INRIA	ATL TRANSFORMATION EXAMPLE	Hugo Brunelière hugo.bruneliere@gmail.com
	Microsoft Office Excel to Software Quality Control	Date 01/08/2005

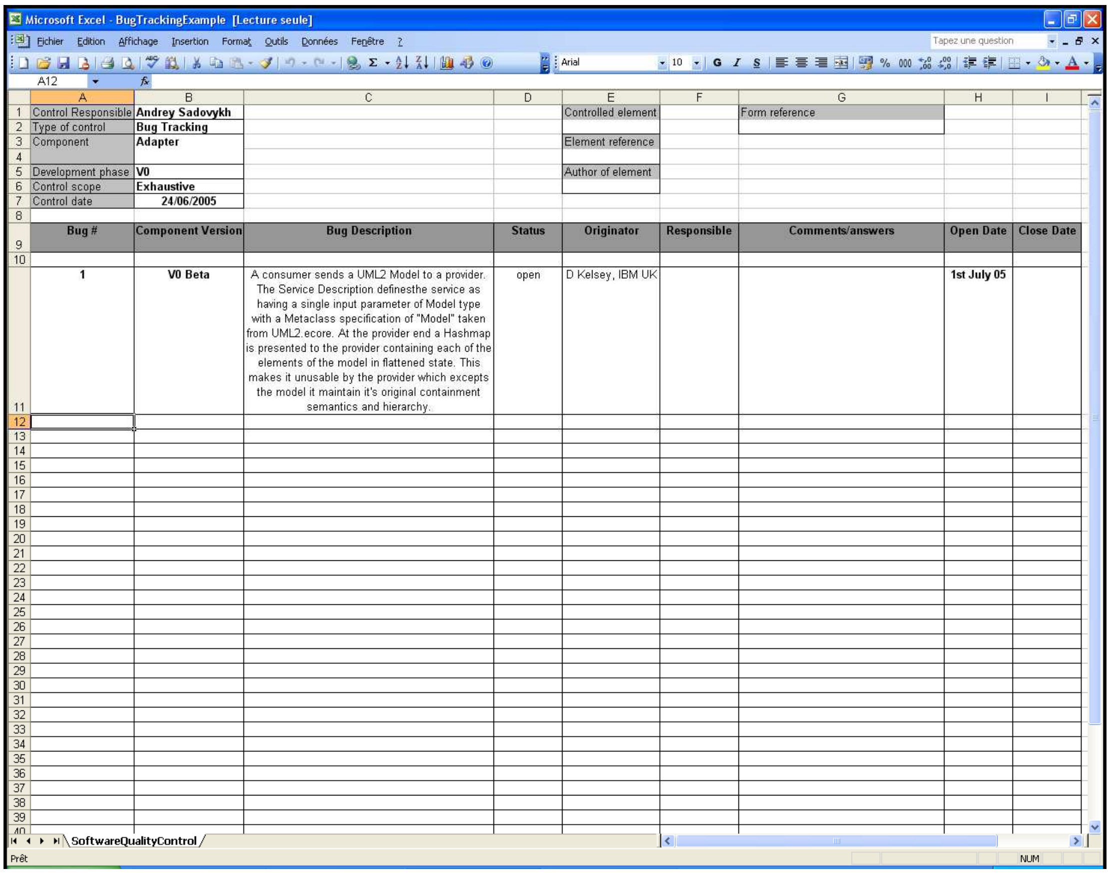
1. ATL Transformation Example

1.1. Example: Microsoft Office Excel → Software Quality Control

The "Microsoft Office Excel to Software Quality Control" example describes a transformation from an Excel workbook containing information about quality controls of a software product to a model storing the same data in a more general and abstract structure. 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. It is also based on a Software Quality Control metamodel which describes a simple structure to manage software quality controls (for example bug-tracking). The input of the transformation is an Excel XML file, containing the description of the Excel workbook, which can be directly opened by Excel 2003. The output is a model which conforms to the SoftwareQualityControl metamodel.

