	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Table to TabularHTML	Date 2006/08/04

## 1 ATL Transformation Example: Table to TabularHTML

The Table to TabularHTML example describes a transformation from a Table model to an HTML file containing HTML tables.

### 1.1 Transformation overview

The aim of this transformation is to generate an HTML file from the input data contained in a Table model. This file can next be read with an HTML viewer or Internet navigator.

The generation of the output HTML file is realised by a first transformation from Table to HTML. Next, an extraction to an HTML file is necessary. This is done by applying a transformation from HTML to XML and the use of the XML extractor to obtained an XML file, which will be renamed into an .html file.

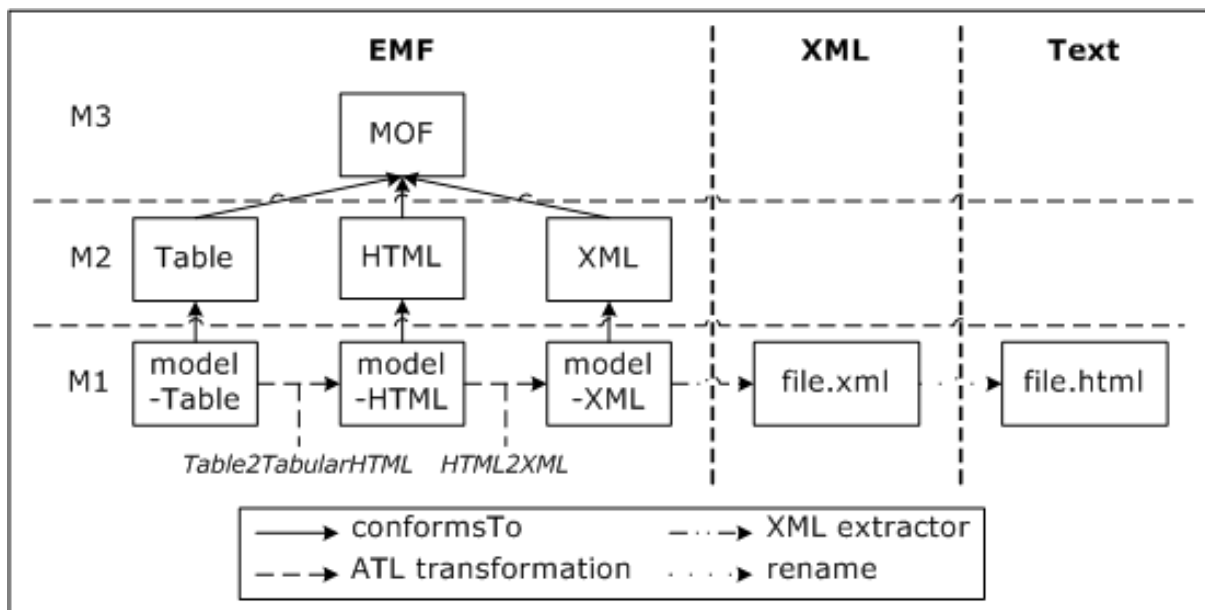


Figure 1: Overview of the transformation

AM3 - Mozilla Firefox

File Edit View Go Bookmarks Tools Help


file://AM3-TabularHTML.html

Go

Metamodel	TNP	TNC	TNC per Package	TNA	TNA per Package	TNA per Class	TNAI	TNAI per Package	TNAI per Class	TNR	TNR per Package	TNR per Class	TNRI	TNRI per Package	TNRI per Class	AIF per Class	DIT	DIT per Package	DIT per Class	NOC	NOC per Package	NOC per Class
AM3	2.0	37.0	18.5	251.0	125.5	6.7	224.0	112.0	6.0	37.0	18.5	1.0	28.0	14.0	0.7	0.8	5.0	2.5	3.0	35.0	17.5	0.9
Package	TNC	TNA	TNA per Class	TNAI	TNAI per Class	TNR	TNR per Class	TNRI	TNRI per Class	AIF	DIT	DIT per Class	NOC	NOC per Class								
AM3	37.0	251.0	6.7	224.0	6.0	37.0	1.0	28.0	0.7	0.8	5.0	3.0	35.0	0.9								
PrimitiveTypes	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0								

Done

Figure 2: Sample of output HTML file

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Table to TabularHTML	Date 2006/08/04

## 1.2 Metamodels

### 1.2.1 Table

The source metamodel of Table is described in Figure 3, and provided in Appendix A in KM3 format.

