1 ATL Transformation Example: KMLInjector

The KMLInjector example describes a transformation from a KML file to a KML model.

1.1 Transformation overview

The aim of this transformation is to inject a KML file into a KML model.

The first step is to rename the KML file to an XML file and delete the KML tag from the file. Next, use the XML injector to obtains an XML model. Finally run the transformation from XML to KML.

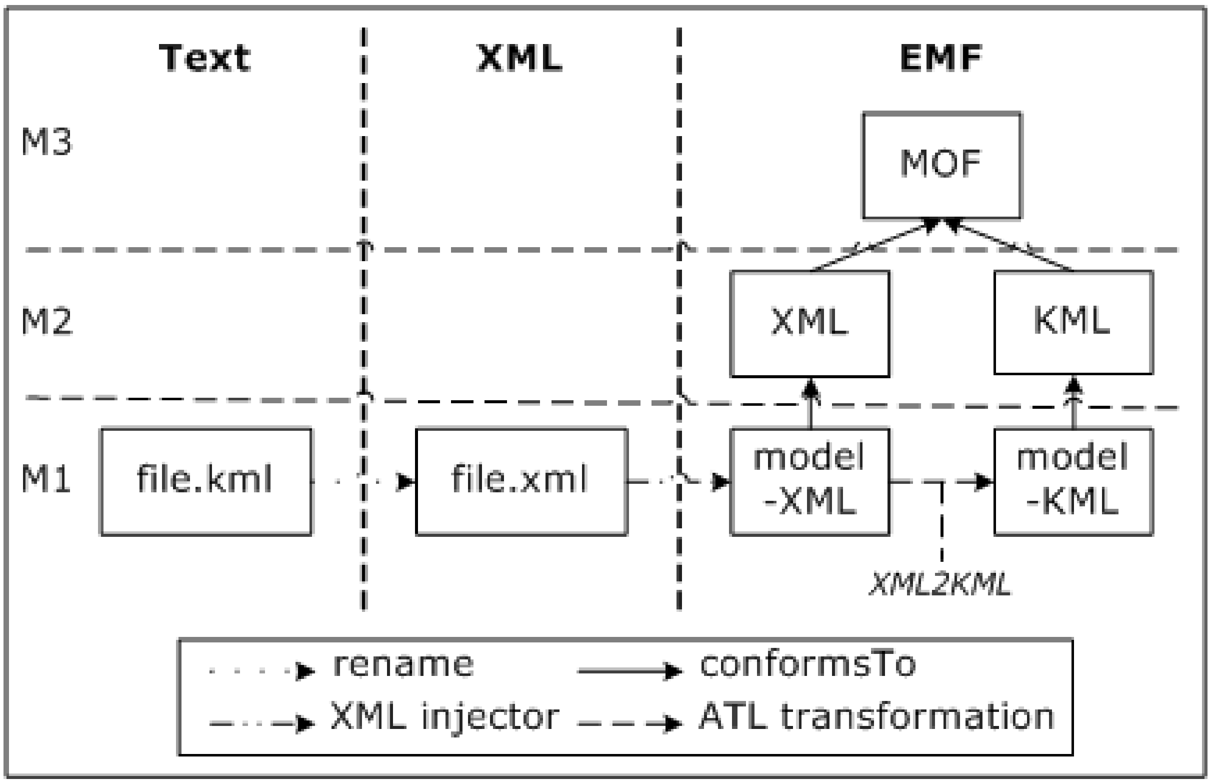

Figure 1: Overview of the transformation

1.2 Metamodel

1.2.1 KML

Simplified KML metamodel is described in Figure 2, and provided in complete version in Appendix A in KM3 format.