1.1.1. Transformation overview

The aim of this transformation is to generate a SoftwareQualityControl model from an Excel workbook whose content is a specific table representation for quality controls of software. Figure 1 gives an example of a simple workbook whose content is a particular representation for "bug-tracing" or "bug-tracking" (which is a type of software quality control). The bug's information contained in the single worksheet of this workbook has to be injected into a SoftwareQualityControl model.


Microsoft Excel - BugTrackingExample [Lecture seule]
Fichier Edition Affichage Insertion Format Outils Données Fenêtre ?
Tapez une question
Arial 10
A12
SoftwareQualityControl
Prêt
NUM

	A	B	C	D	E	F	G	H	I
1	Control Responsible	Andrey Sadovykh			Controlled element		Form reference		
2	Type of control	Bug Tracking							
3	Component	Adapter			Element reference				
4									
5	Development phase	V0			Author of element				
6	Control scope	Exhaustive							
7	Control date	24/06/2005							
8									
9	Bug #	Component Version	Bug Description	Status	Originator	Responsible	Comments/answers	Open Date	Close Date
10									
11	1	V0 Beta	A consumer sends a UML2 Model to a provider. The Service Description definesthe service as having a single input parameter of Model type with a Metaclass specification of "Model" taken from UML2.ecore. At the provider end a Hashmap is presented to the provider containing each of the elements of the model in flattened state. This makes it unusable by the provider which excepts the model it maintain it's original containment semantics and hierarchy.	open	D Kelsey, IBM UK			1st July 05	

Figure 1. An example of an Excel representation for a "bug-tracking" control.

To make the MicrosoftOfficeExcel2SoftwareQualityControl global transformation we proceed in two steps. Indeed, this transformation is in reality a composition of two transformations:

- from Excel XML file to SpreadsheetMLSimplified (inject Excel)
- from SpreadsheetMLSimplified to SoftwareQualityControl

These two steps are summarized in Figure 2.

