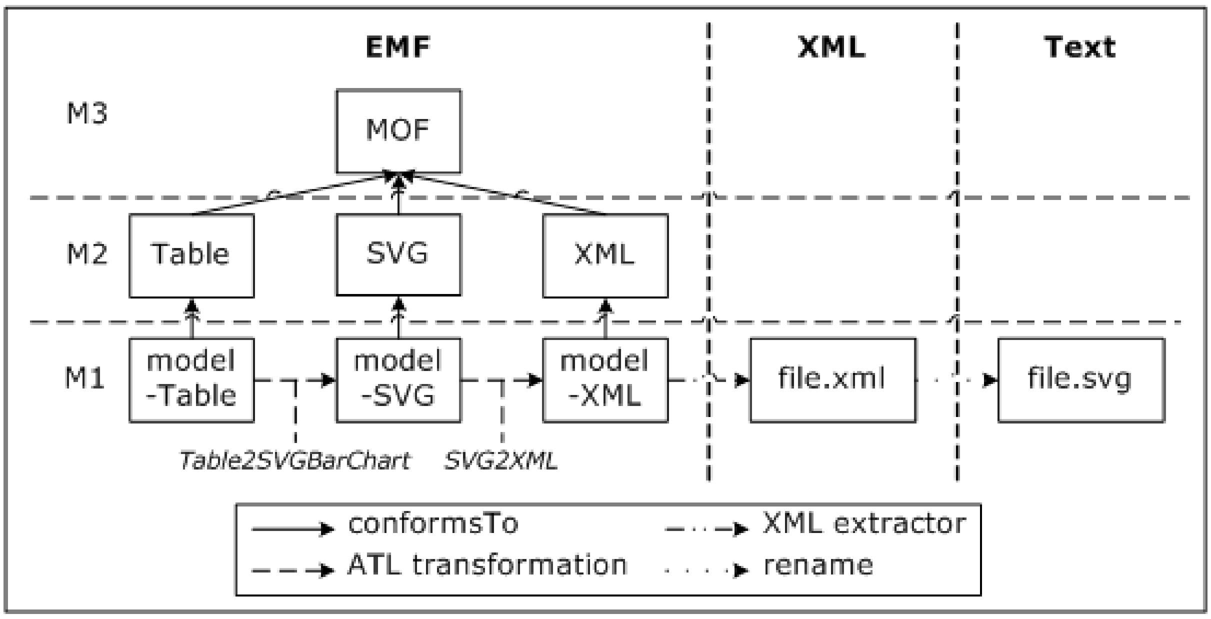

Figure 1: Overview of the transformation

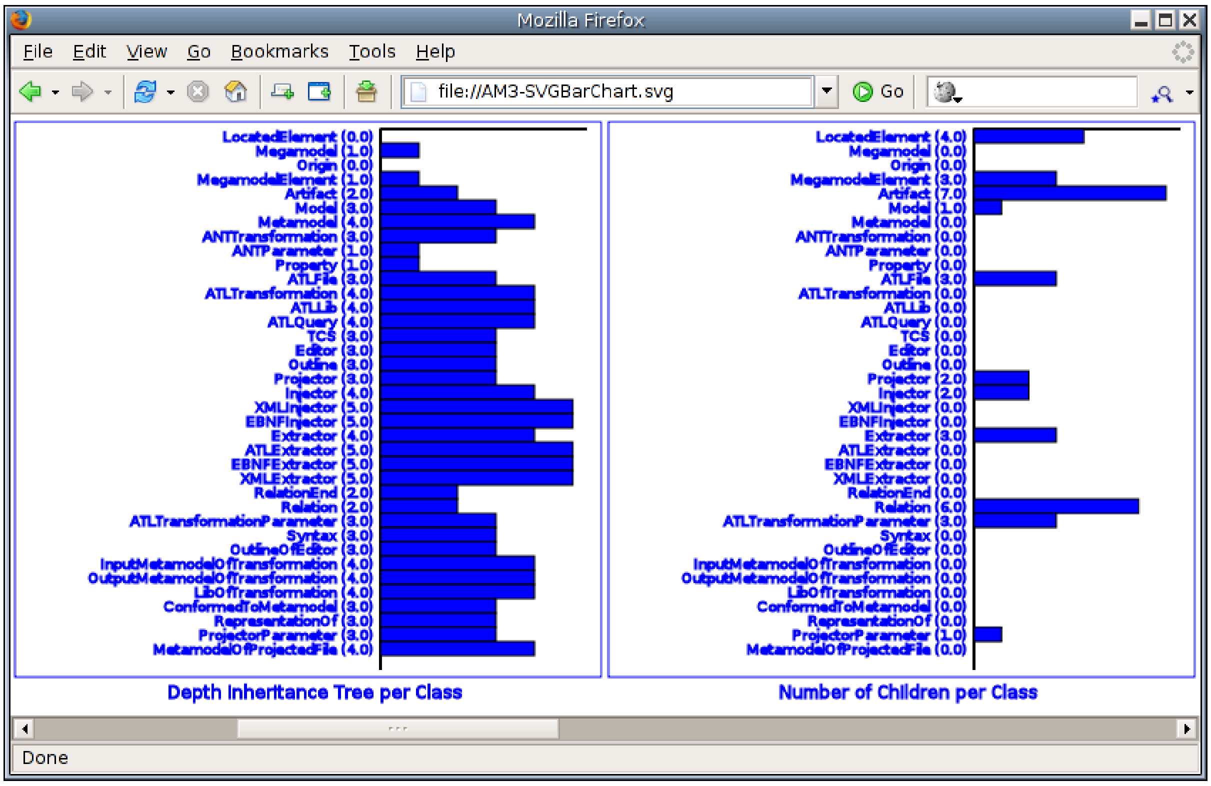

Figure 2: Sample of output bar chart in a SVG file

1.2 Metamodels

1.2.1 Table

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

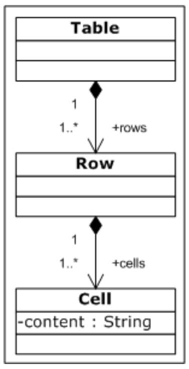

Figure 3: Table metamodel

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.

1.2.2 SVG

The simplified SVG metamodel, is described in Figure 4, and provided in complete version in Appendix B in KM3 format.

