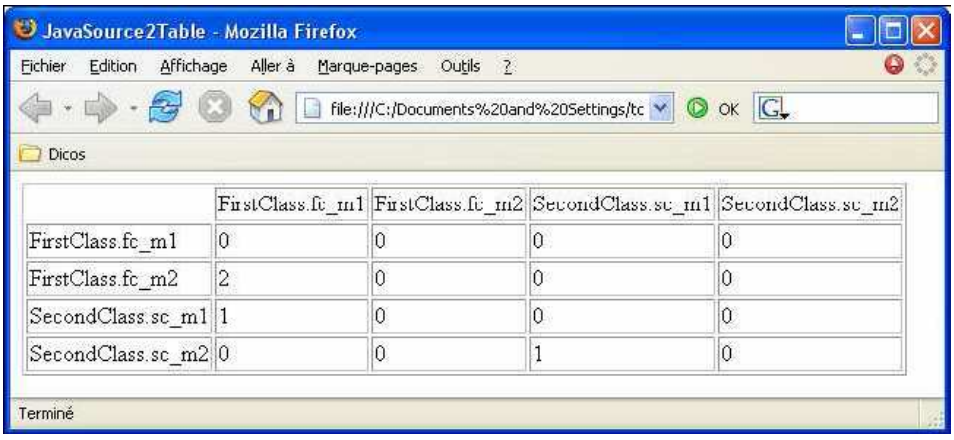
1. ATL Transformation Example

1.1. Example: Table → Microsoft Office Excel

The Table to Microsoft Office Excel example describes a transformation from a very basic table representation to an Excel workbook with one single worksheet containing this table. 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 XML file which can be directly open by Excel 2003. This file describes a workbook with the same content (i.e. the same table) that the one of the input Table model.

1.1.1. Transformation overview

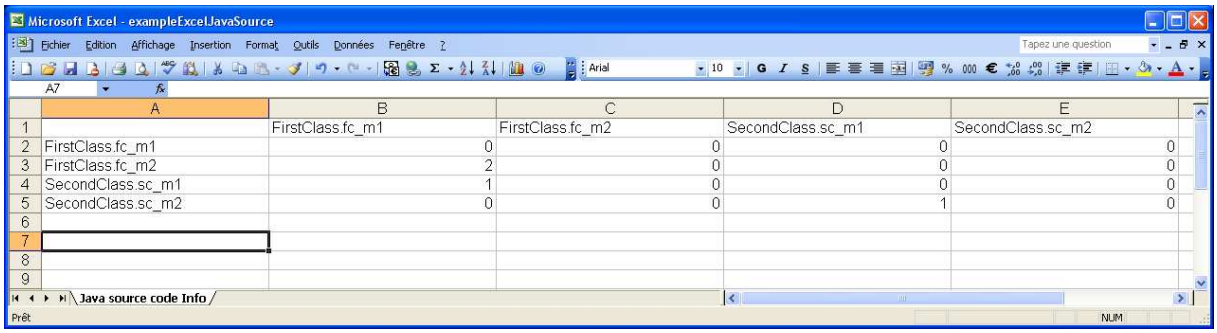
The aim of this transformation is to generate an Excel workbook from a simple table whose content is stored in a Table model. Figure 1 gives an example of a table which is a particular representation for Java source code. This table is the HTML representation of the content of the model (conforms to the "Table" metamodel) which has been generated by the JavaSource2Table transformation described in [1].



	FirstClass.fc_m1	FirstClass.fc_m2	SecondClass.sc_m1	SecondClass.sc_m2
FirstClass.fc_m1	0	0	0	0
FirstClass.fc_m2	2	0	0	0
SecondClass.sc_m1	1	0	0	0
SecondClass.sc_m2	0	0	1	0

Figure 1. A simple table example

As an example of the transformation, Figure 2 provides a screen capture of the Microsoft Office Excel workbook generated from the initial simple table graphically represented in Figure 1. As we already said in 1.1, this workbook is only composed of a single worksheet which contains the table raw data.



	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 2. The corresponding MS Office Excel workbook

To make the Table2MicrosoftOfficeExcel global transformation we proceed in three steps. Indeed, this transformation is in reality a composition of three transformations:

- from Table to SpreadsheetMLSimplified
- from SpreadsheetMLSimplified to XML
- from XML to Excel text (Excel XML file)

These steps are summarized in Figure 3.