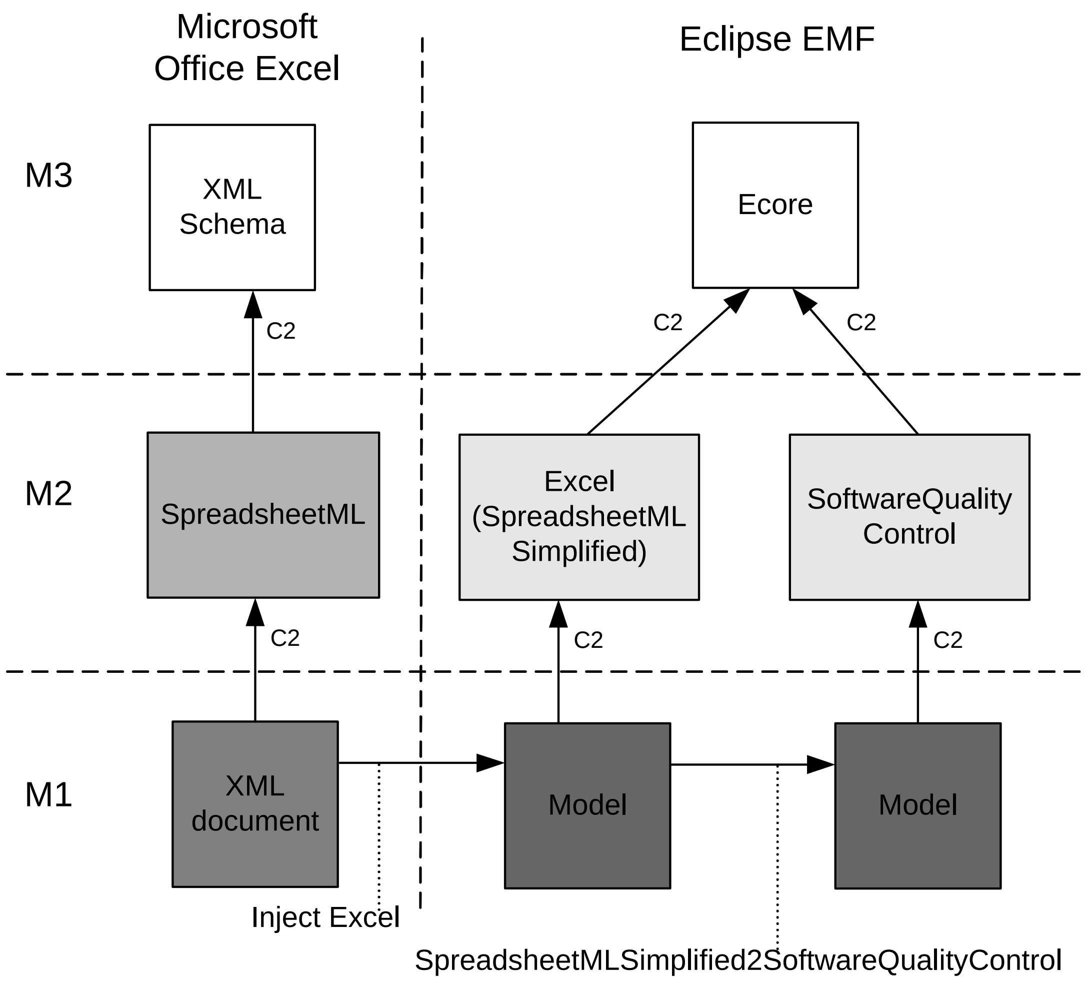


Figure 2. "Microsoft Office Excel to Software Quality Control" transformation's overview

1.2. Metamodels

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 3 and provided in Appendix I 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).