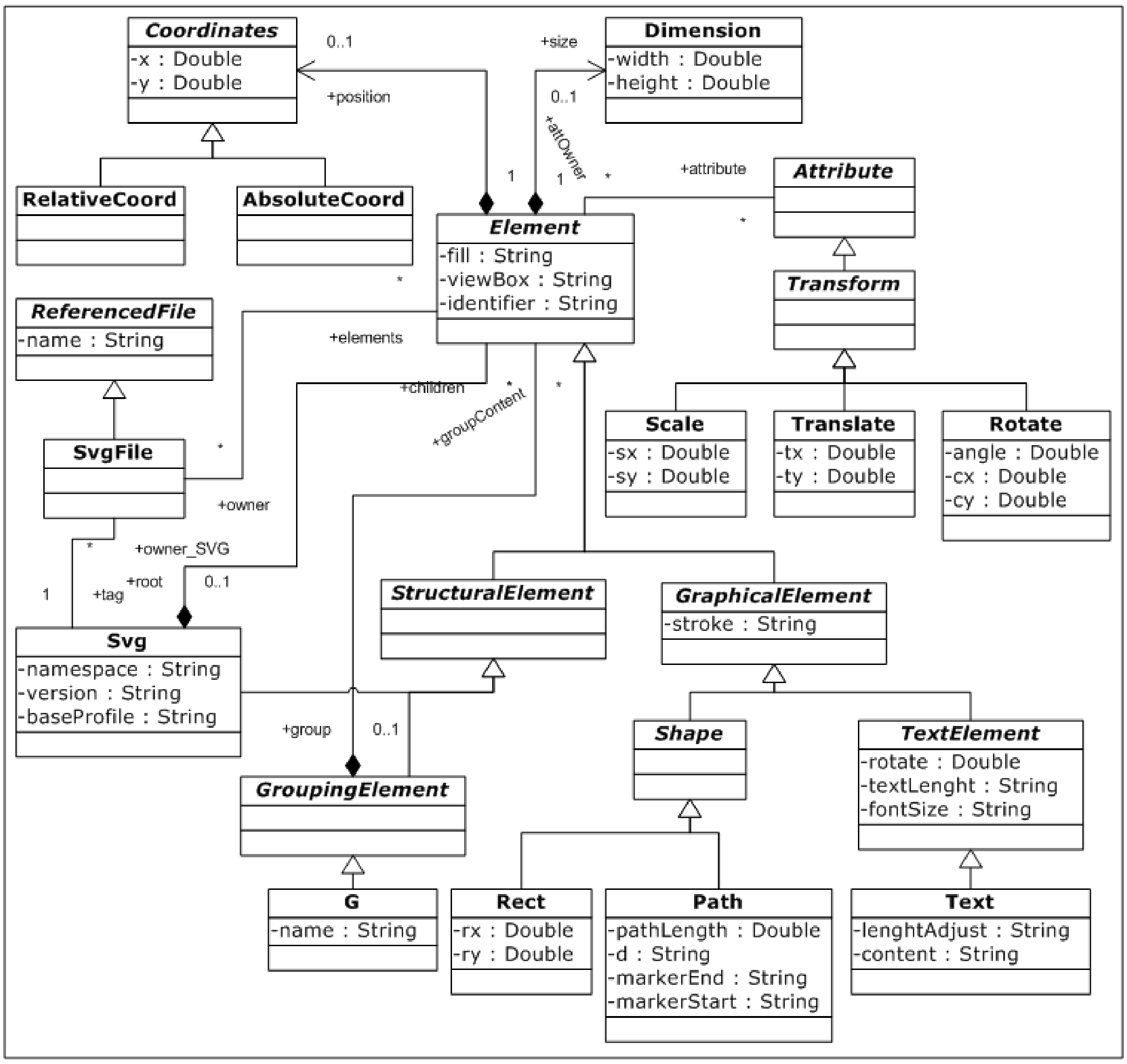

Figure 4: Simplified SVG metamodel

The transformation from input data stored in a table required only a subset of SVG language. In the sense, only the features used are represented on this figure.

1.3 Transformation from Table to SVGBarChart

1.3.1 Rules specification

These are the rules to transform a Table model to a SVG model containing bar charts.

- For the whole model, the following elements are created:
  - A SvgFile element composed of a Svg element.
  - A Svg element, linked to the SvgFile element, composed of a Dimension element. The attribute "namespace" is set to "http://www.w3.org/2000/svg" and the attribute "version" to "1.1".
  - A Dimension element, linked to the Svg element. Which "width" and "height" attribute are calculate according to the entry data.
- For each Table element, the following elements are created:
  - A G element, linked to the unique Svg element, composed of a Rect, Translate, Scale, Path and Text elements, is created.
  - A Rect element, linked to the G element, is created.
  - A Dimension and AbsoluteCoord elements, linked to the Rect element, are created. The value of their attributes are calculated according to the entry data.
  - A Translate, Scale and Path elements, linked to the G element, are created. The value of their attributes are calculated according to the entry data.
  - A Text element, linked to the G element, is created.
  - An AbsoluteCoord element, linked to the Text element, is created. The value of his attributes are calculated according to the entry data.
- For each Row element, the following elements are created:
  - A G element, linked to the G element created for the Table element, composed of a Rect and Text elements, is created.
