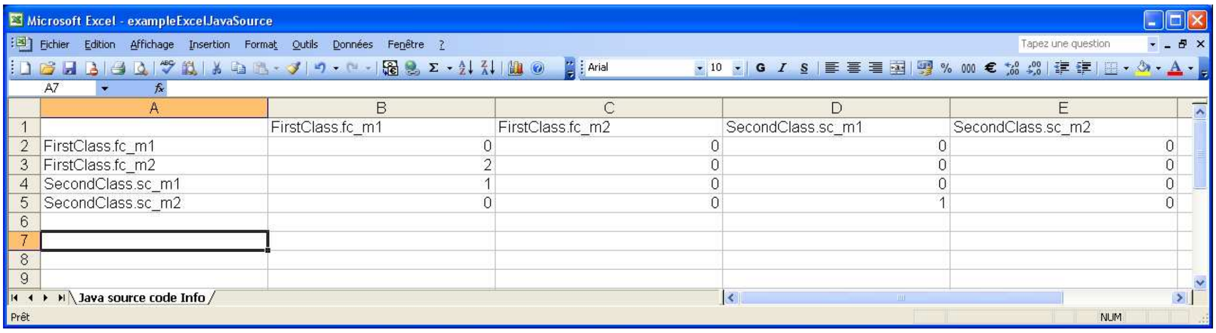
1. ATL Transformation Example

1.1. Example: Microsoft Office Excel Injector

The Microsoft Office Excel injector's example describes a transformation from an Excel workbook to an Excel model. The transformation is based on a simplified subset of the SpreadsheetML XML dialect which is the one used by Microsoft to import/export Excel workbook's data in XML since the 2003 version of Microsoft Office. This transformation produces an Excel model from an Excel XML file which can be directly opened by Excel 2003. This Excel model describes a workbook with the same content that the one stored into the Excel XML file in entry of the transformation.

1.1.1. Transformation overview

The aim of this injector (transformation) is to generate an Excel model that conforms to the SpreadsheetMLSimplified metamodel from an Excel workbook (contained in a valid and well-formed Excel XML file). As an example of the transformation, Figure 1 provides a screen capture of a simple Microsoft Office Excel workbook that may be transformed into a SpreadsheetMLSimplified model by the injector. Note that the Excel models generated by the injector will be able to be reused by other transformations that need input Excel models.


Microsoft Excel - exampleExcelJavaSource
Fichier Edition Affichage Insertion Format Outils Données Fenêtre ?
Tapez une question
Arial
10
A7
Java source code Info
Prêt
NUM

	A	B	C	D	E
1		FirstClass.fc_m1	FirstClass.fc_m2	SecondClass.sc_m1	SecondClass.sc_m2
2	FirstClass.fc_m1	0	0	0	0
3	FirstClass.fc_m2	2	0	0	0
4	SecondClass.sc_m1	1	0	0	0
5	SecondClass.sc_m2	0	0	1	0
6					
7					
8					
9					

Figure 1. The injected MS Office Excel workbook

To make the Microsoft Office Excel injector, we proceed in two steps. Indeed, this transformation is in reality a composition of two transformations:

- from Excel XML file to XML (XML injection)
- from XML to SpreadsheetMLSimplified

These two steps are summarized in Figure 2.

