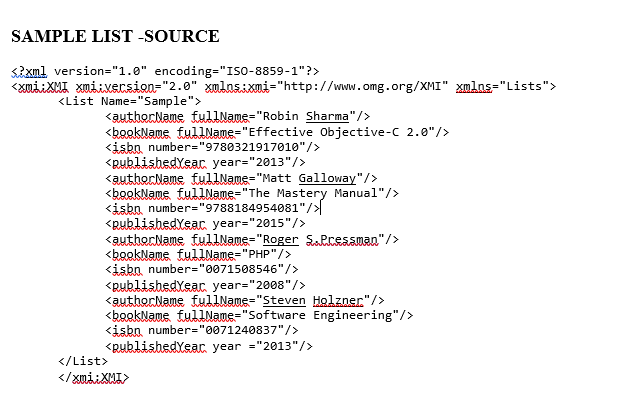
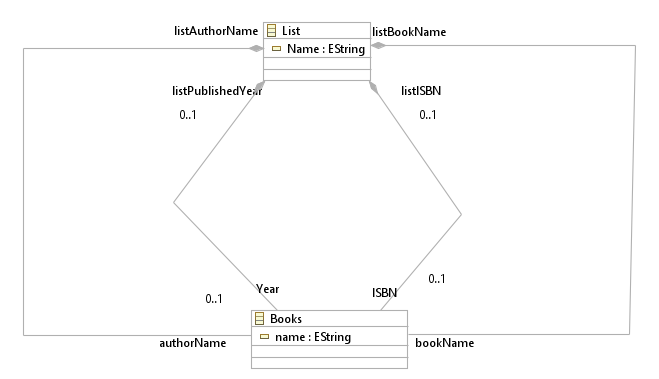
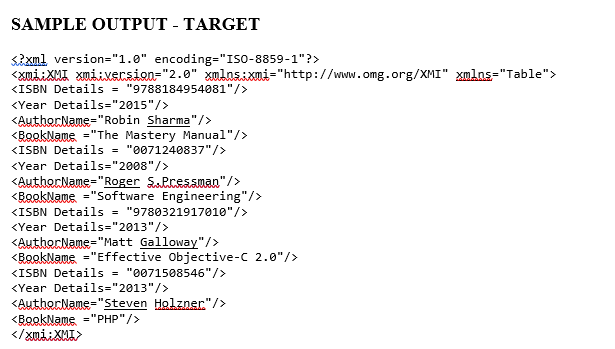
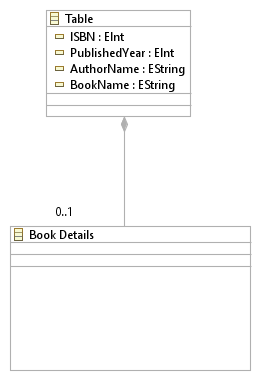
 

**SOURCE MODEL**

**TARGET MODEL**





**EXAMPLE 1**

**ATL Code – List2Details**

module List2Table;

-- @path Lists=/List2Table/List.ecore

-- @path Table=/List2Table/Table.ecore

create OUT: Table from IN: Lists;

helper context Lists!Books def: Name: String =

if not self.listAuthorName.oclIsUndefined() then

self.listAuthorName.Name

else

if not self.listBookName.oclIsUndefined() then

self.listBookName.Name

else

if not self.listISBN.oclIsUndefined() then

self.listISBN.Name

else

self.listPublishedYear.Name

endif

endif

endif;

rule Books2Details {

from

s:Lists!Table (not s.isDetails())

to

t:Lists!BookName(

completeName <- s.authorName + ' '

)