  - A Rect element, linked to the G element, is created.
  - A Dimension and AbsoluteCoord elements, linked to the Rect element, are created. The value of their attributes are calculated according to the entry data.
  - A Text element, linked to the G element, is created.
  - An AbsoluteCoord element, linked to the Text element, is created. The value of his attributes are calculated according to the entry data.

1.3.2 ATL code

This ATL code for the Table2SVGBarChart transformation consists in 4 helpers and 3 rules.

The attribute helper allValidTables gets all the table that can be represented as a SVG bar chart. A valid table is a two columns table (two cells per row). The first row contains two cells with a String (the first cell contains the String 'Bar Chart' and the second cell, the name of the chart). The other rows contains one cell with a name as String and an other cell with the value as Double. The tables are also sorted.

The attribute helper maxSizeName gets, in all the valid tables, the maximum size among all name. This is done to specify the size of one bar chart.

The attribute helper prevRectWidth stores the previous width of all the existing charts. Serves to calculate the new translation of the new SVG bar chart.

The attribute helper svgFile save the unique SvgFile element.

The entrypoint rule SvgFile() allocates the structure of the SVG file. The rule creates an SvgFile element ("svgFile") which is composed of a Svg element ("svg"). The Svg element is composed of a Dimension element ("svgSize") and his attributes "namespace" and "version" are respectively set to "http://www.w3.org/2000/svg" and "1.1". In the do block, the SvgFile element created is associated to the attribute helper svgFile.