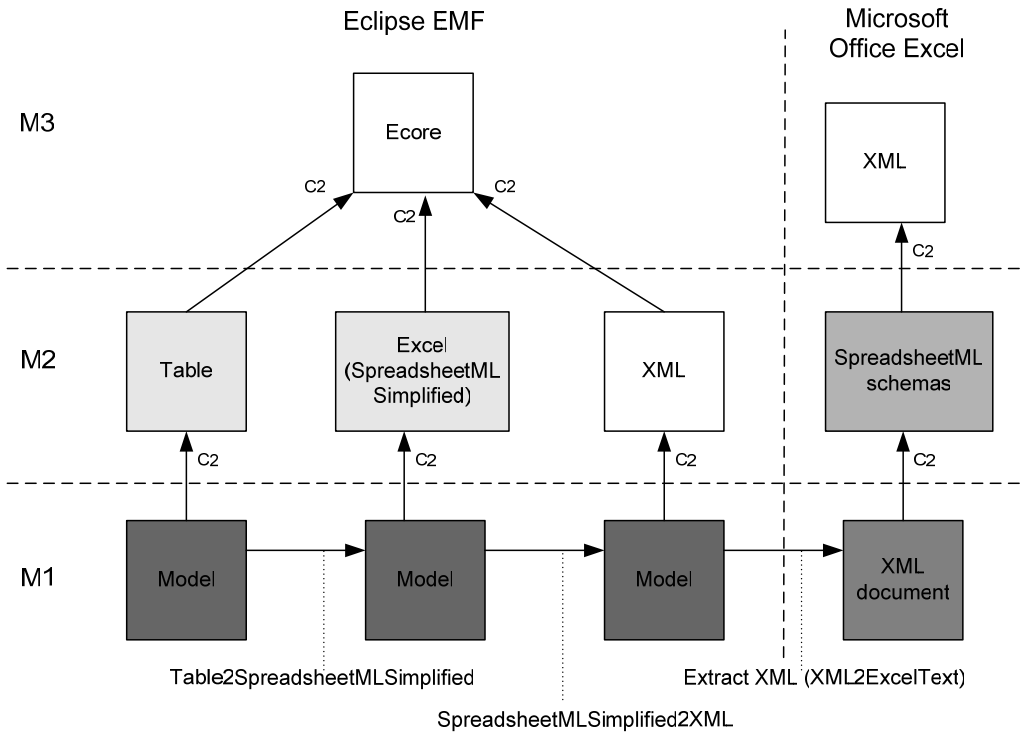


Figure 3. "Table to Microsoft Office Excel" transformation overview

1.2. Metamodels

The first step of the transformation takes in entry a model which conforms to the “Table” metamodel. This very basic abstract metamodel is presented in Figure 4 and provided in Appendix I in KM3 format.

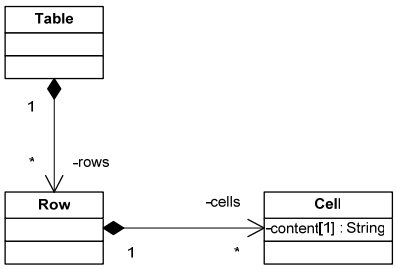


Figure 4. A Table metamodel

A table is represented by a Table element. A table can contain several rows. Each row can be composed of several cells. The raw data are contained in the “content” string attribute of the “Cell” class.

The transformation is also 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 [2]). The metamodel considered here is described in Figure 5 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).