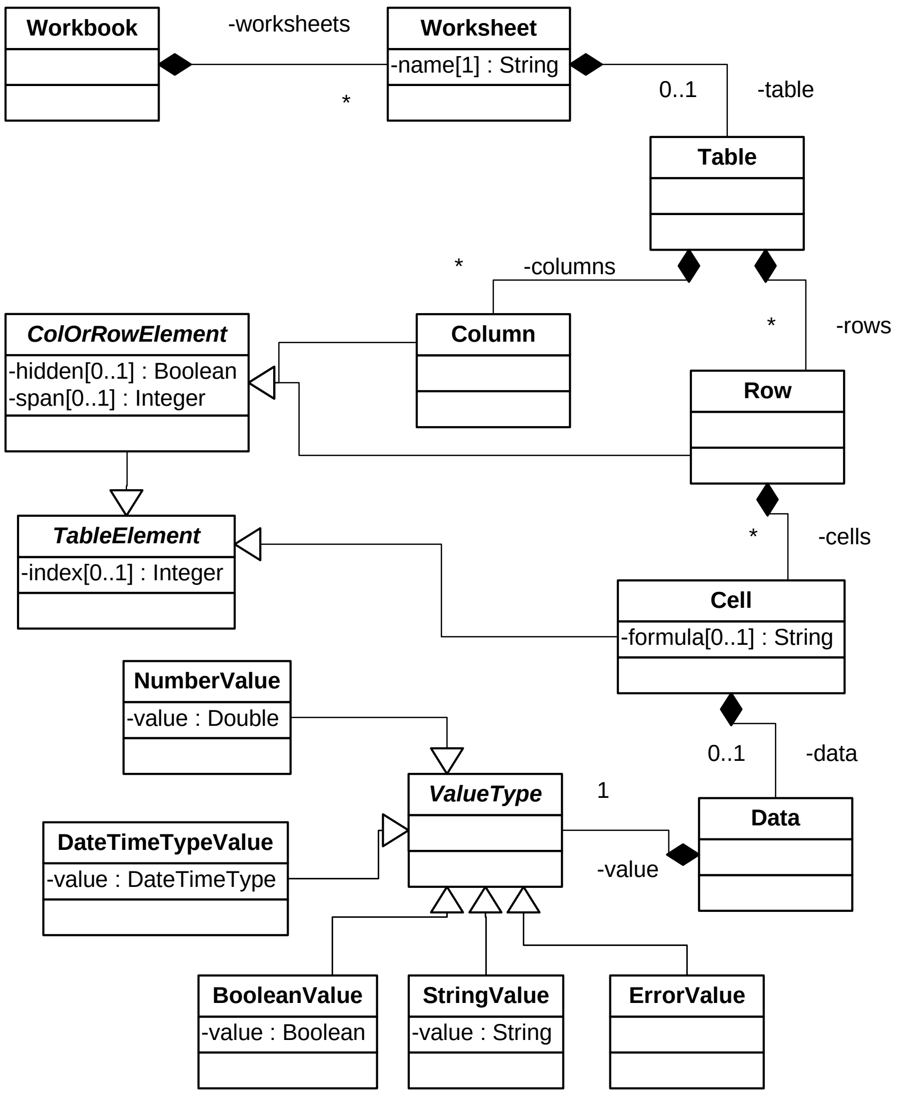


Figure 3. The SpreadsheetMLSimplified metamodel

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".

The transformation is also based on the "SoftwareQualityControl" metamodel which describes a simple structure to manage software quality controls and especially bug tracking. The metamodel considered here is described in Figure 4 and provided in Appendix II in km3 format. Note that we present in this documentation the current version of this metamodel that has been created for our particular example: it could be improved in order to allow handling several other types of quality control.