The rule Table2BarChart allocates a G for each Table element. The rule creates a G element ("g") which is composed of Rect ("rect"), Translate ("trans"), Scale ("scale"), Path ("axis") and Text ("text") elements. The Rect element is composed of a Dimension ("rectSize") and an AbsoluteCoord ("rectCoord") elements. The Text is also composed of an AbsoluteCoord element ("txtCoord"). All the value of the attributes of these elements are calculated with the helpers.
This rule is used to draw a scaled frame with a title, and position them in comparison of the other charts.

The lazy rule Row2Bar allocates a G for each Row element. The rule creates a G element ("g") which is composed of a Rect ("bar") and a Text ("text") elements. The Rect element is composed of a Dimension ("barSize") and an AbsoluteCoord ("barCoord") elements. The Text is also composed of an AbsoluteCoord element ("txtCoord"). All the value of the attributes of these elements are calculated with the helpers.
This lazy rule is used to draw a bar of the chart, with a name and the value, and is call for each Row element of a Table element.

```
--@name Table2SVG
--@version 1.0
--@domains Table, SVG, Bar chart
--@authors Eric Vepa (eric.vepa <at> gmail.com)
--@date 2006/08/04
--@description This transformation is used to transform generic tables into SVG
     bar charts. Each entry table is a two columns table (two cells per row).
    The first row contains two cells with a String (the first cell contains the
    String 'Bar Chart' and the second cell, the name of the chart). The other
    rows contains one cell with a name as String and an other cell with the
    value as Double.

module Table2SVG; -- Module Template
create OUT : SVG from IN : Table;

--@begin attribute helper allValidTables
```

```
--@comments returns all valid tables, ie tables which can be represented as bar
     chart
helper def : allValidTables : Sequence(Table!Table) =
  Table!Table.allInstances()->select(t|t.rows->first().cells->first().content =
       'Bar Chart')->
    asSet()->sortedBy(t|t.rows->first().cells->at(2).content);
--@end attribute helper allValidTables

--@begin helper maxSizeName
--@comments returns the max size of all name of a table
helper context Table!Table def : maxSizeName() : Integer =
  self.rows->subSequence(2,self.rows->size())->iterate(row; max:Integer=0|
    let value : Integer =
      let point : Integer = row.cells->at(2).content->indexOf('.') in
        (row.cells->first().content + row.cells->at(2).content->substring(1,
            point+2))->size() + 2 in
        if value > max
          then value
          else max
        endif)*7;
--@end helper maxSizeName

--@begin attribute helper prevRectWidth
--@comments returns the previous value of SVG rect width
helper def : prevRectWidth : Integer = 0;
--@end attribute helper prevRectWidth

--@begin entrypoint rule SvgFile
--@comments creates the SVG file with one svg tag
helper def: svgFile : SVG!SvgFile = OclUndefined;

entrypoint rule SvgFile() {
  to
    svgFile:SVG!SvgFile (
      tag <- svg
    ),
    svg:SVG!Svg (
      size <- svgSize,
      namespace <- 'http://www.w3.org/2000/svg',
      version <- '1.1'
    ),
    svgSize:SVG!Dimension (
      width <- thisModule.allValidTables->iterate(table; sum:Integer=0|
        sum + table.maxSizeName() + 170),
      height <- thisModule.allValidTables->iterate(table; max:Integer=0|
        if (table.rows.size()-1) > max