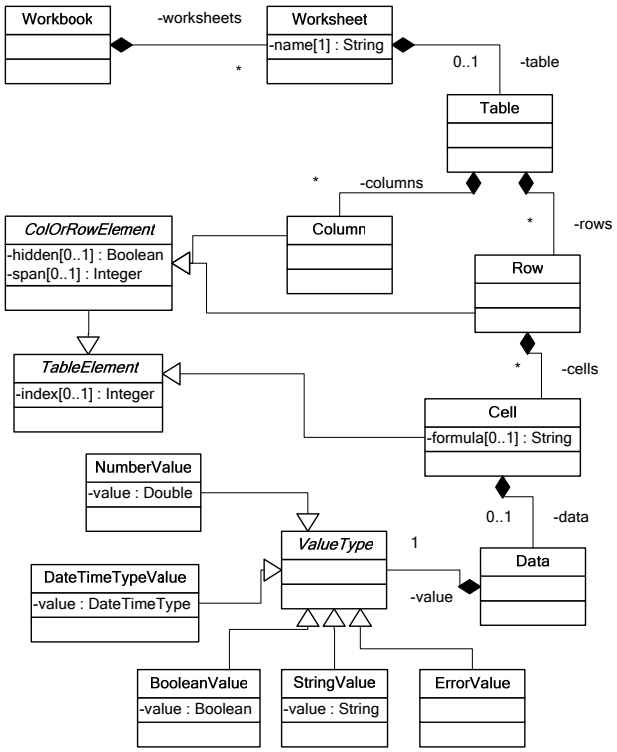


Figure 5. The SpreadsheetMLSimplified metamodel

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 data of a particular type which can be “Number”, “DateTime”, ”Boolean”, ”String” or “Error”.

The last metamodel used by this transformation is a simple XML metamodel which is necessary to export models into XML files. This metamodel is presented in Figure 6 and provided in Appendix III in KM3 format.

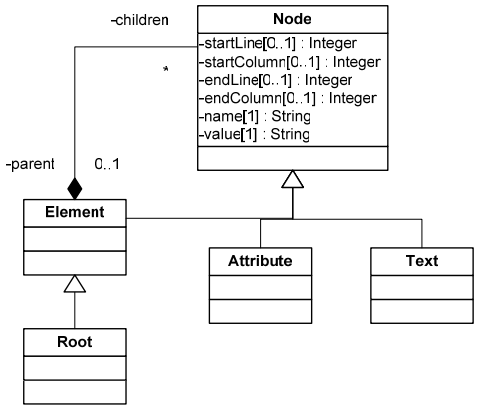


Figure 6. A simple XML metamodel

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 either “Elements”, “Attributes” or “Texts”. An “Element” is usually characterized by its name and has a number of children. An “Attribute” is characterized by its name and its value whereas a “Text” is only assimilated to a single value.

1.3. Rules Specification

There is a set of rules for each transformation of the global transformation sequence. The input model of each transformation is the output model generated by the preceding transformation. Note that the Table model, which is the input model of the global transformation, may have been previously generated by the JavaSource2Table simple transformation (see [1]).

1.3.1. Table to SpreadsheetMLSimplified

These are the rules to transform a Table model to a SpreadsheetMLSimplified model:

- For the root Table element, the following elements are created: a Workbook with one Worksheet that contains a Table which has as many columns as the number of cells in the first row of the input model's table.
- For each row, a Row element is associated to the Table previously created.
- For each cell, a Cell element is created with a Data element that contains a value in the correct data type. For instance, if the "content" attribute value contained in an input model's "Cell" class can be converted into a numeric value, a NumberValue element can be created instead of a StringValue element.

1.3.2. SpreadsheetMLSimplified to XML

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

- For the root Workbook element, the "workbook" Root element is created.
- For each Worksheet element, a "worksheet" Element is added as a child of the "workbook" root.
- For each Table element, a "table" Element is created and set as a child of the corresponding "workbook" Element.
- For each Column or Row element, a "column" or "row" Element is generated and positioned as a child of the corresponding "table" Element.
- For each Cell element, a "cell" Element is created and added as a child of the corresponding "row" element.
- For each Data element, a "data" Element is engendered and associated to the right "cell" element.
- For each NumberValue, StringValue..., an Attribute element is generated with the name "ss:Type" and the value corresponding to the input element's type ("Number","String",…); a Text element containing the data's value is engendered. These two Nodes are set as children of the corresponding "data" Element.

1.3.3. XML to Excel text (Extract XML)

There are no rules defined for this step but only an ATL query (and its associated ATL helpers) that allows generating an Excel valid and well-formed XML text file from an XML model. The aim of this query is to extract each of the elements that compose the input XML model into an output XML file. Look at the "ATL Code" following section to get more details about this ATL query.

1.4. ATL Code

There is one ATL file coding a transformation for each step previously detailed. In this part we will present and describe more precisely the ATL code associated to each implemented transformation.

1.4.1. Table2SpreadsheetMLSimplified

The ATL code for this transformation consists of 1 helper and 3 rules.

The “isNumber()” helper returns a boolean value indicating whether the content of a cell in a “Table” model (which is a string) represents in reality a numeric value or not. This helper takes in arguments the string to process and a boolean set as “true” if the helper is processing the first character of the string. It analyses the string character per character (recursively) to determine whether the string is only composed of alphanumerical characters or not.