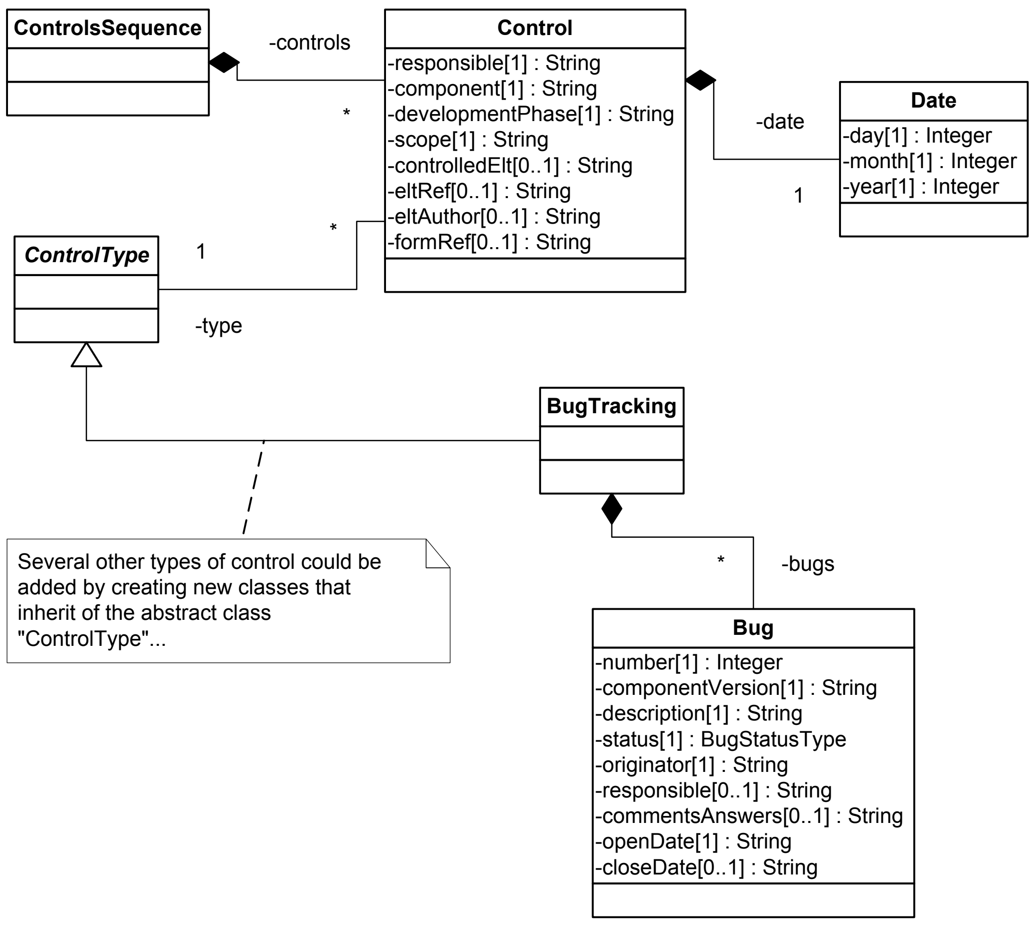

ControlsSequence
-controls
*
Control
-responsible[1] : String
-component[1] : String
-developmentPhase[1] : String
-scope[1] : String
-controlledElt[0..1] : String
-eltRef[0..1] : String
-eltAuthor[0..1] : String
-formRef[0..1] : String
-date
1
Date
-day[1] : Integer
-month[1] : Integer
-year[1] : Integer
ControlType
1
*
-type
BugTracking
Several other types of control could be added by creating new classes that inherit of the abstract class "ControlType"...
*
-bugs
Bug
-number[1] : Integer
-componentVersion[1] : String
-description[1] : String
-status[1] : BugStatusType
-originator[1] : String
-responsible[0..1] : String
-commentsAnswers[0..1] : String
-openDate[1] : String
-closeDate[0..1] : String

Figure 4. The SoftwareQualityControl metamodel

A "SoftwareQualityControl" model is composed of several *Control* elements. Each *Control* is defined by specific information about the component and the element which are concerned, about the person who is responsible for the control, the date, etc. The main information is the type of the control. It determines what kind of actions has been performed and consequently what kind of data has been saved. In the case of our example, we only create *BugTracking* type but it could have a lot of other control types. In this type, the control consists of a set of *Bug* elements in which each *Bug* is identified by a unique number. A *Bug* is characterized by several specific fields such as its description, its status…

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 a model which conforms to the SoftwareQualityControl metamodel (described in Figure 4). The input SpreadsheetMLSimplified model of the second transformation is the output SpreadsheetMLSimplified model generated by the first transformation.

1.3.1. Excel XML file to SpreadsheetMLSimplified (Inject Excel)

A first version of the Microsoft Office Excel injector has already been implemented. It offers the possibility to generate a SpreadsheetMLSimplified model from an Excel XML file that has been previously injected into an XML model. This version is perfectly sufficient for our example: it is described in details in [2] and available on the Eclipse/GMT web site.

1.3.2. SpreadsheetMLSimplified to SoftwareQualityControl

These are the rules to transform a SpreadsheetMLSimplified model into a SoftwareQualityControl model:

- For the root *Workbook* element, a *ControlsSequence* element is generated. It will be linked to the corresponding *Control* elements that will be created during the transformation by the following rule.
- For each *Worksheet* element, a *Control* element is created. Each *Control* element will be linked to an element of type *ControlType* (in our example it can only be a *BugTracking* element) that will be generated by other rules (only the following one in the case of our simple example).
- For each *Table* element in which the “Type of control” cell contains the value “Bug tracking”, a *BugTracking* element is engendered. This element is directly linked to the associated *Control* element that has been created by the preceding rule.
- For each *Row* element in which the first cell contains a number value (i.e. for each filled row of the table that contains the bug’s information), a Bug element is generated. Each *Bug* element is linked to the corresponding *BugTracking* element that has been created by the preceding rule during the transformation.