          then (table.rows.size()-1)
          else max
        endif)*10 + 50)
  do {
    thisModule.svgFile <- svgFile;
    for (table in thisModule.allValidTables) {
      thisModule.Table2BarChart(table);
    }
  }
}
--@end entrypoint rule SvgFile
```

```
--
--@begin lazy rule Table2BarChart
--@comments creates a bar chart (SVG group) for one valid table
lazy rule Table2BarChart {
  from
    table:Table!Table (
      table.rows->first().cells->first().content = 'Bar Chart'
    )
  using {
    rows : Sequence(Table!Row) = table.rows->subSequence(2,table.rows->size());
    scaleFactor : Real = 135/rows->iterate(row; max:Real=0|
      if row.cells->at(2).content.toReal() > max
        then row.cells->at(2).content.toReal()
        else max
      endif);
  }
  to
    g:SVG!G (
      attribute <- transl,
      attribute <- scale,
      groupContent <- text,
      groupContent <- rect,
      groupContent <- axis,
      groupContent <- rows->iterate(row; acc:Sequence(SVG!G)=Sequence{}|
        acc->including(thisModule.Row2Bar(rows.indexOf(row),scaleFactor,row)))
    ),
    rect:SVG!Rect (
      size <- rectSize,
      position <- rectCoord,
      fill <- 'none',
      stroke <- 'blue'
    ),
    rectSize:SVG!Dimension (
      width <- 5+table.maxSizeName()+5+145+5,
      height <- (table.rows.size()+1)*10
    ),
    rectCoord:SVG!AbsoluteCoord (
      x <- 0-table.maxSizeName()-5,
      y <- 0-5
    ),
    transl:SVG!Translate (
      tx <- table.maxSizeName()+10 + thisModule.prevRectWidth,
      ty <- 10
    ),
    scale:SVG!Scale (
      sx <- 1,
      sy <- scale.sx
    ),
    axis:SVG!Path (
      d <- 'M145,0 H0 V' + (table.rows.size()*10).toString() + ',0 z',
      fill <- 'none',
      stroke <- 'black'
    ),
    text:SVG!Text (
      position <- textCoord,
      stroke <- 'blue',
      fontSize <- '12',
      --@comments text-anchor value strored in lengthAdjust attribute
      lengthAdjust <- 'middle',
      content <- table.rows->first().cells->at(2).content
    ),
    textCoord:SVG!AbsoluteCoord (
      x <- rectSize.width/2-table.maxSizeName(),
      y <- rectSize.height+10
    )

  do {
    thisModule.prevRectWidth <- thisModule.prevRectWidth + rectSize.width + 5;
    thisModule.svgFile.tag.children <- g;
  }
}
--@end lazy rule Table2BarChart

--@begin lazy rule Row2Bar
--@comments creates a bar (SVG line) for the row at position given
lazy rule Row2Bar {
  from
    position:Integer,
    scaleFactor:Real,
    row:Table!Row
  using {
    value : String =
      let point : Integer = row.cells->at(2).content->indexOf('.') in
        row.cells->at(2).content->substring(1,point+2);
  }
  to
    g:SVG!G (
      groupContent <- text,
      groupContent <- bar
    ),
    bar:SVG!Rect (
      size <- barSize,
      position <- barCoord,
      fill <- 'blue',
      stroke <- 'black'
    ),
    barSize:SVG!Dimension (
      width <- row.cells->at(2).content.toReal()*scaleFactor,
      height <- 10
    ),
    barCoord:SVG!AbsoluteCoord (
      x <- 0,
      y <-barSize.height*(position-1)
    ),
    text:SVG!Text (
      position <- txtCoord,
      stroke <- 'blue',
      fontSize <- '8',
      --@comments text-anchor value strored in lengthAdjust attribute
      lengthAdjust <- 'end',
      content <- row.cells->first().content + ' (' + value + ')'
    ),
    txtCoord:SVG!AbsoluteCoord (
      x <- 0-5,
            y <- barCoord.y+barSize.height-1
        )
}
--@end lazy rule Row2Bar
```