The rule Table2ExcelTable allocates a Workbook, a Worksheet, a Table and the correct number of Column elements in order to create the global Excel workbook structure. The Table element is linked to all the Row elements that will be generated during the transformation (by the following rule).

The rule Row2ExcelRow allocates a Row element for each row of the “Table” input model. Each Row element is linked to all the corresponding Cell elements that will be created during the transformation (by the following rule).

The rule Cell2ExcelCell allocates a Cell element, a Data element and another element corresponding to the correct value type for each cell of the “Table” input model. The type of the element to be generated is determined thanks to the “isNumber()” helper. This element (a NumberValue or StringValue element) is directly linked to the Data element.

```
1   module Table2SpreadsheetMLSimplified; -- Module Template
2   create OUT : SpreadsheetMLSimplified from IN : Table;
3
4
5   -- This helper permits to determine whether a string contains a number value or not
6   -- The method used in this helper is not exactly correct because it considers as a
7   number a string that can be composed of several '.' characters. It should be
8   improved in order to solve this problem. However, the helper returns the right
9   value in most cases.
10  -- CONTEXT: n/a
11  -- RETURN: Boolean
12  helper context Table!Cell def: isNumber(value : String, itIsFirstChar : Boolean) :
13  Boolean =
14     if value <> ''
15     then
16        let char : String =  value.substring(1,1)
17        in
18           if( char = '.' or char = '0' or char = '1' or char = '2' or char = '3' or
19  char = '4'
20              or char = '5' or char = '6' or char = '7' or char = '8' or char = '9')
21           then
22              self.isNumber(value.substring(2,value.size()),false)
23           else
24              false
25           endif
26     else
27        if itIsFirstChar
28        then
```

```
29            false
30         else
31            true
32         endif
33      endif;
34
35
36
37   -- Rule 'Table2ExcelTable'
38   -- This rule generates the global structure of an Excel document
39   -- and creates the Excel table
40   rule Table2ExcelTable {
41      from
42         t : Table!Table
43
44      using {
45         tableRow : Sequence(Table!Cell) = t.rows->first().cells;
46      }
47
48      to
49         wb : SpreadsheetMLSimplified!Workbook (
50            wb_worksheets <- Sequence{ws}
51         ),
52         ws : SpreadsheetMLSimplified!Worksheet (
53            name <- 'Java source code Info',
54            ws_table <- et
55         ),
56         et : SpreadsheetMLSimplified!Table (
57            t_rows <- Sequence{t.rows->collect(e | thisModule.resolveTemp(e, 'erow'))},
58            t_cols <- Sequence{col}
59         ),
60         col : distinct SpreadsheetMLSimplified!Column foreach(cell in tableRow)(
61            width <- 150.0
62         )
63   }
64
65
66   -- Rule 'Row2ExcelRow'
67   -- This rule generates the rows that will contain the cells
68   rule Row2ExcelRow {
69      from
70         row : Table!Row
71
72      to
73         erow : SpreadsheetMLSimplified!Row (
74            r_cells <- Sequence{ row.cells->collect(e | thisModule.resolveTemp(e,
75   'ecell'))}
76         )
77   }
78
79
80   -- Rule 'Cell2ExcelCell'
81   -- This rule generates the cells that will contain the data
82   rule Cell2ExcelCell {
83      from
84         cell : Table!Cell
85
86      using {
87         stringTypeOrNot : Sequence(String) =
88            let ct : String = cell.content
89            in
90               if cell.isNumber(ct,true)
```

```
91              then
92                 Sequence{}
93              else
94                 Sequence{ct}
95              endif;
96           numberTypeOrNot : Sequence(Real) =
97              let ct : String = cell.content
98              in
99                 if cell.isNumber(ct,true)
100                then
101                   Sequence{ct.toReal()}
102                else
103                   Sequence{}
104                endif;
105     }
106
107     to
108        ecell : SpreadsheetMLSimplified!Cell (
109           c_data <- edata
110        ),
111        edata : SpreadsheetMLSimplified!Data (),
112        stringVal : distinct SpreadsheetMLSimplified!StringValue foreach(stringTypeVal
113  in stringTypeOrNot) (
114           vt_data <- edata,
115           value <- stringTypeVal
116        ),
117        numberVal : distinct SpreadsheetMLSimplified!NumberValue foreach(numberTypeVal
118  in numberTypeOrNot) (
119           vt_data <- edata,
120           value <- numberTypeVal
121        )
122  }
```

1.4.2. SpreadsheetMLSimplified2XML

The ATL code for this transformation consists of 1 helper and 12 rules.