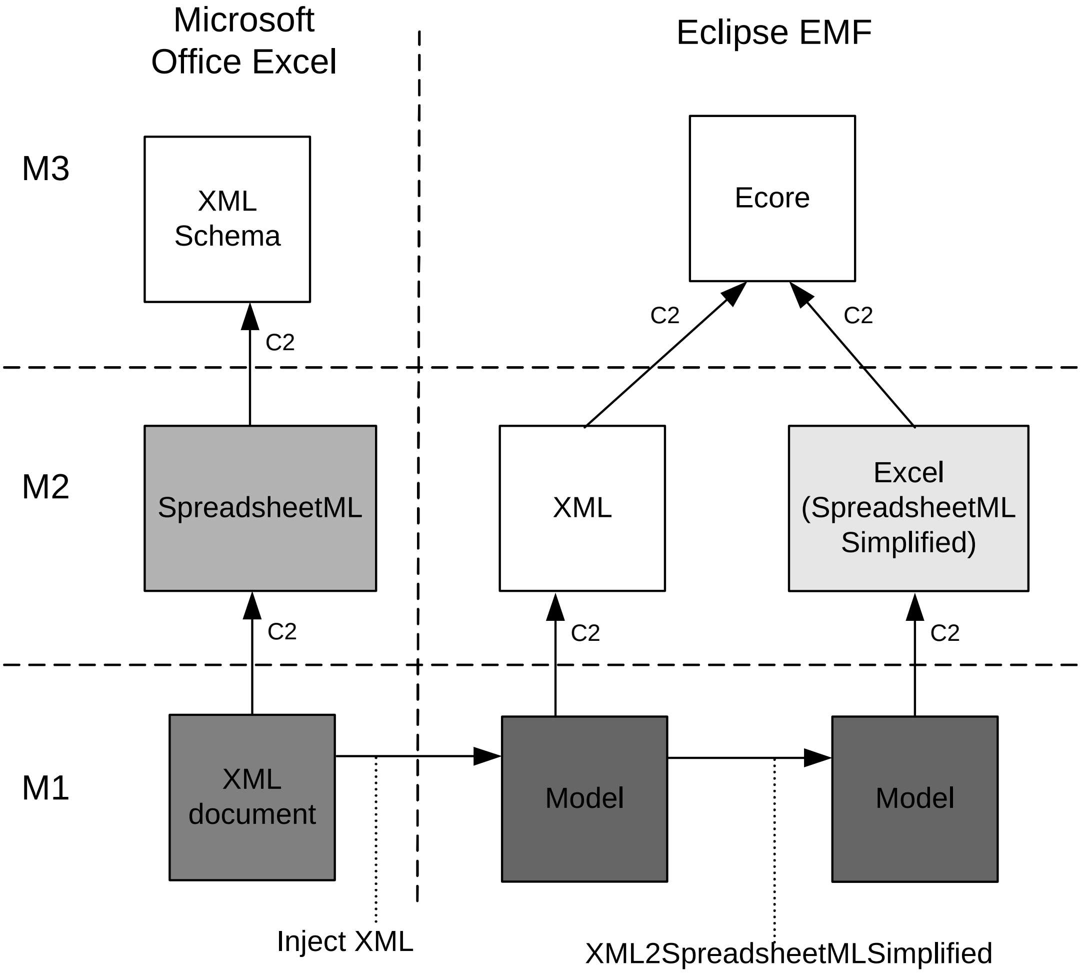


Figure 2. Microsoft Office Excel injector's (transformation's) overview

1.2. Metamodels

The first metamodel used by this transformation is a simple XML metamodel which is necessary to import XML files into XML models. This metamodel is presented in Figure 3 and provided in Appendix I in km3 format.

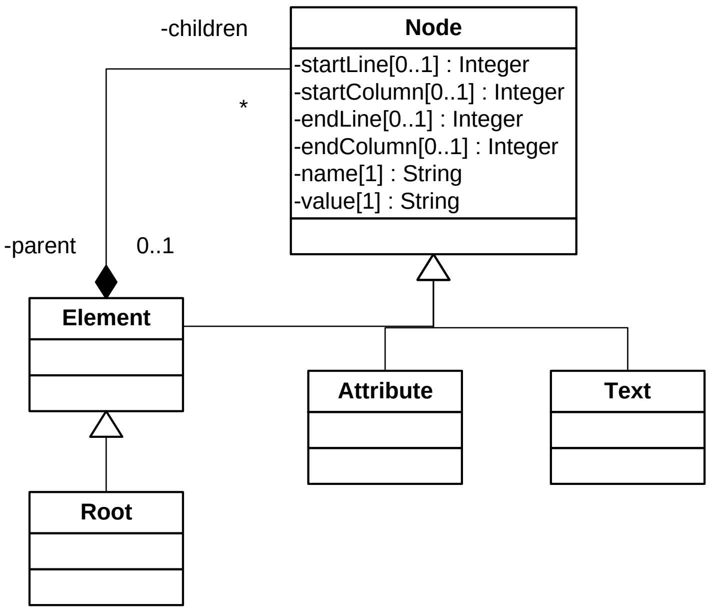


Figure 3. A simple XML metamodel

Each element of an XML document is a *Node*. The root of a document is a *Root* element which is an *Element* in our metamodel. Each *Element* can have several children (nodes) that can be other *Element*, *Attribute* or *Text* elements. An *Element* is usually identified by its name and defined by its children. An *Attribut*e is characterized by its name and its value whereas a *Text* is only assimilated to a single value.

The transformation is based on the “SpreadsheetMLSimplified” metamodel which is a subset of the Microsoft SpreadsheetML XML dialect defined by several complex XML schemas (they can be downloaded at [1]). The metamodel considered here is described in Figure 4 and provided in Appendix II in km3 format (note that some attributes of the metamodel have voluntarily not been mentioned in this figure in order to keep the diagram clear and easily readable).