1.4 Extractor

1.4.1 Rules specifications

These are the rules to transform a SVG model to an XML model.

- For the Svg element, a Root element which name is "svg" is created and composed of two Attribute elements which name and value are these of the "namespace" and "version" attribute.
- For the G, Rect and Path elements, an Element element composed of Attributes elements created for own attributes, is created.
- An additional Text element is created for the Text element which value is the content of the Text element.
- For each Dimension or AbsoluteCoord elements, two Attribute elements which name are the name and value are these of the attribute of the Dimension or AbsoluteCoord elements.
- For the Scale, Translate and Rotate elements, an Attribute element which name is "transform" and which value is an arranged String with the value of the attributes among these three, is created.
- For each attribute of an element of SVG metamodel, an Attribute element, linked to the element created for his owner, which name and value are the same as these of the attribute, is created.

1.4.2 ATL code

This ATL code for the SVG2XML transformation consists in 3 helpers and 7 rules.

The 3 helpers returns a String value for the transformation elements Scale, Translate and Rotate.

The rule Svg2Root allocates a Root for the Svg element. The rule creates a Root element ("root"). Attribute elements are created for "size" reference on Dimension element. Attribute and Element elements are also created and linked for other attributes and children.

The G2Element rule allocates an Element for the G element. The rule creates an Element element ("elmt") which is composed of Element and Attribute. If one of the three transformation element is defined then an Attribute element named "transform" is created. His value is set with the helpers in function of these which are defined.

The next 4 rules have a similar behavior. The minor difference is that the rule Text2Element creates an additional Text element which value is the content of SVG Text element.

The last lazy rule Attribute allocates an Attribute element for attribute of an element of SVG metamodel. The name of the attribute and his defined or default value are required for this lazy rule.

```
--@name SVG2XML
--@version 1.0
--@domains SVG, Pie chart
--@authors Eric Vepa (eric.vepa <at> gmail.com)
--@date 2006/08/04
--@description XML extractor for SVG pie charts.
--@see
--@comments

module SVG2XML; -- Module Template
create OUT : XML from IN : SVG;

--@begin helper scale
--@comments returns the string value for a scale transformation attribute
helper context SVG!Scale def : scale() : String =
  'scale(' + self.sx.toString() +
  if self.sy = self.sx
    then ''
    else ',' + self.sy.toString()
  endif + ')';
--@end helper scale

--@begin helper translate
--@comments returns the string value for a translate transformation attribute
helper context SVG!Translate def : translate() : String =
  'translate(' + self.tx.toString() + ',' + self.ty.toString() + ')';
--@end helper translate

--@begin helper rotate
--@comments returns the string value for a rotate transformation attribute
helper context SVG!Rotate def : rotate() : String =
  'rotate(' + self.angle.toString() + ')';
--@end helper rotate

--@begin rule Svg2Root
rule Svg2Root {
  from
    svg:SVG!Svg
  to
    root:XML!Root (
      name <- 'svg',
      children <- xmlns,
      children <- version,
      children <- thisModule.Attribute('width', if not svg.size.oclIsUndefined
          () then svg.size.width.toString() else '100%' endif),
      children <- thisModule.Attribute('height', if not svg.size.oclIsUndefined
          () then svg.size.height.toString() else '100%' endif),
      children <- svg.children
    ),
    xmlns:XML!Attribute (
      name <- 'xmlns',
      value <- svg.namespace
    ),
    version:XML!Attribute (
      name <- 'version',
      value <- svg.version
    )
}
--@end rule Svg2Root

--@begin rule G2Element
rule G2Element {
  from
    g:SVG!G
  using {
    transforms : Sequence(SVG!Transform) = g.attribute->select(a|a.oclIsKindOf(
        SVG!Transform));
    transformValue : String = transforms->iterate(transf; str:String=''|str +
      if transf.oclIsTypeOf(SVG!Scale)