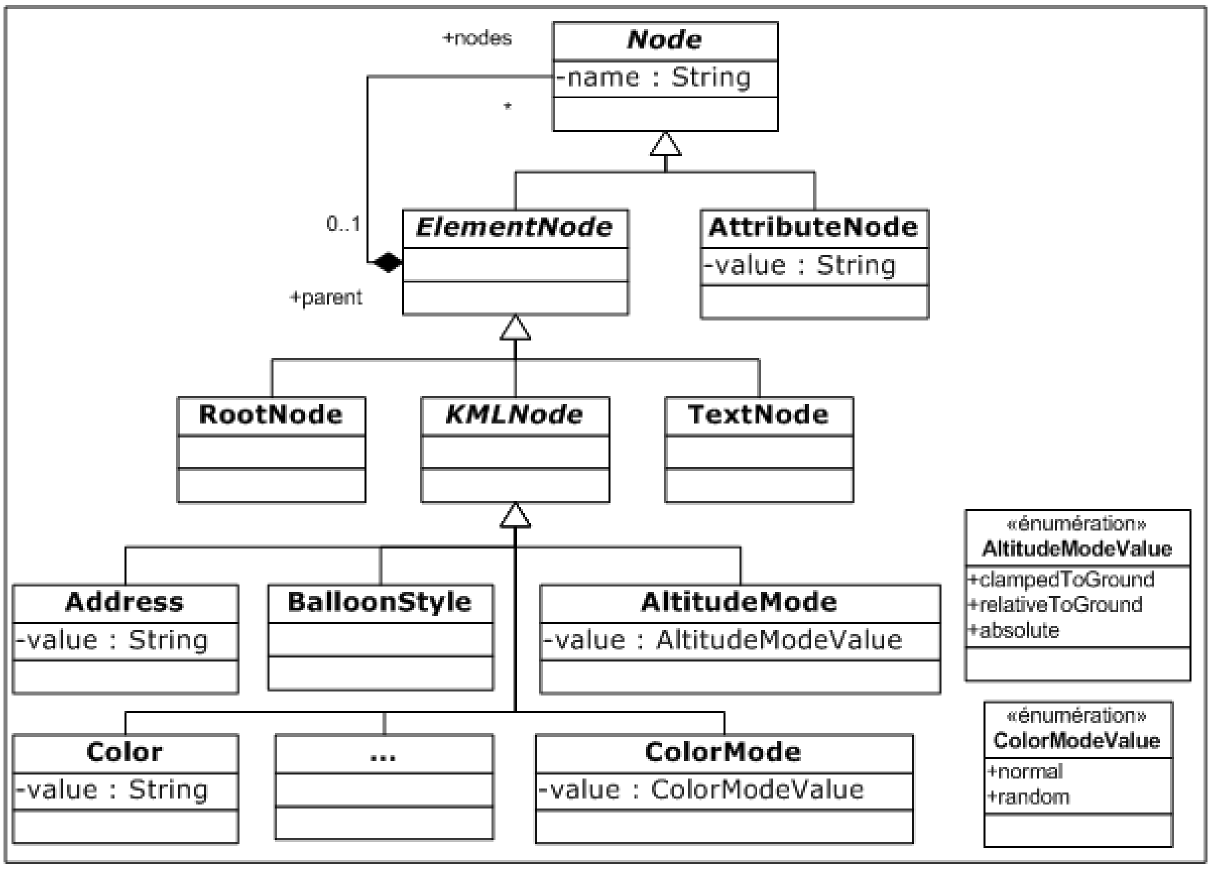

Figure 2: Simplified KML metamodel

This metamodel represents KML language in his version 2.0 [1]. KML is an XML-based language so the base of this metamodel cames from XML metamodel. The difference is that a KMLNode is extends for each tag of the language (on the figure, only some of them are represented).

1.3 Injector

1.3.1 Rules specification

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

- For each XML element, an element, determinates by XML element name, which extends KMLNode class is created. For instance, an AltitudeMode element will be created for an XML element named "altitudeMode". The created elements for the XML children relationship are transferred to KML nodes relationship.

1.3.2 ATL code

This ATL code for the XML2KML transformation consists in 5 helper and 84 rules.

The helper getAttrVal returns the value of the first Attribute element which name is given.

The helper getChildren returns all the children which belong to the sequence of possible children. This is useful to respect the KML Tag Dictionary.

The helper getText returns a sequence of valid Text elements (non empty and valid characters). And the helper getAllText return a String with the concatenation of all the value of these valid Text elements.

The helper getValue returns the value of the first Text element.