}

**SAMPLE LIST -SOURCE**

<?xml version="1.0" encoding="ISO-8859-1"?>

<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="Lists">

<List Name="Sample">

<authorName fullName="Robin Sharma"/>

<bookName fullName="Effective Objective-C 2.0"/>

<isbn number="9780321917010"/>

<publishedYear year="2013"/>

<authorName fullName="Matt Galloway"/>

<bookName fullName="The Mastery Manual"/>

<isbn number="9788184954081"/>

<publishedYear year="2015"/>

<authorName fullName="Roger S.Pressman"/>

<bookName fullName="PHP"/>

<isbn number="0071508546"/>

<publishedYear year="2008"/>

<authorName fullName="Steven Holzner"/>

<bookName fullName="Software Engineering"/>

<isbn number="0071240837"/>

<publishedYear year ="2013"/>

</List>

</xmi:XMI>

**Sample Output - Target**

<?xml version="1.0" encoding="ISO-8859-1"?>

<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="Table">

<ISBN Details = "9788184954081"/>

<Year Details="2015"/>

<AuthorName="Robin Sharma"/>

<BookName ="The Mastery Manual"/>

<ISBN Details = "0071240837"/>

<Year Details="2008"/>

<AuthorName="Roger S.Pressman"/>

<BookName ="Software Engineering"/>

<ISBN Details = "9780321917010"/>

<Year Details="2013"/>

<AuthorName="Matt Galloway"/>

<BookName ="Effective Objective-C 2.0"/>

<ISBN Details = "0071508546"/>

<Year Details="2013"/>

<AuthorName="Steven Holzner"/>

<BookName ="PHP"/>

</xmi:XMI>

**Example 2**

**SpreadsheetMLSimplified2XML**

**module** SpreadsheetMLSimplified2XML; -- Module Template

**create** OUT : XML **from** IN : SpreadsheetMLSimplified;

-- This helper permits to obtain the string associated

-- to a DateTimeType value.

-- CONTEXT: n/a

-- RETURN: String

**helper** **def**: getDateTimeStringValue(dtv : SpreadsheetMLSimplified!DateTimeType) : String =

dtv.year.toString() + '-' + dtv.month.toString() + '-' + dtv.day.toString() + 'T'

+ dtv.hour.toString() + ':' + dtv.minute.toString() + ':' + dtv.second.toString() + '.000';

-- Rule 'DocumentRoot'.

-- This rule generates the root element of an Excel xml file

-- which is the "Workbook" element

**rule** DocumentRoot {

**from**

wb : SpreadsheetMLSimplified!Workbook

**to**

r : XML!Root(

name<-'Workbook',

value <- '',

children <- Sequence{ att1,att2,

wb.wb\_worksheets->collect(e | **thisModule**.resolveTemp(e, 'wsElt')) }

),

att1 : XML!Attribute (

name <- 'xmlns',

value <- 'urn:schemas-microsoft-com:office:spreadsheet'

),

att2 : XML!Attribute (

name <- 'xmlns:ss',

value <-'urn:schemas-microsoft-com:office:spreadsheet'

)

}

-- Rule 'Worksheets'.

-- This rule generates the different "Worksheet" elements

-- contained in a "Workbook" element

**rule** Worksheets {

**from**

ws : SpreadsheetMLSimplified!Worksheet

**to**

wsElt : XML!Element (

name <- 'Worksheet',

children <- Sequence{nameAtt,Sequence{ws.ws\_table}->collect(e | **thisModule**.resolveTemp(e, 'tElt'))->first()}

),

nameAtt : XML!Attribute (

name <- 'ss:Name',

value <- ws.name,

parent <- wsElt

)

}

-- Rule 'WorksheetTable'.

-- This rule generates the "Table" element

-- contained in a "Worksheet" element

**rule** WorksheetTable {

**from**

t : SpreadsheetMLSimplified!Table

**to**

tElt : XML!Element (

name <- 'Table',

children <- Sequence{

t.t\_cols->collect(e | **thisModule**.resolveTemp(e, 'colElt')),

t.t\_rows->collect(e | **thisModule**.resolveTemp(e, 'rowElt'))

}

)

}

-- Rule 'TableColumn'.

-- This rule generates the "Column" elements

-- contained in a "Table" element

**rule** TableColumn {

**from**

col : SpreadsheetMLSimplified!Column

**using** {

widthOrNot : Sequence(String) =

**let** wdh : Real = col.width

**in**

**if** wdh.oclIsUndefined()

**then**

Sequence{}

**else**

Sequence{wdh.toString()}

**endif**;

}

**to**

colElt : XML!Element (

name <- 'Column',

children <- Sequence{colWidth}

),

colWidth : **distinct** XML!Attribute **foreach**(widthValue **in** widthOrNot)(

name <- 'ss:Width',

value <- widthValue

)

}

-- Rule 'TableRow'.

-- This rule generates the "Row" elements

-- contained in a "Table" element

**rule** TableRow {

**from**

row : SpreadsheetMLSimplified!Row

**to**

rowElt : XML!Element (

name <- 'Row',

children <- Sequence{row.r\_cells->collect(e | **thisModule**.resolveTemp(e, 'cellElt'))}

)

}

-- Rule 'RowCell'.