The “getDateTimeStringValue()” helper returns the string value corresponding to the SpreadsheetMLSimplified!DateTimeType argument. The format of this date/time string for the SpreadsheetML XML dialect is “yyyy-mm-ddThh:mm:ss.000”.

Each implemented rule follows the same principle: an XML “Element” element (with sometimes some associated “Attribute” elements) is allocated for each element of the SpreadsheetMLSimplified model. These generated XML elements are correctly linked to each others in order to preserve the global structure of the Excel workbook and to construct an XML model whose content conforms to the SpreadsheetML XML schemas [2].

As an example, the WorksheetTable rule allocates an XML Element and an XML Attribute (which is a child of the Element) for each Table element of the input SpreadsheetMLSimplified model. This Element is linked to the other Elements that will be created, by other rules, to represent table’s rows and columns during the transformation. It is also linked to another Element created by the Worksheets rule in order to represent the worksheet that contains the table in the input model.

The only specificity of this transformation concerns the rule CellData and DataXXXValue ones. Indeed, the link between an XML Element representing a Data and another one representing an XXXValue is made by the corresponding DataXXXValue rule and not by the CellData rule. Because the type of data contained in a cell is still not known when the Data element is parsed (by the CellData rule), it is the XXXValue element that determines its parent (in the DataXXXValue rule).

```
1   module SpreadsheetMLSimplified2XML; -- Module Template
2   create  OUT : XML from  IN : SpreadsheetMLSimplified;
3
4
5
6   -- This helper permits to obtain the string associated
7   -- to a DateTimeType value.
8   -- CONTEXT: n/a
9   -- RETURN: String
10  helper def: getDateTimeStringValue(dtv : SpreadsheetMLSimplified!DateTimeType) :
11  String =
12     dtv.year.toString() + '-' + dtv.month.toString() + '-' + dtv.day.toString() + 'T'
13     + dtv.hour.toString() + ':' + dtv.minute.toString() + ':' + dtv.second.toString()
14  + '.000';
15
16
17
18  -- Rule 'DocumentRoot'.
19  -- This rule generates the root element of an Excel xml file
20  -- which is the "Workbook" element
21  rule DocumentRoot {
22      from
23        wb : SpreadsheetMLSimplified!Workbook
24      to
25        r : XML!Root(
26           name<-'Workbook',
27           value <- '',
28           children <- Sequence{att1,att2,
29                               wb.wb_worksheets->collect(e | thisModule.resolveTemp(e,
30  'wsElt')) }
31        ),
```

```
32          att1 : XML!Attribute (
33             name <- 'xmlns',
34             value <- 'urn:schemas-microsoft-com:office:spreadsheet'
35          ),
36          att2 : XML!Attribute (
37             name <- 'xmlns:ss',
38             value <-'urn:schemas-microsoft-com:office:spreadsheet'
39          )
40    }
41
42
43    -- Rule 'Worksheets'.
44    -- This rule generates the different "Worksheet" elements
45    -- contained in a "Workbook" element
46    rule Worksheets {
47       from
48          ws : SpreadsheetMLSimplified!Worksheet
49
50       to
51          wsElt : XML!Element (
52             name <- 'Worksheet',
53             children <- Sequence{nameAtt,Sequence{ws.ws_table}->collect(e |
54    thisModule.resolveTemp(e, 'tElt'))->first()}
55          ),
56          nameAtt : XML!Attribute (
57             name <- 'ss:Name',
58             value <- ws.name,
59             parent <- wsElt
60          )
61    }
62
63
64    -- Rule 'WorksheetTable'.
65    -- This rule generates the "Table" element
66    -- contained in a "Worksheet" element
67    rule WorksheetTable {
68       from
69          t : SpreadsheetMLSimplified!Table
70
71       to
72          tElt : XML!Element (
73             name <- 'Table',
74             children <- Sequence{
75                            t.t_cols->collect(e | thisModule.resolveTemp(e, 'colElt')),
76                            t.t_rows->collect(e | thisModule.resolveTemp(e, 'rowElt'))
77                         }
78          )
79    }
80
81
82    -- Rule 'TableColumn'.
83    -- This rule generates the "Column" elements
84    -- contained in a "Table" element
85    rule TableColumn {
86       from
87          col : SpreadsheetMLSimplified!Column
88
89       using {
90          widthOrNot : Sequence(String) =
91             let wdh : Real = col.width
92             in
93                if wdh.oclIsUndefined()
```

```
94              then
95                 Sequence{}
96              else
97                 Sequence{wdh.toString()}
98              endif;
99       }
100
101      to
102         colElt : XML!Element (
103            name <- 'Column',