Each rule allocates an element which extends KMLNode class, for each Element element which name determinates the output element created. If the element needs a value, helper, String conversion or possible literal are chosen to do this.

```
-- @name XML2KML
-- @version 1.0
-- @domains KML, Google Earth, Geospatial data
-- @authors Eric Vepa (eric.vepa <at> gmail.com)
-- @date 2006/06/30
-- @description XML to KML transformation for
-- @see KML specifications  : http://earth.google.com/kml/kml_tags.html

module XML2KML; -- Module Template
create OUT : KML from IN : XML;

--@begin helper getAttrVal : returns the value of the first attribute named
    name
helper context XML!Element def : getAttrVal(name : String) : String =
  if self.children->select (c|c.oclIsTypeOf(XML!Attribute))->select(c|c.name =
      name)->notEmpty()
    then self.children->select (c|c.oclIsTypeOf(XML!Attribute))->select(c|c.
        name = name)->first().value
    else ''
  endif;
--@end helper getAttrVal

--@begin helper getChildren : returns all the children which belong to the
    sequence of possible children
helper context XML!Element def : getChildren(seq : Sequence(String)) : Sequence
    (XML!Element) =
  self.children->select(e|e.oclIsKindOf(XML!Element))->select(e|seq->includes(e
      .name));
--@end helper getChildren

--@begin helper getText : returns all valid text nodes
helper context XML!Element def : getText() : Sequence(XML!Text) =
  self.children->select(t|t.oclIsKindOf(XML!Text))->select(t|t.name = '#text')
      ->select(t|t.value.trim() <> '');
--@end helper getText

--@begin helper getAllText : returns a String containing all text nodes
helper context XML!Element def : getAllText() : String =
  self.getText()->iterate(s ; str : String = '' | str.concat(s.value.trim()).
      concat(' '));
--@end helper getAllText

--@begin helper getValue : returns the value of the first text node
helper context XML!Element def : getValue() : String =
  self.getText()->first().value.trim();
--@end helper getValue


--@begin rule Address
rule Address {
  from e:XML!Element (
    e.name = 'address'
  )
  to adr:KML!Address (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule Address

--@begin rule AltitudeMode
rule AltitudeMode {
  from e:XML!Element (
    e.name = 'altitudeMode'
  )
  to altMode:KML!AltitudeMode (
    name <- e.name,
    value <- if e.getValue() = 'relativeToGround'
          then #relativeToGround
          else if e.getValue() = 'absolute'
              then #absolute
              else #clampedToGround
            endif
          endif
  )
}
--@end rule AltitudeMode

--@begin rule BalloonStyle
rule BalloonStyle {
  from e:XML!Element (
    e.name = 'BalloonStyle'
  )
  to ball:KML!BalloonStyle (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'text', 'textColor', 'color'})
  )
}
--@end rule BalloonStyle

--@begin rule Color
rule Color {
  from e:XML!Element (
    e.name = 'color'
  )
  to color:KML!Color (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule Color

--@begin rule ColorMode
rule ColorMode {
  from e:XML!Element (
    e.name = 'colorMode'
  )
  to colMode:KML!ColorMode (
    name <- e.name,
    value <- if e.getValue() = 'random'
          then #random
          else #normal
         endif
  )
}
--@end rule ColorMode

--@begin rule Cookie
rule Cookie {
  from e:XML!Element (
    e.name = 'cookie'
  )
  to cook:KML!Cookie (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule Cookie

--@begin rule Coordinates
rule Coordinates {
  from e:XML!Element (
    e.name = 'coordinates'
  )
  to coords:KML!Coordinates (
    name <- e.name,
    nodes <- coord
  ),
  coord: distinct KML!Coordinate foreach (c in e.getText()) (
    name <- c.value.trim()
  )
}
--@end rule Coordinates

--@begin rule Description
rule Description {
  from e:XML!Element (
    e.name = 'description'
  )
  to desc:KML!Description (
    name <- e.name,
    value <- e.getAllText()
  )
}
--@end rule Description

--@begin rule Document
rule Document {
  from e:XML!Element (
    e.name = 'Document'
  )
  to doc:KML!Document (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'description', 'Document', 'Folder', '
        GroundOverlay', 'name', 'LookAt', 'NetworkLink', 'NetworkLinkControl', '
        Placemark', 'ScreenOverlay', 'Style', 'StyleMap', 'visibility'})
  )
}
--@end rule Document

--@begin rule DrawOrder
rule DrawOrder {
  from e:XML!Element (
    e.name = 'drawOrder'
  )
  to dOrder:KML!DrawOrder (
    name <- e.name,
    value <- e.getValue().toInteger()
  )
}
--@end rule DrawOrder

--@begin rule East
rule East {
  from e:XML!Element (
    e.name = 'east'
  )
  to east:KML!East (
    name <- e.name,
    value <- e.getValue().toReal()
  )
}
--@end rule East

--@begin rule Extrude
rule Extrude {
  from e:XML!Element (
    e.name = 'extrude'
  )
  to extr:KML!Extrude (
    name <- e.name,
    value <- if e.getValue() = '1'
           then true
           else false
         endif
  )
}
--@end rule Extrude

--@begin rule Fill
rule Fill {
  from e:XML!Element (
    e.name = 'fill'
  )
  to fill:KML!Fill (
    name <- e.name,
    value <- if e.getValue() = '0'
           then false
           else true
         endif
  )
}
--@end rule Fill

--@begin rule Folder
rule Folder {
  from e:XML!Element (
    e.name = 'Folder'
  )
  to fd:KML!Folder (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'description', 'Document', 'Folder', '