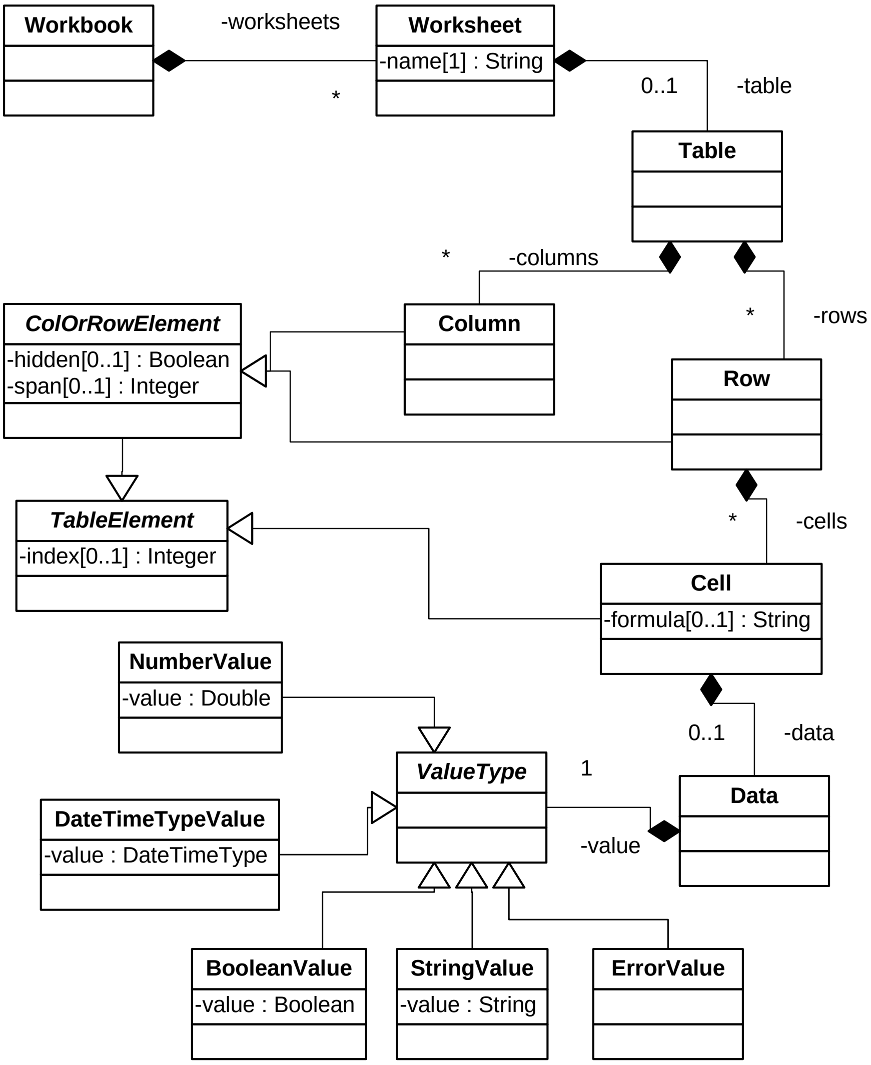


Figure 4. The SpreadsheetMLSimplified metamodel

Within this metamodel, a workbook is associated with a *Workbook* element. Such an element can contain several worksheets. A table is most of the time associated to each worksheet. A table is composed of a set of *TableElement*: columns and rows are contained in the table; cells are contained in the rows. Each cell can store a data in a particular type which can be “Number”, “DateTime”, ”Boolean”, ”String” or “Error”.

1.3. Rules Specification

The input of the global transformation is an Excel XML file whose content conforms to the SpreadsheetML schemas; the output is an Excel model which conforms to the SpreadsheetMLSimplified metamodel (described in Figure 4). The input XML model of the second transformation is the output XML model generated by the first transformation.

1.3.1. Excel XML file to XML (i.e. XML injection)

The XML injector (i.e. the “XML file to XML model” transformation) is already implemented by an ATL plug-in which is included in the ATL environment under Eclipse. This plug-in gives the possibility to inject the content of an XML file into an XML model which conforms to the simple XML metamodel presented in Figure 3. This is the reason why we will not spend to much time to detail this transformation in this documentation.

1.3.2. XML to SpreadsheetMLSimplified

These are the rules to transform an XML model into a SpreadsheetMLSimplified model:

- For the “workbook” *XML!Root*, the *SpreadsheetMLSimplified!Workbook* element is created. It will be linked to the corresponding *SpreadsheetMLSimplified!Worksheet* elements that will be generated during the transformation by the following rule.

- For each “worksheet” *XML!Element* (which is a child of the “workbook” *XML!Root*), a *SpreadsheetMLSimplified!Worksheet* element is generated. Each *Worksheet* element will be linked to the associated *SpreadsheetMLSimplified!Tabl*e element that will be created during the transformation by the following rule.

- For each “table” *XML!Element* (which is a child of a “worksheet” *XML!Element*), a *SpreadsheetMLSimplified!Table* element is created. Each *Table* element will be linked to the corresponding *SpreadsheetMLSimplified!Column* and *SpreadsheetMLSimplified!Row* elements that will be generated during the transformation by the following rule.

- For each “column” or “row” *XML!Element* (which is a child of a “table” *XML!Element*), a *SpreadsheetMLSimplified!Column* or *SpreadsheetMLSimplified!Row* element is generated. Each *Row* element will be linked to the corresponding *SpreadsheetMLSimplified!Cell* element that will be created during the transformation.