1.4. ATL Code

Since a version of the Microsoft Office Excel injector is already implemented (see [2]), there is only one new ATL file (coding the transformation) to detail. In this part we will present and describe more precisely the implementation of this part of the transformation.

The ATL code for the “SpreadsheetMLSimplified2SoftwareQualityControl” transformation consists of 4 helpers and 4 rules.

The *getStringValueByRowAndCell* helper returns the string value stored into a cell of the SpreadsheetMLSimplified!Worksheet context’s element thanks to the row and cell numbers passed in arguments. The model is traversed from the Worksheet (context of the helper) to the Data contained into the Cell indicated by the parameters.

The *getOptStringValueByRowAndCell* helper is quite similar to the previously described one. The only difference is that it returns “void” if the Cell element indicated by its two arguments does not contain any Data element.

The *getStringValueByColumn* helper returns the string value stored into a cell of the SpreadsheetMLSimplified!Row context’s element thanks to the column number (which is most of the time equivalent to the cell number in a row). Once again, the “void” value is returned if the indicated Cell element does not contain any Data element (i.e. if the cell is empty).

The *getBugStatus* helper returns the SoftwareQualityControl!BugStatusType value corresponding to the string value passed in argument. Note that if the string argument is not defined (i.e. “void”) or empty, the default returned value is “#bst_open”.

The rule *Workbook2ControlsSequence* allocates a ControlsSequence element for the Workbook root element of the SpreadsheetMLSimplified input model. This ControlsSequence element will be linked to the corresponding Control elements generated by the following rule (thanks to a “resolveTemp(…)” method’s call).

The rule *Worksheet2Control* allocates a Control element for each Worksheet element of the input model. The attributes (required or optional) of the created Control element are correctly initialized thanks to the previously defined helpers (respectively by the *getStringValueByRowAndCell* one and by the *getOptStringValueByRowAndCell* one).

The rule *BugTracking* allocates a BugTracking element for each Table element whose “Type of control” field (cell) is set with the “Bug Tracking” value. Each generated BugTracking element is immediately linked to the associated Control element (thanks to a “resolveTemp(…)” method’s call) that has been created during the transformation by the preceding rule.