        GroundOverlay', 'LookAt', 'name', 'NetworkLink', 'open', 'Placemark', '
        ScreenOverlay', 'Style', 'visibility'})
  )
}
--@end rule Folder

--@begin rule GroundOverlay
rule GroundOverlay {
  from e:XML!Element (
    e.name = 'GroundOverlay'
  )
  to go:KML!GroundOverlay (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'description', 'color', 'drawOrder', 'Icon
        ', 'LatLonBox', 'LookAt', 'name', 'visibility'})
  )
}
--@end rule GroundOverlay

--@begin rule H
rule H {
  from e:XML!Element (
    e.name = 'h'
  )
  to h:KML!H (
    name <- e.name,
    value <- e.getValue().toInteger()
  )
}
--@end rule H

--@begin rule Heading
rule Heading {
  from e:XML!Element (
    e.name = 'heading'
  )
  to hding:KML!Heading (
    name <- e.name,
    value <- e.getValue().toReal()
  )
}
--@end rule Heading

--@begin rule Href
rule Href {
  from e:XML!Element (
    e.name = 'href'
  )
  to href:KML!Href (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule Href

--@begin rule Icon
rule Icon {
  from e:XML!Element (
    e.name = 'Icon'
  )
  to icon:KML!Icon (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'h', 'href', 'refreshMode', '
        viewRefreshMode', 'w', 'x', 'y'})
  )
}
--@end rule Icon

--@begin rule IconStyle
rule IconStyle {
  from e:XML!Element (
    e.name = 'IconStyle'
  )
  to iconS:KML!IconStyle (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'color', 'colorMode', 'heading', 'Icon', '
        scale'})
  )
}
```

```
--@end rule IconStyle

--@begin rule InnerBoundaryIs
rule InnerBoundaryIs {
  from e:XML!Element (
    e.name = 'innerBoundaryIs'
  )
  to inner:KML!InnerBoundaryIs (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'LinearRing'})
  )
}
--@end rule InnerBoundaryIs

--@begin rule Key
rule Key {
  from e:XML!Element (
    e.name = 'key'
  )
  to key:KML!Key (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule Key

--@begin rule LabelStyle
rule LabelStyle {
  from e:XML!Element (
    e.name = 'LabelStyle'
  )
  to labelS:KML!LabelStyle (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'color', 'colorMode', 'scale'})
  )
}
--@end rule LabelStyle

--@begin rule Latitude
rule Latitude {
  from e:XML!Element (
    e.name = 'latitude'
  )
  to lat:KML!Latitude (
    name <- e.name,
    value <- e.getValue().toReal()
  )
}
--@end rule Latitude

--@begin rule LatLonBox
rule LatLonBox {
  from e:XML!Element (
    e.name = 'LatLonBox'
  )
  to llb:KML!LatLonBox (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'east', 'west', 'north', 'south', 'rotation
        '})
  )
}
--@end rule LatLonBox

--@begin rule LinearRing
rule LinearRing {
  from e:XML!Element (
    e.name = 'LinearRing'
  )
  to lr:KML!LinearRing (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'coordinates'})
  )
}
--@end rule LinearRing

--@begin rule LineString
rule LineString {
  from e:XML!Element (
    e.name = 'LineString'
  )
  to ls:KML!LineString (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'coordinates', 'tessellate', 'extrude', '
        altitudeMode'})
  )
}
--@end rule LineString

--@begin rule LineStyle
rule LineStyle {
  from e:XML!Element (
    e.name = 'LineStyle'
  )
  to lineS:KML!LineStyle (
    name <- e.name,
    id <- e.getAttrVal('id'),
    nodes <- e.getChildren(Sequence{'color', 'colorMode', 'width'})
  )
}
--@end rule LineStyle

--@begin rule LinkDescription
rule LinkDescription {
  from e:XML!Element (
    e.name = 'linkDescription'
  )
  to lDesc:KML!LinkDescription (
    name <- e.name,
    value <- e.getAllText()
  )
}
--@end rule LinkDescription

--@begin rule LinkName
```

```
rule LinkName {
  from e:XML!Element (
    e.name = 'LinkName'
  )
  to lName:KML!LinkName (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule LinkName

--@begin rule Longitude
rule Longitude {
  from e:XML!Element (
    e.name = 'longitude'
  )
  to long:KML!Longitude (
    name <- e.name,
    value <- e.getValue().toReal()
  )
}
--@end rule Longitude

--@begin rule LookAt
rule LookAt {
  from e:XML!Element (
    e.name = 'LookAt'
  )
  to la:KML!LookAt (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'heading', 'latitude', 'longitude', 'range
        ', 'tilt'})
  )
}
--@end rule LookAt

--@begin rule Message
rule Message {
  from e:XML!Element (
    e.name = 'message'
  )
  to msg:KML!Message (
    name <- e.name,
    nodes <- text
  ),
  text: distinct KML!TextNode foreach (txt in e.getText()) (
    name <- txt.value
  )
}
--@end rule Message

--@begin rule MinRefreshPeriod
rule MinRefreshPeriod {
  from e:XML!Element (
    e.name = 'minRefreshPeriod'
  )
  to minRP:KML!MinRefreshPeriod (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule MinRefreshPeriod

--@begin rule MultiGeometry
rule MultiGeometry {
  from e:XML!Element (
    e.name = 'MultiGeometry'
  )
  to mgeo:KML!MultiGeometry (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'LineString', 'MultiGeometry', 'Point', '
       Polygon'})
  )
}
--@end rule MultiGeometry

--@begin rule Name
rule Name {
  from e:XML!Element (
    e.name = 'name'
  )
  to name:KML!Name (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule Name

--@begin rule NetworkLink
rule NetworkLink {
  from e:XML!Element (
    e.name = 'NetworkLink'
  )
  to netLink:KML!NetworkLink (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'description', 'name', 'refreshVisibility',
        'Url', 'visibility'}),
    nodes <- ftv
  ),
  -- FlyToView
  ftv:KML!FlyToView (