        then transf.scale()
        else if transf.oclIsTypeOf(SVG!Translate)
          then transf.translate()
          else if transf.oclIsTypeOf(SVG!Rotate)
            then transf.rotate()
            else ''
          endif
        endif
      endif +
      if transf <> transforms->last()
        then ' '
        else ''
      endif);
  }
  to
    elmt:XML!Element (
      name <- 'g',
      children <- thisModule.Attribute('transform', if transforms->notEmpty()
          then transformValue else '' endif),
      children <- thisModule.Attribute('fill', if not g.fill.oclIsUndefined()
          then g.fill else 'black' endif),
      children <- g.groupContent
    )
}
--@end rule G2Element

--@begin rule Rect2Element
rule Rect2Element {
  from
    rect:SVG!Rect
  to
    elmt:XML!Element (
      name <- 'rect',
      children <- thisModule.Attribute('x', if not rect.position.oclIsUndefined
          () then rect.position.x.toString() else '0' endif),
      children <- thisModule.Attribute('y', if not rect.position.oclIsUndefined
          () then rect.position.y.toString() else '0' endif),
      children <- thisModule.Attribute('width', if not rect.size.oclIsUndefined
          () then rect.size.width.toString() else '100%' endif),
      children <- thisModule.Attribute('height', if not rect.size.
          oclIsUndefined() then rect.size.height.toString() else '100%' endif),
      children <- thisModule.Attribute('fill', if not rect.fill.oclIsUndefined
          () then rect.fill else 'black' endif),
      children <- thisModule.Attribute('stroke', if not rect.stroke.
          oclIsUndefined() then rect.stroke else 'none' endif)
    )
}
--@end rule Rect2Element

--@begin rule Circle2Element
rule Circle2Element {
  from
    circ:SVG!Circle
  to
    elmt:XML!Element (
      name <- 'circle',
      children <- thisModule.Attribute('x', if not circ.position.oclIsUndefined
          () then circ.position.x.toString() else '0' endif),
      children <- thisModule.Attribute('y', if not circ.position.oclIsUndefined
          () then circ.position.y.toString() else '0' endif),
      children <- thisModule.Attribute('r', if not circ.size.oclIsUndefined()
          then circ.size.width.toString() else '100%' endif),
      children <- thisModule.Attribute('fill', if not circ.fill.oclIsUndefined
          () then circ.fill else 'black' endif),
      children <- thisModule.Attribute('stroke', if not circ.stroke.
          oclIsUndefined() then circ.stroke else 'none' endif)
    )
}
--@end rule Circle2Element

--@begin rule Path2Element
rule Path2Element {
  from
    path:SVG!Path
  to
    elmt:XML!Element (
      name <- 'path',
      children <- thisModule.Attribute('d', path.d),
      children <- thisModule.Attribute('fill', if not path.fill.oclIsUndefined
          () then path.fill else 'black' endif),
      children <- thisModule.Attribute('stroke', if not path.stroke.
          oclIsUndefined() then path.stroke else 'none' endif)
    )
}
--@end rule Path2Element

--@begin rule Text2Element
rule Text2Element {
  from
    text:SVG!Text
  to
    elmt:XML!Element (
      name <- 'text',
      children <- thisModule.Attribute('x', if not text.position.oclIsUndefined
          () then text.position.x.toString() else '0' endif),
      children <- thisModule.Attribute('y', if not text.position.oclIsUndefined
          () then text.position.y.toString() else '0' endif),
      children <- thisModule.Attribute('stroke', if not text.stroke.
          oclIsUndefined() then text.stroke else 'none' endif),
      children <- thisModule.Attribute('font-size', if not text.fontSize.
          oclIsUndefined() then text.fontSize else 'medium' endif),
      --@comments text-anchor value stored in lengthAdjust attribute
      children <- thisModule.Attribute('text-anchor', if not text.lengthAdjust.
          oclIsUndefined() then text.lengthAdjust else 'start' endif),
      children <- txt
    ),
    txt:XML!Text (
      value <- text.content
    )
}
--@end rule Text2Element

--@begin lazy rule Attribute
lazy rule Attribute {
  from
    attrName:String,
    attrValue:String
  to
    attr:XML!Attribute (
      name <- attrName,
      value <- attrValue
    )
}
--@end lazy rule Attribute
```