-- This rule generates the "Cell" elements

-- contained in a "Row" element

**rule** RowCell {

**from**

cell : SpreadsheetMLSimplified!Cell

**to**

cellElt : XML!Element (

name <- 'Cell',

children <- Sequence{

Sequence{cell.c\_data}->collect(e | **thisModule**.resolveTemp(e, 'dataElt'))->first()

}

)

}

-- Rule 'CellData'.

-- This rule generates the "Data" element

-- contained in a "Cell" element

**rule** CellData {

**from**

data : SpreadsheetMLSimplified!Data

**to**

dataElt : XML!Element (

name <- 'Data'

)

}

-- Rule 'DataStringValue'.

-- This rule generates the string value

-- associated to a "Data" element

**rule** DataStringValue {

**from**

strVal: SpreadsheetMLSimplified!StringValue

**to**

strValAtt : XML!Attribute (

parent <- Sequence{strVal.vt\_data}->collect(e | **thisModule**.resolveTemp(e, 'dataElt'))->first(),

name <- 'ss:Type',

value <- 'String'

),

strValTxt : XML!Text (

parent <- Sequence{strVal.vt\_data}->collect(e | **thisModule**.resolveTemp(e, 'dataElt'))->first(),

value <- strVal.value

)

}

-- Rule 'DataNumberValue'.

-- This rule generates the number value

-- associated to a "Data" element

**rule** DataNumberValue {

**from**

numVal: SpreadsheetMLSimplified!NumberValue

**to**

numValAtt : XML!Attribute (

parent <- Sequence{numVal.vt\_data}->collect(e | **thisModule**.resolveTemp(e, 'dataElt'))->first(),

name <- 'ss:Type',

value <- 'Number'

),

numValTxt : XML!Text (

parent <- Sequence{numVal.vt\_data}->collect(e | **thisModule**.resolveTemp(e, 'dataElt'))->first(),

value <- numVal.value.toString()

)

}

-- Rule 'DataBooleanValue'.

-- This rule generates the boolean value

-- associated to a "Data" element

**rule** DataBooleanValue {

**from**

boolVal: SpreadsheetMLSimplified!BooleanValue

**to**

boolValAtt : XML!Attribute (

parent <- Sequence{boolVal.vt\_data}->collect(e | **thisModule**.resolveTemp(e, 'dataElt'))->first(),

name <- 'ss:Type',

value <- 'Boolean'

),

boolValTxt : XML!Text (

parent <- Sequence{boolVal.vt\_data}->collect(e | **thisModule**.resolveTemp(e, 'dataElt'))->first(),

value <- boolVal.value.toString()

)

}

-- Rule 'DataErrorValue'.

-- This rule generates the error value

-- associated to a "Data" element

**rule** DataErrorValue {

**from**

errVal: SpreadsheetMLSimplified!ErrorValue

**to**

errValAtt : XML!Attribute (

parent <- Sequence{errVal.vt\_data}->collect(e | **thisModule**.resolveTemp(e, 'dataElt'))->first(),

name <- 'ss:Type',

value <- 'Error'

)

}

-- Rule 'DataDateTimeValue'.

-- This rule generates the date/time value

-- associated to a "Data" element

**rule** DataDateTimeValue {

**from**

dtVal: SpreadsheetMLSimplified!DateTimeTypeValue

**to**

dtValAtt : XML!Attribute (

parent <- Sequence{dtVal.vt\_data}->collect(e | **thisModule**.resolveTemp(e, 'dataElt'))->first(),

name <- 'ss:Type',

value <- 'DateTime'

),

dtValTxt : XML!Text (

parent <- Sequence{dtVal.vt\_data}->collect(e | **thisModule**.resolveTemp(e, 'dataElt'))->first(),

value <- **thisModule**.getDateTimeStringValue(dtVal.value)

)

}

**Table2SpreadsheetML**

**module** Table2SpreadsheetMLSimplified; -- Module Template

**create** OUT : SpreadsheetMLSimplified **from** IN : Table;

-- This helper permits to determine whether a string contains a number value or not.

-- The method used in this helper is not exactly correct because it considers as a number

-- a string that can be composed of several '.' characters. It should be improved in order

-- to solve this problem. However, the helper returns the right value in most cases.