The rule *Bug* allocates a Bug element for each Row element whose first Cell element contains a Data with a number value (i.e. for each row of the bug-tracking table that stores specific information about a bug). Each created Bug element is directly linked to the corresponding BugTracking element (thanks to a “resolveTemp(…)” method’s call) that has been generated by the preceding rule. The attributes of the created Bug element are initialized thanks to the previously defined *getStringValueByColumn* helper.

```
module SpreadsheetMLSimplified2SoftwareQualityControl; -- Module Template
create OUT : SoftwareQualityControl from IN : SpreadsheetMLSimplified;


-- This helper permits to recover a string value in the worksheet thanks to the row
and cell numbers.
```

```
-- CONTEXT: SpreadsheetMLSimplified!Worksheet
-- RETURN: String
helper context SpreadsheetMLSimplified!Worksheet def:
getStringValueByRowAndCell(rowNb : Integer, cellNb : Integer) : String =
   self.ws_table.t_rows->at(rowNb).r_cells->at(cellNb).c_data.value.value;


-- This helper permits to recover an optional string value in the worksheet thanks
to the row and cell numbers.
-- CONTEXT: SpreadsheetMLSimplified!Worksheet
-- RETURN: String
helper context SpreadsheetMLSimplified!Worksheet def:
getOptStringValueByRowAndCell(rowNb : Integer, cellNb : Integer) : String =
   let d : SpreadsheetMLSimplified!Data = self.ws_table.t_rows->at(rowNb).r_cells-
>at(cellNb).c_data
   in
      if d.oclIsUndefined()
      then
         String
      else
         d.value.value
      endif;


-- This helper permits to recover a string value in a row thanks to its column
number,
-- if it exists.
-- CONTEXT: SpreadsheetMLSimplified!Row
-- RETURN: String
helper context SpreadsheetMLSimplified!Row def: getStringValueByColumn(colNb :
Integer) : String =
   let d : SpreadsheetMLSimplified!Data = self.r_cells->at(colNb).c_data
   in
      if d.oclIsUndefined()
      then
         String
      else
         d.value.value
      endif;


-- This helper permits to recover the bug status value from a string
-- CONTEXT: n/a
-- RETURN: SoftwareQualityControl!BugStatusType
helper def: getBugStatus(st : String) : SoftwareQualityControl!BugStatusType =
   if ( not st.oclIsUndefined() )
   then
      if st = 'open'
      then
         #bst_open
      else
         if st = 'closed'
         then
            #bst_closed
         else
            if st = 'skipped'
            then
               #bst_skipped
            else
               #bst_open
            endif
         endif
      endif
   else
      #bst_open
   endif;



-- Rule 'Workbook2ControlsSequence'
-- This rule generates the sequence of controls which is the
-- root element of the "SoftwareQualityControl" metamodel
rule Workbook2ControlsSequence {
   from
      wb : SpreadsheetMLSimplified!Workbook

   to
      cs : SoftwareQualityControl!ControlsSequence (
         controls <- wb.wb_worksheets->collect(e | thisModule.resolveTemp(e, 'c'))
      )
}


-- Rule 'Worksheet2Control'
-- This rule generates the controls that are contained in the controls sequence.
-- Each worksheet of the Excel workbook describes a specific control.
rule Worksheet2Control {
   from
      ws : SpreadsheetMLSimplified!Worksheet

   using {
      controlDate : SpreadsheetMLSimplified!DateTimeType = ws.ws_table.t_rows-
>at(7).r_cells->at(2).c_data.value.value;
   }

   to
      c : SoftwareQualityControl!Control (
         responsible <- ws.getStringValueByRowAndCell(1,2),
         component <- ws.getStringValueByRowAndCell(3,2),
         developmentPhase <- ws.getStringValueByRowAndCell(5,2),
         scope <- ws.getStringValueByRowAndCell(6,2),
         date <- d,
         controlledElt <- ws.getOptStringValueByRowAndCell(2,3),
         eltRef <- ws.getOptStringValueByRowAndCell(4,1),
         eltAuthor <- ws.getOptStringValueByRowAndCell(6,3),
         formRef <- ws.getOptStringValueByRowAndCell(2,4)
      ),
      d : SoftwareQualityControl!Date (
         day <- controlDate.day,
         month <- controlDate.month,
         year <- controlDate.year
      )
}


-- Rule 'BugTracking'
-- This rule generates the Bug Tracking's type of control
-- from the "Type of control field" of the worksheet.
rule BugTracking {
   from
      btt : SpreadsheetMLSimplified!Table (
         btt.t_rows->at(2).r_cells->at(2).c_data.value.value='Bug Tracking'
      )

   to
      bt : SoftwareQualityControl!BugTracking (
         ct_control <- Sequence{btt.t_worksheet}->collect(e | 
thisModule.resolveTemp(e, 'c'))->first()
      )
}

-- Rule 'Bug'
-- This rule generates a bug from a line of the worksheet's bugs table.
rule Bug {
   from
      row : SpreadsheetMLSimplified!Row (
         let d : SpreadsheetMLSimplified!Data = row.r_cells->at(1).c_data
         in
            if d.oclIsUndefined()
            then
               false
            else
               d.value.oclIsTypeOf(SpreadsheetMLSimplified!NumberValue)
            endif
      )

   to
      bug : SoftwareQualityControl!Bug(
         b_bugTracking <- Sequence{row.r_table}->collect(e | 
thisModule.resolveTemp(e, 'bt'))->first(),
         number <- row.r_cells->at(1).c_data.value.value.round(),
         componentVersion <- row.getStringValueByColumn(2),
         description <- row.getStringValueByColumn(3),
         status <- thisModule.getBugStatus(row.getStringValueByColumn(4)),
         originator <- row.getStringValueByColumn(5),
         responsible <- row.getStringValueByColumn(6),
         commentsAnswers <- row.getStringValueByColumn(7),
         openDate <- row.getStringValueByColumn(8),
         closeDate <- row.getStringValueByColumn(9)
      )
}
```