A Appendix: Table metamodel in KM3 format

```
-- @name  Table
-- @version 1.1
-- @domains spreadsheet
-- @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;
}
```

B Appendix : SVG metamodel in KM3 format

```
-- @name    SVG
-- @version   1.1
-- @domains   graphics, XML
-- @authors   Jean Palies
-- @date    2005/04/07
-- @description This metamodel defines a susbset of the W3C standard SVG (
    Scalable Vector Graphics), an XMLbased format for graphical rendering.
-- @see     Scalable Vector Graphics 1.1, World Wide Web Consortium, http://www
    .w3.org/TR/SVG11/

package SVG {

-- @comment Element is the top of the hierarchy
  abstract class Element {
    reference owner[*] : SvgFile oppositeOf elements;
    reference target[*] : Use oppositeOf use;
    reference "attribute"[*] : Attribute oppositeOf attOwner;
    reference position[0-1] container : Coordinates;
    reference size[0-1] container : Dimension;
    reference root[0-1] : Svg oppositeOf children;
    attribute fill[0-1] : String;
    attribute viewBox[0-1] : String;
    reference group[0-1] : GroupingElement oppositeOf groupContent;
    attribute identifier[0-1] : String;
    reference drawsMarker[0-1] : Marker oppositeOf drawing;
  }


-- @comment Structural Elements
  abstract class StructuralElement extends Element {
  }

  class Image extends StructuralElement {
    reference referee[*] : ReferencedFile oppositeOf referer;
  }

  class Svg extends StructuralElement {
    reference owner_SVG[*] : SvgFile oppositeOf tag;
    reference children[*] ordered container : Element oppositeOf root;
    attribute namespace[0-1] : String;
    attribute version[0-1] : String;
    attribute baseProfile[0-1] : String;
  }

  abstract class GroupingElement extends StructuralElement {
    reference groupContent[*] ordered container : Element oppositeOf group;
  }

  class G extends GroupingElement {
    attribute name[0-1] : String;
  }

  class Defs extends GroupingElement {
  }

  class Symbol extends GroupingElement {
  }

  class Use extends StructuralElement {
    reference use[*] : Element oppositeOf target;
  }

  abstract class GraphicalElement extends Element {
    attribute stroke[0-1] : String;
  }

  abstract class Shape extends GraphicalElement {
  }

  abstract class TextElement extends GraphicalElement {
    attribute rotate[0-1] : Double;
    attribute textLength[0-1] : String;
    attribute fontSize[0-1] : String;
  }

-- @comment Geometry
  class Rect extends Shape {
    attribute rx[0-1] : Double;
    attribute ry[0-1] : Double;
  }

  class Circle extends Shape {
  }

  class Ellipse extends Shape {
  }

  class Line extends Shape {
    reference between[2-2] : Point;
    attribute markerEnd[0-1] : String;
    attribute markerStart[0-1] : String;
  }

  class Polyline extends Shape {
    reference waypoints[*] ordered container : Point;
    attribute strokeDashArray[0-1] : String;
    attribute markerEnd[0-1] : String;
    attribute markerStart[0-1] : String;
  }

  class Polygon extends Shape {
    reference waypoints[*] ordered : Point;
    attribute markerEnd[0-1] : String;
    attribute markerStart[0-1] : String;
  }

  class Path extends Shape {
    attribute pathLength[0-1] : Double;
    attribute d : String;
    attribute markerEnd[0-1] : String;
    attribute markerStart[0-1] : String;
  }

  class Point extends Shape {
  }

  class Marker extends Shape {
    attribute markerUnits[0-1] : String;
    attribute refX[0-1] : Double;
    attribute refY[0-1] : Double;
    attribute markerWidth[0-1] : Double;
    attribute markerHeight[0-1] : Double;
    attribute orient[0-1] : String;
    reference drawing[*] container : Element oppositeOf drawsMarker;
  }
-- End Geometry

-- @comment Text
  class Text extends TextElement {
    attribute lengthAdjust[0-1] : String;
    attribute content : String;
  }

  class Tspan extends TextElement {
    attribute content[0-1] : String;
  }

  class Tref extends TextElement {
    reference xlinkHref : TextElement;
  }
-- End Text

-- @comment Special attributes
  abstract class Attribute {
    reference attOwner[*] : Element oppositeOf "attribute";
  }

  abstract class Transform extends Attribute {
  }

  class Scale extends Transform {
    attribute sx : Double;
    attribute sy : Double;
  }

  class Translate extends Transform {
    attribute tx : Double;
    attribute ty : Double;
  }

  class Rotate extends Transform {
    attribute angle : Double;
    attribute cx : Double;
    attribute cy : Double;
  }

  class Visibility extends Attribute {
    attribute visible : Boolean;
  }

  class FontWeight extends Attribute {
    attribute bold : Boolean;
  }

  class FontStyle extends Attribute {
    attribute italic : Boolean;
  }
-- End special attributes

-- @comment Coordinates and Dimension
  -- @comment For width, height. length is the longer radius of an ellipse.
  class Dimension {
    attribute width : Double;
    attribute height : Double;
  }

  -- @comment Coordinates are either relative or absolute
  abstract class Coordinates {
    attribute x : Double;
    attribute y : Double;
  }
  class RelativeCoord extends Coordinates {
  }

  class AbsoluteCoord extends Coordinates {
  }
-- End Coordinates and Dimension

-- @comment Files
  -- @comment A file that is referenced by some tag in the document
  abstract class ReferencedFile {
    reference referer[*] : Image oppositeOf referee;
    attribute name : String;
  }

  -- @comment A svg file that is referenced via a use tag calling its svg tag
  class SvgFile extends ReferencedFile {
    reference tag : Svg oppositeOf owner_SVG;
    reference elements[*] : Element oppositeOf owner;
  }
-- End Files
}

package PrimitiveTypes {
  datatype Boolean;
  datatype Integer;
  datatype String;
  datatype Double;
}
```