- For each “cell” *XML!Element* (which is a child of a “row” *XML!Element*), a *SpreadsheetMLSimplified!Cell* element is created.

- For each “data” *XML!Element* (which is a child of a “cell” *XML!Element*), a *SpreadsheetMLSimplified!Data* element is engendered and linked to the right *SpreadsheetMLSimplified!Cell* element. A *SpreadsheetMLSimplified!XXXValue* element (corresponding to the value of the “ss:Type” *XML!Attribute* of the “Data” *XML!Element*) is also created and linked to this *SpreadsheetMLSimplified!Data* element.

1.4. ATL Code

Since the XML injector is already integrated into the ATL environment, there is only one ATL file coding the transformation: "XML2SpreadsheetMLSimplified.atl". In this part we will present and describe more precisely the implementation of this transformation.

The ATL code for the "XML2SpreadsheetMLSimplified" transformation consists of 10 helpers and 11 rules.

The *getStringAttrValue* helper returns the string value of the XML!Attribute (of the XML!Element on which the helper is called) that has got as name the value of the "attrName" parameter. If there is no XML!Attribute named "attrName", an empty string is returned by this helper.

The *getChildrenByName* helper offers the possibility to recover a set of XML!Element that are all the children of the XML!Element on which the helper is called. All the returned children have got as name the value of the "name" parameter. This helper may return an empty set if there are no children of the XML!Element type and named "name".

The *getOptIntAttrValue* helper returns the integer value of an optional XML!Attribute which has got as name the value of the "attrName" parameter. This helper uses the *getStringAttrValue* one to recover the string value of the sought XML!Attribute. The result is converted into an integer value by a "String toInteger() : Integer" method's call only if the returned value is not an empty string (i.e. if the XML!Attribute exists). In the contrary case, the "void" value is assigned to the value returned by the helper.

The *getOptBoolAttrValue* helper is quite similar to the previously detailed *getOptIntAttrValue* one. However, instead of returning an integer value, it returns a boolean one (or the "void" value if the sought XML!Attribute does not exist).

The *getOptRealAttrValue* helper is also quite similar to the *getOptIntAttrValue* one. The only difference is that it returns a real value obtained thanks to a "String toReal() : Real" method's call. Obviously, the "void" value is also returned if the sought XML!Attribute does not exist.

The *getOptStringAttrValue* helper can be considered as an extension of the *getStringAttrValue* one. Indeed, instead of returning an empty string if the XML!Attribute does not exist, it returns the "void" value.

The *getStringDataValue* helper returns the string value of the data contained in an XML!Element. This data may be sometimes stored into several XML!Text children of this XML!Element. This is the reason why the helper makes a loop on all these XML!Text elements and reconstructs the string data by using the "String concat(String) : String" method with the XML!Text values.

The *getSimpleStringDataValue* helper also returns the string value of the data contained in an XML!Element. But when we use this helper in the "to" part of an ATL rule, we are sure that the data we seek is only contained in one XML!Text; so the string data's value can be directly returned (without making any loop on XML!Text elements).

The *getNumberDataValue* helper returns the real value of the data contained in an XML!Element. It calls the *getSimpleStringDataValue* helper in order to recover the data's string value and converts this value into a real one thanks to the "String toReal() : Real" method. If the data does not exist (i.e. if the *getSimpleStringDataValue* helper returns an empty string), the returned value is "0.0".

The *getBooleanDataValue* helper is quite similar to the previously described *getNumberDataValue* one. The only difference is that, instead of returning a real value, it returns a boolean value. Note that the “false” value is returned by the helper when the *getSimpleStringDataValue* helper’s call returns an empty string.

The 6 first rules of the ATL file follow the principle exposed in this paragraph. For each XML!Element encountered, the corresponding SpreadsheetMLSimplified element is allocated. Each type of SpreadsheetMLSimplified element has its proper rule: for example the *Workbook* rule creates a SpreadsheetMLSimplified!Workbook element from an XML!Element whose name is “Workbook”… The attributes’ values of each SpreadsheetMLSimplified model’s element are initialized (or not if it is not necessary) thanks to the helpers previously described. The generated SpreadsheetMLSimplified elements are correctly linked the ones to the others, by using “resolveTemp(…)” method’s calls, in order to preserve the global structure of the Excel workbook and to ensure that the created model conforms to the SpreadsheetMLSimplified metamodel. Thus, in the generated model, the workbook can contain several worksheets and each of these can store a table…