    name <- 'flyToView',
    value <- if e.getChildren(Sequence{'flyToView'})->notEmpty()
          then if e.getChildren(Sequence{'flyToView'})->first().getValue() =
              '1'
              then true
              else false
             endif
          else false
         endif
  )
  -- FlyToView
}
--@end rule NetworkLink
```

```
--@begin rule NetworkLinkControl
rule NetworkLinkControl {
  from e:XML!Element (
    e.name = 'NetworkLinkControl'
  )
  to netLinCtrlk:KML!NetworkLinkControl (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'cookie', 'linkDescription', 'linkName', '
        message', 'minRefreshPeriod'})
  )
}
--@end rule NetworkLinkControl

--@begin rule North
rule North {
  from e:XML!Element (
    e.name = 'north'
  )
  to north:KML!North (
    name <- e.name,
    value <- e.getValue().toReal()
  )
}
--@end rule North

--@begin rule ObjArrayField
rule ObjArrayField {
  from e:XML!Element (
    e.name = 'ObjArrayField'
  )
  to objAF:KML!ObjArrayField (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'name', 'type'})
  )
}
--@end rule ObjArrayField

--@begin rule ObjField
rule ObjField {
  from e:XML!Element (
    e.name = 'ObjField'
  )
  to objF:KML!ObjField (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'name', 'type'})
  )
}
--@end rule ObjField

--@begin rule Open
rule Open {
  from e:XML!Element (
    e.name = 'open'
  )
  to open:KML!Open (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule Open

--@begin rule OuterBoundaryIs
rule OuterBoundaryIs {
  from e:XML!Element (
    e.name = 'outerBoundaryIs'
  )
  to outerBI:KML!OuterBoundaryIs (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'LinearRing'})
  )
}
--@end rule OuterBoundaryIs

--@begin rule Outline
rule Outline {
  from e:XML!Element (
    e.name = 'outline'
  )
  to out:KML!Outline (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule Outline

--@begin rule OverlayXY
rule OverlayXY {
  from e:XML!Element (
    e.name = 'overlayXY'
  )
  to over:KML!OverlayXY (
    name <- e.name,
    x <- e.getAttrVal('x'),
    y <- e.getAttrVal('y'),
    xunits <- e.getAttrVal('xunits'),
    yunits <- e.getAttrVal('yunits')
  )
}
--@end rule OverlayXY

--@begin rule Pair
rule Pair {
  from e:XML!Element (
    e.name = 'Pair'
  )
  to pair:KML!Pair (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'key', 'styleUrl'})
  )
}
--@end rule Pair

--@begin rule Parent
```

```
rule Parent {
  from e:XML!Element (
    e.name = 'parent'
  )
  to parent:KML!Parent (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule Parent

--@begin rule Placemark
rule Placemark {
  from e:XML!Element (
    e.name = 'Placemark'
  )
  to pmk:KML!Placemark (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'address', 'description', 'LookAt', '
        LineString', 'MultiGeometry', 'name', 'Point', 'Polygon', 'Snippet', '
        Style', 'styleUrl', 'visibility'})
  )
}
--@end rule Placemark

--@begin rule Point
rule Point {
  from e:XML!Element (
    e.name = 'Point'
  )
  to pt:KML!Point (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'coordinates', 'tessallate', 'extrude', '
        altitudeMode'})
  )
}
--@end rule Point

--@begin rule Polygon
rule Polygon {
  from e:XML!Element (
    e.name = 'Polygon'
  )
  to poly:KML!Polygon (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'outerBoundaryIs', 'innerBoundaryIs', '
        tessellate', 'extrude', 'altitudeMode'})
  )
}
--@end rule Polygon

--@begin rule PolyStyle
rule PolyStyle {
  from e:XML!Element (
    e.name = 'PolyStyle'
  )
  to polyS:KML!PolyStyle (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'color', 'colorMode', 'fill', 'outline'})
  )
}
--@end rule PolyStyle

--@begin rule Range
rule Range {
  from e:XML!Element (
    e.name = 'range'
  )
  to rng:KML!Range (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule Range

--@begin rule RefreshInterval
rule RefreshInterval {
  from e:XML!Element (
    e.name = 'refreshInterval'
  )
  to refrI:KML!RefreshInterval (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule RefreshInterval

--@begin rule RefreshMode
rule RefreshMode {
  from e:XML!Element (
    e.name = 'refreshMode'
  )
  to refrM:KML!RefreshMode (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule RefreshMode

--@begin rule RefreshVisibility
rule RefreshVisibility {
  from e:XML!Element (
    e.name = 'refreshVisibility'
  )
  to refrV:KML!RefreshVisibility (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule RefreshVisibility

--@begin rule Rotation
rule Rotation {
  from e:XML!Element (
    e.name = 'rotation'
  )
  to rot:KML!Rotation (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule Rotation

--@begin rule Schema
rule Schema {
  from e:XML!Element (
    e.name = 'Schema'
  )
  to schema:KML!Schema (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'name', 'parent', 'ObjField', '
        ObjArrayField', 'SimpleField', 'SimpleArrayField'})
  )
}
--@end rule Schema

--@begin rule Scale
rule Scale {
  from e:XML!Element (
    e.name = 'scale'
  )
  to scale:KML!Scale (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule Scale

--@begin rule ScreenOverlay
rule ScreenOverlay {
  from e:XML!Element (
    e.name = 'ScreenOverlay'
  )
  to scrO:KML!ScreenOverlay (
    name <- e.name,
    id <- e.getAttrVal('id'),
    nodes <- e.getChildren(Sequence{'description', 'drawOrder', 'Icon', '
        overlayXY', 'rotation', 'screenXY', 'size', 'name', 'visibility'})