-- CONTEXT: n/a

-- RETURN: Boolean

**helper** **context** Table!Cell **def**: isNumber(value : String, itIsFirstChar : Boolean) : Boolean =

**if** value <> ''

**then**

**let** char : String = value.substring(1,1)

**in**

**if**( char = '.' **or** char = '0' **or** char = '1' **or** char = '2' **or** char = '3' **or** char = '4'

**or** char = '5' **or** char = '6' **or** char = '7' **or** char = '8' **or** char = '9')

**then**

**self**.isNumber(value.substring(2,value.size()),false)

**else**

false

**endif**

**else**

**if** itIsFirstChar

**then**

false

**else**

true

**endif**

**endif**;

-- Rule 'Table2ExcelTable'

-- This rule generates the global structure of an Excel document

-- and creates the Excel table

**rule** Table2ExcelTable {

**from**

t : Table!Table

**using** {

tableRow : Sequence(Table!Cell) = t.rows->first().cells;

}

**to**

wb : SpreadsheetMLSimplified!Workbook (

wb\_worksheets <- Sequence{ws}

),

ws : SpreadsheetMLSimplified!Worksheet (

name <- 'Java source code Info',

ws\_table <- et

),

et : SpreadsheetMLSimplified!Table (

t\_rows <- Sequence{t.rows->collect(e | **thisModule**.resolveTemp(e, 'erow'))},

t\_cols <- Sequence{col}

),

col : **distinct** SpreadsheetMLSimplified!Column **foreach**(cell **in** tableRow)(

width <- 150.0

)

}

-- Rule 'Row2ExcelRow'

-- This rule generates the rows that will contain the cells

**rule** Row2ExcelRow {

**from**

row : Table!Row

**to**

erow : SpreadsheetMLSimplified!Row (

r\_cells <- Sequence{ row.cells->collect(e | **thisModule**.resolveTemp(e, 'ecell'))}

)

}

-- Rule 'Cell2ExcelCell'

-- This rule generates the cells that will contain the data

**rule** Cell2ExcelCell {

**from**

cell : Table!Cell

**using** {

stringTypeOrNot : Sequence(String) =

**let** ct : String = cell.content

**in**

**if** cell.isNumber(ct,true)

**then**

Sequence{}

**else**

Sequence{ct}

**endif**;

numberTypeOrNot : Sequence(Real) =

**let** ct : String = cell.content

**in**

**if** cell.isNumber(ct,true)

**then**

Sequence{ct.toReal()}

**else**

Sequence{}

**endif**;

}

**to**

ecell : SpreadsheetMLSimplified!Cell (

c\_data <- edata

),

edata : SpreadsheetMLSimplified!Data (),

stringVal : **distinct** SpreadsheetMLSimplified!StringValue **foreach**(stringTypeVal **in** stringTypeOrNot) (

vt\_data <- edata,

value <- stringTypeVal

),

numberVal : **distinct** SpreadsheetMLSimplified!NumberValue **foreach**(numberTypeVal **in** numberTypeOrNot) (

vt\_data <- edata,

value <- numberTypeVal

)

}

**XML2Exceltext**

**query** XML2Text = XML!Root.allInstances()

->asSequence()

->first().ExcelFile().writeTo('C:\\Documents and Settings\\Hugo\\Mes documents\\Stage\\eclipse\_workspace\\OfficeTransformations\\OfficeExtractors\\SpreadsheetML2Text\\exampleExcelJavaSource.xml');

**helper** **context** XML!Root **def**: ExcelFile() : String =

'<?xml version="1.0"?>'+'\n'+'<?mso-application progid="Excel.Sheet"?>'+'\n'

+ **self**.toString2('');

**helper** **context** XML!Element **def**: toString2(indent : String) : String =

**let** na : Sequence(XML!Node) =

**self**.children->select(e | **not** e.oclIsKindOf(XML!Attribute)) **in**

**let** a : Sequence(XML!Node) =

**self**.children->select(e | e.oclIsKindOf(XML!Attribute)) **in**

indent + '<' + **self**.name +

a->iterate(e; acc : String = '' |

acc + ' ' + e.toString2()

) +

**if** na->size() > 0 **then**

'>'

+ na->iterate(e; acc : String = '' |

acc +

**if** e.oclIsKindOf(XML!Text) **then**

''

**else**

'\r\n'

**endif**

+ e.toString2(indent +

' ')

) +

**if** na->first().oclIsKindOf(XML!Text) **then**

'</' + **self**.name + '>'

**else**

'\r\n' + indent + '</' + **self**.name + '>'

**endif**

**else**

'/>'

**endif**;

**helper** **context** XML!Attribute **def**: toString2() : String =

**self**.name + '=\"' + **self**.value + '\"';

**helper** **context** XML!Text **def**: toString2() : String =

**self**.value;