The 5 *XXXData* rules are a little different. In all cases, a SpreadsheetMLSimplified!Data element is created (from an XML!Element whose name is “Data”) and linked to the corresponding SpreadsheetMLSimplified!Cell element by a call to the “resolveTemp(…)” method. But other elements also have to be allocated. The type of these SpreadsheetMLSimplified elements depends on the value of the “ss:Type” XML!Attribute of the XML!Element (in the rule’s “from” part): for example if the “ss:Type”-named XML!Attribute’s value is “String”, a SpreadsheetMLSimplified!StringValue element is created and directly linked to its parent’s SpreadsheetMLSimplified!Data element…

```
module XML2SpreadsheetMLSimplified; -- Module Template
create OUT : SpreadsheetMLSimplified from IN : XML;

-- This helper permits to recover the value of a string attribute thanks to its
name.
-- It returns an empty string if the attribute doesn't exist.
-- CONTEXT: XML!Element
-- RETURN: String
helper context XML!Element def: getStringAttrValue(attrName : String) : String =
   let eltC : Sequence(XML!Attribute) =
      self.children->select(a | a.oclIsTypeOf(XML!Attribute) and a.name = attrName)-
>asSequence()
   in
      if eltC->notEmpty()
      then
         eltC->first().value
      else
         ''
      endif;


-- This helper permits to recover the element's set of children thanks to their
name.
-- CONTEXT: XML!Element
-- RETURN: Set(XML!Element)
helper context XML!Element def: getChildrenByName(name : String) : Set(XML!Element)
=
   self.children->select(e | e.oclIsTypeOf(XML!Element) and e.name = name);


-- This helper permits to recover the value of an optional integer attribute
-- if it exists
-- CONTEXT: XML!Element
-- RETURN: Integer
helper context XML!Element def: getOptIntAttrValue(attrName : String) : Integer =
   let val : String = self.getStringAttrValue(attrName)
   in
```

```
     if val <> ''
     then
        val.toInteger()
     else
        Integer
     endif;


-- This helper permits to recover the value of an optional boolean attribute
-- if it exists
-- CONTEXT: XML!Element
-- RETURN: Boolean
helper context XML!Element def: getOptBoolAttrValue(attrName : String) : Boolean =
   let val : String = self.getStringAttrValue(attrName)
   in
     if val <> ''
     then
        if val = '0'
        then
             false
        else
             true
        endif
     else
        Boolean
     endif;


-- This helper permits to recover the value of an optional real attribute
-- if it exists
-- CONTEXT: XML!Element
-- RETURN: Real
helper context XML!Element def: getOptRealAttrValue(attrName : String) : Real =
   let val : String = self.getStringAttrValue(attrName)
   in
     if val <> ''
     then
        val.toReal()
     else
        Real
     endif;


-- This helper permits to recover the value of an optional string attribute
-- if it exists
-- CONTEXT: XML!Element
-- RETURN: String
helper context XML!Element def: getOptStringAttrValue(attrName : String) : String =
   let val : String = self.getStringAttrValue(attrName)
   in
     if val <> ''
     then
        val
     else
        String
     endif;


-- This helper permits to recover the value of a string data.
-- The string have to be sometimes reconstructed.
-- It returns an empty string if the value doesn't exist.
-- CONTEXT: XML!Element
-- RETURN: String
helper context XML!Element def: getStringDataValue() : String =
   let eltC : Sequence(XML!Text) =
      self.children->select(d | d.oclIsTypeOf(XML!Text))->asSequence()
   in
      if eltC->notEmpty()
      then
           eltC->iterate(txt; res : String = '' |
              res.concat(txt.value)
           )
      else
         ''
      endif;


-- This helper permits to recover the value of a simple string data.
-- It returns an empty string if the value doesn't exist.
-- CONTEXT: XML!Element
-- RETURN: String
helper context XML!Element def: getSimpleStringDataValue() : String =
   let eltC : Sequence(XML!Text) =
      self.children->select(d | d.oclIsTypeOf(XML!Text))->asSequence()
   in
      if eltC->notEmpty()
      then
         eltC->first().value
      else
         ''
      endif;


-- This helper permits to recover the value of a number data.
-- It returns 0.0 if the value doesn't exist.
-- CONTEXT: XML!Element
-- RETURN: Real
helper context XML!Element def: getNumberDataValue() : Real =
   let val : String = self.getSimpleStringDataValue()
   in
      if val <> ''
      then
         val.toReal()
      else
         0.0
      endif;


-- This helper permits to recover the value of a boolean data.
-- It returns false if the value doesn't exist.
-- CONTEXT: XML!Element
-- RETURN: Boolean
helper context XML!Element def: getBooleanDataValue() : Boolean =
   let val : String = self.getSimpleStringDataValue()
   in
      if val <> ''
      then
         if val = '0'
         then
            false
         else
            true
         endif
      else
```

```
         false
      endif;



-- Rule 'Workbook'
-- This rule generates the workbook which is the
-- root container of a SpreadsheetML document
rule Workbook {
   from
      rw : XML!Root (
         rw.name = 'Workbook'
      )

   to
      wb : SpreadsheetMLSimplified!Workbook (
         wb_worksheets <- Sequence{rw.getChildrenByName('Worksheet')->collect(e |
thisModule.resolveTemp(e, 'ws'))}
      )
}


-- Rule 'Worksheet'
-- This rule generates the worksheets that are contained
-- in a workbook.
rule Worksheet {
   from
      ew : XML!Element (
         ew.name = 'Worksheet'
      )

   to
      ws : SpreadsheetMLSimplified!Worksheet (
         name <- ew.getStringAttrValue('ss:Name'),
         ws_table <- Sequence{ew.getChildrenByName('Table')->first()}->collect(e |
thisModule.resolveTemp(e, 'tab'))->first()
      )
}


-- Rule 'Table'
-- This rule generates the table for a worksheet.
-- It's the table which contains the columns and rows.
rule Table {
   from
      et : XML!Element (
         et.name = 'Table'
      )

   to
      tab : SpreadsheetMLSimplified!Table (
         t_cols <- Sequence{et.getChildrenByName('Column')->collect(e |
thisModule.resolveTemp(e, 'col'))},
         t_rows <- Sequence{et.getChildrenByName('Row')->collect(e |
thisModule.resolveTemp(e, 'row'))}
      )
}


-- Rule 'Column'
-- This rule generates the columns contained in a table.
-- They don't store the data but they give some specific information about columns
format.
rule Column {
   from
      ec : XML!Element (
         ec.name = 'Column'
      )

   to
      col : SpreadsheetMLSimplified!Column (
         index <- ec.getOptIntAttrValue('ss:Index'),
         hidden <- ec.getOptBoolAttrValue('ss:Hidden'),
         span <- ec.getOptIntAttrValue('ss:Span'),
         autoFitWidth <- ec.getOptBoolAttrValue('ss:AutoFitWidth'),
         width <- ec.getOptRealAttrValue('ss:Width')
      )
}


-- Rule 'Row'
-- This rule generates the rows contained in a table.
-- They store the data (in the cells) and give some specific information about rows
format.
rule Row {
   from
      er : XML!Element (
         er.name = 'Row'
      )

   to
      row : SpreadsheetMLSimplified!Row (
         r_cells <- Sequence{er.getChildrenByName('Cell')->collect(e |
thisModule.resolveTemp(e, 'cell'))},
         index <- er.getOptIntAttrValue('ss:Index'),
         hidden <- er.getOptBoolAttrValue('ss:Hidden'),
         span <- er.getOptIntAttrValue('ss:Span'),
         autoFitHeight <- er.getOptBoolAttrValue('ss:AutoFitHeight'),
         height <- er.getOptRealAttrValue('ss:Height')
      )
}


-- Rule 'Cell'
-- This rule generates the cells of the table.
-- They are contained in the rows and they store the data.
rule Cell {
   from
      ece : XML!Element (
         ece.name = 'Cell'
      )

   to
      cell : SpreadsheetMLSimplified!Cell (
         index <- ece.getOptIntAttrValue('ss:Index'),
         arrayRange <- ece.getOptStringAttrValue('ss:ArrayRange'),
         formula <- ece.getOptStringAttrValue('ss:Formula'),
         hRef <- ece.getOptStringAttrValue('ss:Href'),
         mergeAcross <- ece.getOptRealAttrValue('ss:Href'),
         mergeDown <- ece.getOptRealAttrValue('ss:Href')
      )
}