  )
}
--@end rule ScreenOverlay

--@begin rule ScreenXY
rule ScreenXY {
  from e:XML!Element (
    e.name = 'screenXY'
  )
  to scrXY:KML!ScreenXY (
    name <- e.name,
    x <- e.getAttrVal('x'),
    y <- e.getAttrVal('y'),
    xunits <- e.getAttrVal('xunits'),
    yunits <- e.getAttrVal('yunits')
  )
}
--@end rule ScreenXY

--@begin rule SimpleArrayField
rule SimpleArrayField {
  from e:XML!Element (
    e.name = 'SimpleArrayField'
  )
  to sAF:KML!SimpleArrayField (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'name', 'type'})
  )
}
--@end rule SimpleArrayField

--@begin rule SimpleField
rule SimpleField {
  from e:XML!Element (
    e.name = 'SimpleField'
  )
  to sF:KML!SimpleField (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'name', 'type'})
  )
}
--@end rule SimpleField

--@begin rule Size
rule Size {
  from e:XML!Element (
    e.name = 'size'
  )
  to size:KML!Size (
    name <- e.name,
    x <- e.getAttrVal('x'),
    y <- e.getAttrVal('y'),
    xunits <- e.getAttrVal('xunits'),
    yunits <- e.getAttrVal('yunits')
  )
}
--@end rule Size

--@begin rule South
rule South {
  from e:XML!Element (
    e.name = 'south'
  )
  to south:KML!South (
    name <- e.name,
    value <- e.getValue().toReal()
  )
}
--@end rule South
```

```
--@begin rule Snippet
rule Snippet {
  from e:XML!Element (
    e.name = 'Snippet'
  )
  to snip:KML!Snippet (
    name <- e.name,
    nodes <- text
  ),
  text: distinct KML!TextNode foreach (txt in e.getText()) (
    name <- txt.value
  )
}
--@end rule Snippet

--@begin rule Style
rule Style {
  from e:XML!Element (
    e.name = 'Style'
  )
  to style:KML!Style (
    name <- e.name,
    id <- e.getAttrVal('id'),
    nodes <- e.getChildren(Sequence{'BalloonStyle', 'IconStyle', 'LabelStyle',
        'LineStyle', 'PolyStyle'})
  )
}
--@end rule Style

--@begin rule StyleMap
rule StyleMap {
  from e:XML!Element (
    e.name = 'StyleMap'
  )
  to sMap:KML!StyleMap (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'Pair'})
  )
}
--@end rule StyleMap

--@begin rule StyleUrl
rule StyleUrl {
  from e:XML!Element (
    e.name = 'styleUrl'
  )
  to sURL:KML!StyleUrl (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule StyleUrl

--@begin rule Tessellate
rule Tessellate {
  from e:XML!Element (
    e.name = 'tessellate'
  )
  to tess:KML!Tessellate (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule Tessellate

--@begin rule Text
rule Text {
  from e:XML!Element (
    e.name = 'text'
  )
  to txt:KML!Text (
    name <- e.name,
    nodes <- text,
    nodes <- e.getChildren(Sequence{'textColor'})
  ),
  text: distinct KML!TextNode foreach (txt in e.getText()) (
    name <- txt.value
  )
}
--@end rule Text

--@begin rule TextColor
rule TextColor {
  from e:XML!Element (
    e.name = 'textColor'
  )
  to txtColor:KML!TextColor (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule TextColor

--@begin rule Tilt
rule Tilt {
  from e:XML!Element (
    e.name = 'tilt'
  )
  to tilt:KML!Tilt (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule Tilt

--@begin rule Type
rule Type {
  from e:XML!Element (
    e.name = 'type'
  )
  to type:KML!Type (
    name <- e.name,
    value <- e.getValue()
  )
```

```
}
--@end rule Type

--@begin rule Url
rule Url {
  from e:XML!Element (
    e.name = 'Url'
  )
  to url:KML!Url (
    name <- e.name,
    nodes <- e.getChildren(Sequence{'href', 'refreshInterval', 'refreshMode', '
        ViewFormat', 'viewRefreshMode', 'viewRefreshTime'})
  )
}
--@end rule Url

--@begin rule ViewBoundScale
rule ViewBoundScale {
  from e:XML!Element (
    e.name = 'viewBoundScale'
  )
  to viewBS:KML!ViewBoundScale (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule ViewBoundScale

--@begin rule ViewRefreshMode
rule ViewRefreshMode {
  from e:XML!Element (
    e.name = 'viewRefreshMode'
  )
  to viewRM:KML!ViewRefreshMode (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule ViewRefreshMode

--@begin rule ViewRefreshTime
rule ViewRefreshTime {
  from e:XML!Element (
    e.name = 'viewRefreshTime'
  )
  to viewRT:KML!ViewRefreshTime (
    name <- e.name,
    value <- e.getValue().toInteger()
  )
}
--@end rule ViewRefreshTime

--@begin rule ViewFormat
rule ViewFormat {
  from e:XML!Element (
    e.name = 'viewFormat'
  )
  to viewF:KML!ViewFormat (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule ViewFormat

--@begin rule Visibility
rule Visibility {
  from e:XML!Element (
    e.name = 'visibility'
  )
  to vis:KML!Visibility (
    name <- e.name,
    value <- e.getValue()
  )
}
--@end rule Visibility

--@begin rule W
rule W {
  from e:XML!Element (
    e.name = 'w'
  )
  to w:KML!W (
    name <- e.name,
    value <- e.getValue().toInteger()
  )
}
--@end rule W

--@begin rule West
rule West {
  from e:XML!Element (
    e.name = 'west'
  )
  to west:KML!West (
    name <- e.name,
    value <- e.getValue().toReal()
  )
}
--@end rule West

--@begin rule Width
rule Width {
  from e:XML!Element (
    e.name = 'width'
  )
  to width:KML!Width (
    name <- e.name,
    value <- e.getValue().toReal()
  )
}
--@end rule Width

--@begin rule X
rule X {
  from e:XML!Element (
    e.name = 'x'
  )
  to x:KML!X (
    name <- e.name,
    value <- e.getValue().toInteger()
  )
}
--@end rule X

--@begin rule Y
rule Y {
  from e:XML!Element (
    e.name = 'y'
  )
  to y:KML!Y (
    name <- e.name,
    value <- e.getValue().toInteger()
  )
}
--@end rule Y
```