I. 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;

}
```

II. SoftwareQualityControl metamodel in KM3 format

```
-- @name   SoftwareQualityControl
-- @version  1.0
-- @domains  Software, Quality control, Software life cycle
-- @authors  Hugo Bruneliere (hugo.bruneliere@gmail.com)
-- @date   2005/07/04
-- @description  This metamodel describes a simple structure to manage software
quality control and especially bug tracking. It is based on a simple Excel table
representation.

package SoftwareQualityControl {

   -- @begin Controls' general information

   -- @comment Defines the format for the dates (DD/MM/YY).
   class Date {
      attribute day : Integer;
      attribute month : Integer;
      attribute year : Integer;
   }

   -- @comment Defines a sequence of controls. This is the root container.
   class ControlsSequence {
      reference controls[*] ordered container : Control oppositeOf
c_controlsSequence;
   }

   -- @comment Defines a control (general information, type, details...)
   class Control {
      reference c_controlsSequence : ControlsSequence oppositeOf controls;

      -- @comment The surname and name of the person who is responsible for this
control.
      attribute responsible : String;
      -- @comment The name of the component which is concerned by this control.
      attribute component : String;
      -- @comment The name of the development phase during which the control takes
place.
      attribute developmentPhase : String;
      -- @comment The scope of this control, for example "Exhaustive".
      attribute scope : String;
      -- @comment The date of this control (in the format : DD/MM/YY).
      reference date container : Date;
      -- @comment The name of the specific element which is controlled.
      attribute controlledElt[0-1] : String;
      -- @comment The reference of this specific element.
      attribute eltRef[0-1] : String;
      -- @comment The author's name of this specific element.
      attribute eltAuthor[0-1] : String;
      -- @comment The form reference for this control.
      attribute formRef[0-1] : String;

      -- @comment The type of this control. The data contained in a "Control"
element depends on the type of this control.
      reference type : ControlType oppositeOf ct_control;
   }

   -- @end Controls' general information
   -- @begin Specific information for types of control

   -- @comment Defines the abstract concept of type of control. It exists several
types of control. Each class which represents a type of control must inherit of
this class.
   abstract class ControlType {
      reference ct_control[*] : Control oppositeOf type;
   }

   -- @comment Defines a special control type which is bug tracking.
   class BugTracking extends ControlType {
      -- @comment Represents the different bugs tracked during the control.
      reference bugs[*] ordered container : Bug oppositeOf b_bugTracking;
   }

   -- @comment Defines a bug and the associated information.
   class Bug {
      reference b_bugTracking : BugTracking oppositeOf bugs;

      -- @comment The bug identification number
      attribute number : Integer;
      -- @comment The version of the component from which the bug has been detected.
      attribute componentVersion : String;
      -- @comment The complete description of the bug.
      attribute description : String;
      -- @comment The current status of the bug
      attribute status : BugStatusType;
      -- @comment The name of the person who find the bug.
      attribute originator : String;
      -- @comment The name of the person who is responsible for this bug.
      attribute responsible[0-1] : String;
      -- @comment Special comments or possible answers to correct this bug.
      attribute commentsAnswers[0-1] : String;
      -- @comment The date when the bug has been indexed.
      attribute openDate : String;
      -- @comment The date when the bug has been resolved.
      attribute closeDate[0-1] : String;
   }

   -- @comment Defines the type of status for a bug.
   enumeration BugStatusType {
      literal bst_open;
      literal bst_closed;
      literal bst_skipped;
   }

   -- @end Specific information for types of control

}


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

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