```

```

-- Rule 'StringData'
-- This rule generates the string data of the table.
-- They are contained in the cells.
rule StringData {
   from
     esd : XML!Element (
        esd.name = 'Data' and esd.getStringAttrValue('ss:Type')='String'
     )

   to
     sdata : SpreadsheetMLSimplified!Data (
        d_cell <- Sequence{esd.parent}->collect(e | thisModule.resolveTemp(e,
'cell'))->first(),
        value <- sv
     ),
     sv : SpreadsheetMLSimplified!StringValue (
        value <- esd.getStringDataValue()
     )
}

-- Rule 'NumberData'
-- This rule generates the number data of the table.
-- They are contained in the cells.
rule NumberData {
   from
     end : XML!Element (
        end.name = 'Data' and end.getStringAttrValue('ss:Type')='Number'
     )

   to
     ndata : SpreadsheetMLSimplified!Data (
        d_cell <- Sequence{end.parent}->collect(e | thisModule.resolveTemp(e,
'cell'))->first(),
        value <- nv
     ),
     nv : SpreadsheetMLSimplified!NumberValue (
        value <- end.getNumberDataValue()
     )
}

-- Rule 'BooleanData'
-- This rule generates the boolean data of the table.
-- They are contained in the cells.
rule BooleanData {
   from
     ebd : XML!Element (
        ebd.name = 'Data' and ebd.getStringAttrValue('ss:Type')='Boolean'
     )

   to
     bdata : SpreadsheetMLSimplified!Data (
        d_cell <- Sequence{ebd.parent}->collect(e | thisModule.resolveTemp(e,
'cell'))->first(),
        value <- bv
     ),
     bv : SpreadsheetMLSimplified!BooleanValue (
        value <- ebd.getBooleanDataValue()
     )
}

-- Rule 'DateTimeData'
-- This rule generates the "DateTime" data of the table.
-- They are contained in the cells.
rule DateTimeData {
   from
      edtd : XML!Element (
         edtd.name = 'Data' and edtd.getStringAttrValue('ss:Type')='DateTime'
      )

   using {
      dateTimeString : String = edtd.getSimpleStringDataValue();
   }

   to
      dtdata : SpreadsheetMLSimplified!Data (
         d_cell <- Sequence{edtd.parent}->collect(e | thisModule.resolveTemp(e,
'cell'))->first(),
         value <- dttv
      ),
      dttv : SpreadsheetMLSimplified!DateTimeTypeValue (
         value <- dt
      ),
      -- The format for date/time fields in Excel is : yyyy-mm-ddThh:mm:ssZ
      dt : SpreadsheetMLSimplified!DateTimeType (
         year <- dateTimeString.substring(1,4).toInteger(),
         month <- dateTimeString.substring(6,7).toInteger(),
         day <- dateTimeString.substring(9,10).toInteger(),
         hour <- dateTimeString.substring(12,13).toInteger(),
         minute <- dateTimeString.substring(15,16).toInteger(),
         second <- dateTimeString.substring(18,19).toInteger()
      )
}

-- Rule 'ErrorData'
-- This rule generates the "error" data of the table.
-- They are contained in the cells.
rule ErrorData {
   from
      eed : XML!Element (
         eed.name = 'Data' and eed.getStringAttrValue('ss:Type')='Error'
      )

   to
      edata : SpreadsheetMLSimplified!Data (
         d_cell <- Sequence{eed.parent}->collect(e | thisModule.resolveTemp(e,
'cell'))->first(),
         value <- ev
      ),
      ev : SpreadsheetMLSimplified!ErrorValue ()
}
```