2 References

[1] KML 2.0 Tag Dictionary (http://earth.google.com/kml/kml_tags.html).

A Appendix: KML metamodel in KM3 format

```
-- @name KML 2.0 (Keyhole Markup Language)
-- @version 1.0
-- @domains KML, Google Earth, Geospatial data
-- @authors Eric Vepa (eric.vepa <at> gmail.com)
-- @date 2006/06/30
-- @description KML (Keyhole Markup Language) is an XML-based language for
    managing three-dimensionageospatial data in the program Google Earth
-- @see KML specifications : http://earth.google.com/kml/kml_tags.html
-- @comments Each KML tag is represented as a class, details for each class are
     in the specifications (see the previous link).

--@begin package Keyhole Markup Language
package KML {
  --@begin class Node
  abstract class Node {
      attribute name : String;
    reference parentNode[0-1] : ElementNode oppositeOf nodes;
  }
  --@end class Node

  --@begin class ElementNode
  abstract class ElementNode extends Node {
    reference nodes[*] ordered container : Node oppositeOf parentNode;
  }
  --@end class ElementNode

  --@begin class AttributeNode
  class AttributeNode extends Node {
    attribute value : String;
  }
  --@end class AttributeNode


  --@begin class RootNode
  abstract class RootNode extends ElementNode {}
  --@end class RootNode


  --@begin class TextNode
  class TextNode extends ElementNode {}
  --@end class TextNode

  --@begin class KMLNode
  abstract class KMLNode extends ElementNode {}
  --@end class KMLNode

  --@begin class KMLRootNode
  class KMLRootNode extends RootNode {}
  --@end class KMLRootNode
```

```
--@begin class Address
class Address extends KMLNode {
  attribute value : String;
}
--@end class Address

--@begin class AltitudeMode
class AltitudeMode extends KMLNode {
  attribute value : AltitudeModeValue;
}
--@end class AltitudeMode

--@begin enumeration AltitudeModeValue
enumeration AltitudeModeValue {
  literal clampedToGround;
  literal relativeToGround;
  literal absolute;
}
--@end enumeration AltitudeModeValue

--@begin class BalloonStyle
class BalloonStyle extends KMLNode {}
--@end class BalloonStyle

--@begin class Color
class Color extends KMLNode {
  attribute value : String;
}
--@end class Color

--@begin class ColorMode
class ColorMode extends KMLNode {
  attribute value : ColorModeValue;
}
--@end class ColorMode

--@begin enumeration ColorModeValue
enumeration ColorModeValue {
  literal normal;
  literal random;
}
--@end enumeration ColorModeValue

--@begin class Cookie
class Cookie extends KMLNode {
  attribute value : String;
}
--@end class Cookie

--@begin class Coordinates
class Coordinates extends KMLNode {}
--@end class Coordinates

--@begin class Coordinate
class Coordinate extends KMLNode {}
```

```
--@end class Coordinate

--@begin class Description
class Description extends KMLNode {
  attribute value : String;
}
--@end class Description

--@begin class Document
class Document extends KMLNode {}
--@end class Document

--@begin class DrawOrder
class DrawOrder extends KMLNode {
  attribute value : Integer;
}
--@end class DrawOrder

--@begin class East
class East extends KMLNode {
  attribute value : Double;
}
--@end class East

--@begin class Extrude
class Extrude extends KMLNode {
  attribute value : Boolean;
}
--@end class Extrude

--@begin class Fill
class Fill extends KMLNode {
  attribute value : Boolean;
}
--@end class Fill

--@begin class FlyToView
--@comments always present in a NetworkLink (default value to false)
class FlyToView extends KMLNode {
  attribute value : Boolean;
}
--@end class FlyToView

--@begin class Folder
class Folder extends KMLNode {}
--@end class Folder

--@begin class GroundOverlay
class GroundOverlay extends KMLNode {}
--@end class GroundOverlay

--@begin class H
class H extends KMLNode {
  attribute value : Integer;
}
--@end class H
```

```
--@begin class Heading
class Heading extends KMLNode {
  attribute value : Double;
}
--@end class Heading

--@begin class Href
class Href extends KMLNode {
  attribute value : String;
}
--@end class Href

--@begin class Icon
class Icon extends KMLNode {}
--@end class Icon

--@begin class IconStyle
class IconStyle extends KMLNode {}
--@end class IconStyle

--@begin class InnerBoundaryIs
class InnerBoundaryIs extends KMLNode {}
--@end class InnerBoundaryIs

--@begin class Key
class Key extends KMLNode {
  attribute value : String;
}
--@end class Key

--@begin class LabelStyle
class LabelStyle extends KMLNode {}
--@end class LabelStyle

--@begin class Latitude
class Latitude extends KMLNode {
  attribute value : Double;
}
--@end class Latitude

--@begin class LatLonBox
class LatLonBox extends KMLNode {}
--@end class LatLonBox

--@begin class LinearRing
class LinearRing extends KMLNode {}
--@end class LinearRing

--@begin class LineString
class LineString extends KMLNode {}
--@end class LineString

--@begin class LineStyle
class LineStyle extends KMLNode {
  attribute id : String;
}
--@end class LineStyle
```

```
--@begin class LinkDescription
class LinkDescription extends KMLNode {
  attribute value : String;
}
--@end class LinkDescription

--@begin class LinkName
class LinkName extends KMLNode {
  attribute value : String;
}
--@end class LinkName

--@begin class Longitude
class Longitude extends KMLNode {
  attribute value : Double;
}
--@end class Longitude

--@begin class LookAt
class LookAt extends KMLNode {}
--@end class LookAt

--@begin class Message
class Message extends KMLNode {}
--@end class Message

--@begin class MinRefreshPeriod