104            children <- Sequence{colWidth}
105         ),
106         colWidth : distinct XML!Attribute foreach(widthValue in widthOrNot)(
107            name <- 'ss:Width',
108            value <- widthValue
109         )
110   }
111
112
113   -- Rule 'TableRow'.
114   -- This rule generates the "Row" elements
115   -- contained in a "Table" element
116   rule TableRow {
117      from
118         row : SpreadsheetMLSimplified!Row
119
120      to
121         rowElt : XML!Element (
122            name <- 'Row',
123            children <- Sequence{row.r_cells->collect(e | thisModule.resolveTemp(e,
124   'cellElt'))}
125         )
126   }
127
128
129   -- Rule 'RowCell'.
130   -- This rule generates the "Cell" elements
131   -- contained in a "Row" element
132   rule RowCell {
133      from
134         cell : SpreadsheetMLSimplified!Cell
135
136      to
137         cellElt : XML!Element (
138            name <- 'Cell',
139            children <- Sequence{
140                        Sequence{cell.c_data}->collect(e | thisModule.resolveTemp(e,
141   'dataElt'))->first()
142                     }
143         )
144   }
145
146
147   -- Rule 'CellData'.
148   -- This rule generates the "Data" element
149   -- contained in a "Cell" element
150   rule CellData {
151      from
152         data : SpreadsheetMLSimplified!Data
153
154      to
155         dataElt : XML!Element (
```

```
156            name <- 'Data'
157         )
158   }
159
160
161   -- Rule 'DataStringValue'.
162   -- This rule generates the string value
163   -- associated to a "Data" element
164   rule DataStringValue {
165      from
166         strVal: SpreadsheetMLSimplified!StringValue
167
168      to
169         strValAtt : XML!Attribute (
170            parent <- Sequence{strVal.vt_data}->collect(e | thisModule.resolveTemp(e,
171   'dataElt'))->first(),
172            name <- 'ss:Type',
173            value <- 'String'
174         ),
175         strValTxt : XML!Text (
176            parent <- Sequence{strVal.vt_data}->collect(e | thisModule.resolveTemp(e,
177   'dataElt'))->first(),
178            value <- strVal.value
179         )
180   }
181
182
183   -- Rule 'DataNumberValue'.
184   -- This rule generates the number value
185   -- associated to a "Data" element
186   rule DataNumberValue {
187      from
188         numVal: SpreadsheetMLSimplified!NumberValue
189
190      to
191         numValAtt : XML!Attribute (
192            parent <- Sequence{numVal.vt_data}->collect(e | thisModule.resolveTemp(e,
193   'dataElt'))->first(),
194            name <- 'ss:Type',
195            value <- 'Number'
196         ),
197         numValTxt : XML!Text (
198            parent <- Sequence{numVal.vt_data}->collect(e | thisModule.resolveTemp(e,
199   'dataElt'))->first(),
200            value <- numVal.value.toString()
201         )
202   }
203
204
205   -- Rule 'DataBooleanValue'.
206   -- This rule generates the boolean value
207   -- associated to a "Data" element
208   rule DataBooleanValue {
209      from
210         boolVal: SpreadsheetMLSimplified!BooleanValue
211
212      to
213         boolValAtt : XML!Attribute (
214            parent <- Sequence{boolVal.vt_data}->collect(e | thisModule.resolveTemp(e,
215   'dataElt'))->first(),
216            name <- 'ss:Type',
217            value <- 'Boolean'
```

```
218         ),
219         boolValTxt : XML!Text (
220            parent <- Sequence{boolVal.vt_data}->collect(e | thisModule.resolveTemp(e,
221   'dataElt'))->first(),
222            value <- boolVal.value.toString()
223         )
224   }
225
226
227   -- Rule 'DataErrorValue'.
228   -- This rule generates the error value
229   -- associated to a "Data" element
230   rule DataErrorValue {
231      from
232         errVal: SpreadsheetMLSimplified!ErrorValue
233
234      to
235         errValAtt : XML!Attribute (
236            parent <- Sequence{errVal.vt_data}->collect(e | thisModule.resolveTemp(e,
237   'dataElt'))->first(),
238            name <- 'ss:Type',
239            value <- 'Error'
240         )
241   }
242
243
244   -- Rule 'DataDateTimeValue'.
245   -- This rule generates the date/time value
246   -- associated to a "Data" element
247   rule DataDateTimeValue {
248      from
249         dtVal: SpreadsheetMLSimplified!DateTimeTypeValue
250
251      to
252         dtValAtt : XML!Attribute (
253            parent <- Sequence{dtVal.vt_data}->collect(e | thisModule.resolveTemp(e,
254   'dataElt'))->first(),
255            name <- 'ss:Type',
256            value <- 'DateTime'
257         ),
258         dtValTxt : XML!Text (
259            parent <- Sequence{dtVal.vt_data}->collect(e | thisModule.resolveTemp(e,
260   'dataElt'))->first(),
261            value <- thisModule.getDateTimeStringValue(dtVal.value)
262         )
263   }
```