I. XML metamodel in KM3 format

```
-- @name   XML
-- @version  1.1
-- @domains  XML
-- @authors  Peter Rosenthal (peter.rosenthal@univ-nantes.fr)
-- @date   2005/06/13
-- @description This metamodel defines a subset of Extensible Markup Language (XML)
and particulary XML document. It describes an XML document composed of one root
node. Node is an abstract class having two direct children, namely ElementNode and
AttributeNode. ElementNode represents the tags, for example a tag named xml:
<xml></xml>. ElementNodes can be composed of many Nodes. AttributeNode represents
attributes, which can be found in a tag, for example the attr attribute: <xml
attr="value of attr"/>. ElementNode has two sub classes, namely RootNode and
TextNode. RootNode is the root element. The TextNode is a particular node, which
does not look like a tag; it is only a string of characters.

package XML {
   abstract class Node {
      attribute startLine[0-1] : Integer;
      attribute startColumn[0-1] : Integer;
      attribute endLine[0-1] : Integer;
      attribute endColumn[0-1] : Integer;
      attribute name : String;
      attribute value : String;
      reference parent[0-1] : Element oppositeOf children;
   }

   class Attribute extends Node {}

   class Text extends Node {}

   class Element extends Node {
      reference children[*] ordered container : Node oppositeOf parent;
   }

   class Root extends Element {}
}

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

II. SpreadsheetMLSimplified metamodel in KM3 format

```
-- @name   SpreadsheetMLSimplified
-- @version  1.2
-- @domains  Microsoft Office Excel, XML
-- @authors  Hugo Bruneliere (hugo.bruneliere@gmail.com)
-- @date   2005/07/01
-- @description This metamodel describes a simplified subset of SpreadsheetML, an
XML dialect developed by Microsoft to represent the information in an Excel
spreadsheet. The root element for an XML spreadsheet is the Workbook element. A
Workbook element can contain multiple Worksheet elements. A Worksheet element can
contain a Table element. It holds the row elements that define a spreadsheet. A row
holds the cell elements that make it up. A Cell element holds the data. In
addition, Column elements (children of the Table element) can be used to define the
attributes of columns in the spreadsheet.
-- @see    excelss.xsd; Microsoft Office 2003 XML Reference Schemas;
http://www.microsoft.com/downloads/details.aspx?familyid=FE118952-3547-420A-A412-
00A2662442D9&displaylang=en

package SpreadsheetMLSimplified {

-- @begin MS Office - Special Types definition

   -- @comment The format for date/time fields is yyyy-mm-ddThh:mm:ssZ. (This format
can be described as follows: a four-digit year, hyphen, two-digit month, hyphen,
two-digit day, uppercase letter T, two-digit hour, colon, two-digit minute value,
colon, two-digit seconds value, uppercase letter Z.).
   class DateTimeType {
      attribute year : Integer;
      attribute month : Integer;
      attribute day : Integer;
      attribute hour : Integer;
      attribute minute : Integer;
      attribute second : Integer;
   }