class MinRefreshPeriod extends KMLNode {
  attribute value : String;
}
--@end class MinRefreshPeriod

--@begin class MultiGeometry
class MultiGeometry extends KMLNode {}
--@end class MultiGeometry

--@begin class name
class Name extends KMLNode {
  attribute value : String;
}
--@end class name

--@begin class NetworkLink
class NetworkLink extends KMLNode {}
--@end class NetworkLink

--@begin class NetworkLinkControl
class NetworkLinkControl extends KMLNode {}
--@end class NetworkLinkControl

--@begin class North
class North extends KMLNode {
  attribute value : Double;
}
--@end class North
```

```
--@begin class ObjArrayField
class ObjArrayField extends KMLNode {}
--@end class ObjArrayField

--@begin class ObjField
class ObjField extends KMLNode {}
--@end class ObjField

--@begin class Open
class Open extends KMLNode {
  attribute value : String;
}
--@end class Open

--@begin class OuterBoundaryIs
class OuterBoundaryIs extends KMLNode {}
--@end class OuterBoundaryIs

--@begin class Outline
class Outline extends KMLNode {
  attribute value : String;
}
--@end class Outline

--@begin class OverlayXY
class OverlayXY extends KMLNode {
  attribute x : String;
  attribute y : String;
  attribute xunits : String;
  attribute yunits : String;
}
--@end class OverlayXY

--@begin class Pair
class Pair extends KMLNode {}
--@end class Pair

--@begin class Parent
class Parent extends KMLNode {
  attribute value : String;
}
--@end class Parent

--@begin class Placemark
class Placemark extends KMLNode {}
--@end class Placemark

--@begin class Point
class Point extends KMLNode {}
--@end class Point

--@begin class Polygon
class Polygon extends KMLNode {}
--@end class Polygon

--@begin class PolyStyle
class PolyStyle extends KMLNode {}
```

```
--@end class PolyStyle

--@begin class Range
class Range extends KMLNode {
  attribute value : String;
}
--@end class Range

--@begin class RefreshInterval
class RefreshInterval extends KMLNode {
  attribute value : String;
}
--@end class RefreshInterval

--@begin class RefreshMode
class RefreshMode extends KMLNode {
  attribute value : String;
}
--@end class RefreshMode

--@begin class RefreshVisibility
class RefreshVisibility extends KMLNode {
  attribute value : String;
}
--@end class RefreshVisibility

--@begin class Rotation
class Rotation extends KMLNode {
  attribute value : String;
}
--@end class Rotation

--@begin class Schema
class Schema extends KMLNode {}
--@end class Schema

--@begin class Scale
class Scale extends KMLNode {
  attribute value : String;
}
--@end class Scale

--@begin class ScreenOverlay
class ScreenOverlay extends KMLNode {
  attribute id : String;
}
--@end class ScreenOverlay

--@begin class ScreenXY
class ScreenXY extends KMLNode {
  attribute x : String;
  attribute y : String;
  attribute xunits : String;
  attribute yunits : String;
}
--@end class ScreenXY
```

```
--@begin class SimpleArrayField
class SimpleArrayField extends KMLNode {}
--@end class SimpleArrayField

--@begin class SimpleField
class SimpleField extends KMLNode {}
--@end class SimpleField

--@begin class Size
class Size extends KMLNode {
  attribute x : String;
  attribute y : String;
  attribute xunits : String;
  attribute yunits : String;
}
--@end class Size

--@begin class South
class South extends KMLNode {
  attribute value : Double;
}
--@end class South

--@begin class Snippet
class Snippet extends KMLNode {}
--@end class Snippet

--@begin class Style
class Style extends KMLNode {
  attribute id : String;
}
--@end class Style

--@begin class StyleMap
class StyleMap extends KMLNode {}
--@end class StyleMap

--@begin class StyleUrl
class StyleUrl extends KMLNode {
  attribute value : String;
}
--@end class StyleUrl

--@begin class Tessellate
class Tessellate extends KMLNode {
  attribute value : String;
}
--@end class Tessellate

--@begin class Text
class Text extends KMLNode {}
--@end class Text

--@begin class TextColor
class TextColor extends KMLNode {
  attribute value : String;
}
```

```
--@end class TextColor

--@begin class Tilt
class Tilt extends KMLNode {
  attribute value : String;
}
--@end class Tilt

--@begin class Type
class Type extends KMLNode {
  attribute value : String;
}
--@end class Type

--@begin class Url
class Url extends KMLNode {}
--@end class Url

--@begin class ViewBoundScale
class ViewBoundScale extends KMLNode {
  attribute value : String;
}
--@end class ViewBoundScale

--@begin class ViewRefreshMode
class ViewRefreshMode extends KMLNode {
  attribute value : String;
}
--@end class ViewRefreshMode

--@begin class ViewRefreshTime
class ViewRefreshTime extends KMLNode {
  attribute value : Integer;
}
--@end class ViewRefreshTime

--@begin class ViewFormat
class ViewFormat extends KMLNode {
  attribute value : String;
}
--@end class ViewFormat

--@begin class Visibility
class Visibility extends KMLNode {
  attribute value : String;
}
--@end class Visibility

--@begin class W
class W extends KMLNode {
  attribute value : Integer;
}
--@end class W

--@begin class West
class West extends KMLNode {
  attribute value : Double;
  }
  --@end class West

  --@begin class Width
  class Width extends KMLNode {
    attribute value : Double;
  }
  --@end class Width

  --@begin class X
  class X extends KMLNode {
    attribute value : Integer;
  }
  --@end class X

  --@begin class Y
  class Y extends KMLNode {
    attribute value : Integer;
  }
  --@end class Y
}
--@end package Keyhole Markup Language

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