1.4.3. XML2ExcelText

The ATL code for this transformation consists in 4 helpers and 1 query.

Contrary to rules that are implemented to generate a model from another model, a query allows calculating output text files from an input model (see [3]). This is the reason why we need to use queries for this type of transformation: generating an XML file from an XML model. The implemented query get the Root element of the XML model and call the “ExcelFile()” helper on it. It recovers the string value returned by this helper (the generated XML text) and writes it into an XML file located in the path passed as a parameter. The parsing of all input model’s elements is recursively made from the “ExcelFile()” helper.

The “ExcelFile()” helper returns a string which is composed of the specific Excel XML file’s header and of the Excel XML file’s content. This content is generated by the “toString2()” helper called on the Root element of the XML model.

There are three “toString2()” helpers with different contexts. The XML!Attribute one simply returns the name and the value of an attribute in the correct string format. The XML!Text one only returns the string value contained in a text node. The XML!Element one returns the valid and well-formed content of the output XML file by parsing recursively all the element of the input XML model. Note that it sometimes calls the XML!Attribute and XML!Text “toString2()” helpers.

```
1   query XML2Text = XML!Root.allInstances()
2        ->asSequence()
3        ->first().ExcelFile().writeTo('C:\\ … path to be completed before using the
4   transformation … \\exampleExcelJavaSource.xml');
5
6
7
8   helper context XML!Root def: ExcelFile() : String =
9      '<?xml version="1.0"?>'+'\n'+'<?mso-application progid="Excel.Sheet"?>'+'\n'
10     + self.toString2('');
11
12
13  helper context XML!Element def: toString2(indent : String) : String =
14     let na : Sequence(XML!Node) =
15        self.children->select(e | not e.oclIsKindOf(XML!Attribute)) in
16     let a : Sequence(XML!Node) =
17        self.children->select(e | e.oclIsKindOf(XML!Attribute)) in
18     indent + '<' + self.name +
19     a->iterate(e; acc : String = '' |
20        acc + ' ' + e.toString2()
21     ) +
22     if na->size() > 0 then
23        '>'
24        + na->iterate(e; acc : String = '' |
25           acc +
26           if e.oclIsKindOf(XML!Text) then
27              ''
28           else
29              '\r\n'
30           endif
31           + e.toString2(indent + '  ')
32        ) +
33        if na->first().oclIsKindOf(XML!Text) then
34           '</' + self.name + '>'
35           else
```

```
36              '\r\n' + indent + '</' + self.name + '>'
37          endif
38       else
39          '/>'
40       endif;
41
42
43    helper context XML!Attribute def: toString2() : String =
44       self.name + '=\"' + self.value + '\"';
45
46
47    helper context XML!Text def: toString2() : String =
48       self.value;
```

I. 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;
}
```

II. SpreadsheetMLSimplified metamodel in km3 format

```
-- @name   SpreadsheetMLSimplified
-- @version  1.2
-- @domains  Microsoft Office Excel, XML
-- @authors  Hugo Bruneliere (hbruneliere@free.fr)
-- @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;

}
```

III. 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;
}
```

References

[1] ExampleJavaSource2Table[v00.01].pdf, http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/gmt-home/subprojects/ATL/ATL_examples/Java2Table/ExampleJavaSource2Table%5Bv00.01%5D.pdf

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

[3] ATL User manual, “4.1 Queries and the Generation of Text” subsection, http://www.eclipse.org/gmt/, ATL subproject, ATL Documentation Section