   -- @comment Office manages five types of value : String, Number, DateTime,
Boolean and Error.
   abstract class ValueType {
      reference vt_data : Data oppositeOf value;
   }

   class StringValue extends ValueType {
      attribute value : String;
   }

   class NumberValue extends ValueType {
      attribute value : Double;
   }

   class DateTimeTypeValue extends ValueType {
      reference value container : DateTimeType;
   }

   class BooleanValue extends ValueType {
      attribute value : Boolean;
   }

   class ErrorValue extends ValueType {}

-- @end MS Office - Special Types definition
-- @begin MS Office - Excel workbook basic definition

   -- @comment Defines a workbook that will contain one or more Worksheet elements.
   class Workbook {
      -- @comment At least one instance of the Worksheet element is required for a
valid spreadsheet but the XML schema permit having no instance.
      reference wb_worksheets[*] ordered container : Worksheet oppositeOf
ws_workbook;
   }

   -- @comment Defines a worksheet within the current workbook.
   class Worksheet {
      reference ws_workbook : Workbook oppositeOf wb_worksheets;

      -- @comment Only one instance of a Table element is valid for a single
worksheet.
      reference ws_table[0-1] container : Table oppositeOf t_worksheet;

      -- @comment Specifies the name of a worksheet. This value must be unique
within the list of worksheet names of a given workbook.
      attribute name : String;
   }

   -- @comment Defines the table to contain the cells that constitute a worksheet.
   class Table {
      reference t_worksheet :  Worksheet oppositeOf ws_table;

      -- @comment A table contains columns and rows.
      reference t_cols[*] ordered container : Column oppositeOf c_table;
      reference t_rows[*] ordered container : Row oppositeOf r_table;
   }

   -- @comment Defines a table element, that is to say a column, a row or a cell.
   abstract class TableElement {
      -- @comment Specifies the position of the element in the table. For a cell, it
specifies the column index.
      attribute index[0-1] : Integer;
   }

   -- @comment Defines a row or a column.
   abstract class ColOrRowElement extends TableElement {
      -- @comment Specifies whether a row or a column is hidden.
      attribute hidden[0-1] : Boolean;
      -- @comment Specifies the number of adjacent columns/rows with the same
formatting as the defined column/row. This integer mustn't be negative.
      attribute span[0-1] : Integer;
   }

   -- @comment Defines the formatting and properties for a column
   class Column extends ColOrRowElement {
      reference c_table : Table oppositeOf t_cols;

      -- @comment Specifies whether a column is automatically resized to fit numeric
and date values. Columns are not resized to fit text data.
      attribute autoFitWidth[0-1] : Boolean;
      -- @comment Specifies the width of a column in points. This value must be
greater than or equal to zero.
      attribute width[0-1] : Double;
   }

   -- @comment Defines the formatting and properties for a row
   class Row extends ColOrRowElement {
      reference r_table : Table oppositeOf t_rows;

      -- @comment A row contains zero or more cells.
      reference r_cells[*] ordered container : Cell oppositeOf c_row;

      -- @comment Specifies whether the height of a row is automatically resized to
fit the contents of cells.
      attribute autoFitHeight[0-1] : Boolean;
      -- @comment Specifies the height of a row in points. This value must be
greater than or equal to zero.
      attribute height[0-1] : Double;
   }

   -- @comment Defines the properties of a cell in a worksheet.
   class Cell extends TableElement {
      -- @comment A cell is contained in a row.
      reference c_row : Row oppositeOf r_cells;

      -- @comment Specifies the range of cells to which an array formula applies.
      attribute arrayRange[0-1] : String;
      -- @comment Specifies a formula for a cell.
      attribute formula[0-1] : String;
      -- @comment Specifies a URL to which to which a cell is linked.
      attribute hRef[0-1] : String;
      -- @comment Specifies the number of adjacent cells to merge with the current
cell. The cells to merge will be to the right of the current cell unless the
worksheet is set to display left-to-right.
      attribute mergeAcross[0-1] : Double;
      -- @comment Specifies the number of adjacent cells below the current cell that
are to be merged with the current cell.
      attribute mergeDown[0-1] : Double;
      -- @comment A cell can contain a data.
      reference c_data[0-1] container : Data oppositeOf d_cell;
   }

   -- @comment Specifies the value of a cell. The value should be specified in the
format and type appropriate for (String, Number, DateTime, Boolean or Error).
   class Data {
      reference d_cell : Cell oppositeOf c_data;

      -- @comment Defines the value of the cell in the correct type
      reference value container : ValueType oppositeOf vt_data;
   }

-- @end MS Office - Excel workbook basic definition

}


package PrimitiveTypes {

   datatype Integer;
   datatype String;
   datatype Boolean;
   datatype Double;

}
```

References

[1] Office 2003: XML Reference Schemas, http://www.microsoft.com/downloads/details.aspx?FamilyId=FE118952-3547-420A-A412-00A2662442D9&displaylang=en