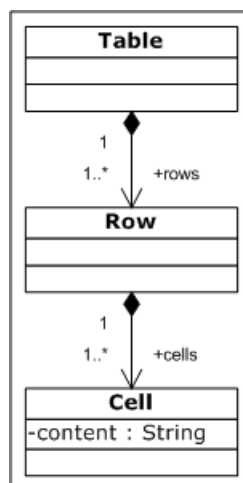



Figure 3: Table metamodel

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.



	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Table to TabularHTML	Date 2006/08/04

### 1.3 Transformation from Table to TabularHTML

#### 1.3.1 Rules specification

These are the rules to transform a Table model to a HTML model.

- For the whole model, the following elements are created:
  - An HTML element composed of a HEAD element and a BODY element.
  - A HEAD element, linked to the HTML element, composed of a TITLE element.
  - A TITLE element, linked to the HEAD element. The value of the title is set to an empty String because the HTML specifications says that "Every HTML document **must** have a TITLE element in the HEAD section."
  - An BODY element, linked to the HTML element.
- For each Table element, the following elements are created:
  - A TABLE element, linked to the unique BODY element, with border attribute set to "1" and composed of several TR elements.
  - A TR element, linked to the TABLE element, for the first Row element and composed of several TH elements.
  - A TH element, linked to the TR element, for each Cell element of the first Row element. The value is set to the content of the Cell element.
- For each Row element, the following elements are created:
  - A TR element, linked to the TABLE element, composed of several TD elements.
- For each Cell element, the following elements are created:
  - A TD element, linked to the TR element. The value is set to the content of the Cell element.

#### 1.3.2 ATL code


This ATL code for the Table2TabularHTML transformation consists in 2 helpers and 5 rules.

The attribute helper `allValidTables` returns all valid sorted tables.

The helper `roundValue` returns the content of a Cell element in a simplified version (one decimal after the dot for a Real).

The entrypoint rule `HTML()` allocates the structure of the HTML file. The rule creates an HTML element ("h") which is composed of a HEAD element ("head") and an empty BODY element ("body"). A TITLE element ("title"), with an empty String value, is also created and associated to the HEAD element.

The lazy rule `Table2TABLE` allocates a TABLE for each Table element. The rule creates a TABLE element ("table") which is composed of a first TR element ("firstRow").

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Table to TabularHTML	Date 2006/08/04

The lazy rule Row2TR allocates a TR for each Row element. The rule creates a TR element ("tr") which is composed of the elements allocated for each of his Cell elements.

The lazy rule Cell2TD and Cell2TH allocate a TD or TH for each Row element. These rules create a TD ("td") or TH ("th") elements with a value equal of the content of the Cell element (simplified for Real).


```
--@name Table2HTML
--@version 1.0
--@domains Table, HTML
--@authors Eric Vepa (eric.vepa <at> gmail.com)
--@date 2006/08/04
--@description This transformation is used to transform generic tables into
      HTML model with tables.

module Table2HTML; -- Module Template
create OUT : HTML from IN : Table;

--@begin attribute helper allValidTables
--@comments returns all valid sorted tables, ie tables which are not for bar or
      pie chart representations
helper def : allValidTables : Sequence(Table!Table) =
  Table!Table.allInstances()->
    asSet()->sortedBy(table|table.rows->first().cells->first().content);
--@end attribute helper allValidTables

--@begin helper roundValue
--@comments for String with a Real content (a dot exists), returns the String
      with only one decimal after the dot
helper context Table!Cell def : roundValue() : String =
  let point : Integer = self.content->indexOf('.') in
    if point > 0
      then self.content->substring(1,point+2)
      else self.content
    endif;
--@end helper roundValue

--@begin entrypoint rule HTML
--@comments only one HTML tag is created with one BODY tag
entrypoint rule HTML() {
  using {
    metamodelTables : Sequence(Table!Table) = thisModule.allValidTables->select
      (table|table.rows->first().cells->first().content = 'Metamodel');
  }
  to
  h:HTML!HTML (
    head <- head,
    body <- body
  ),
  head:HTML!HEAD (
    headElements <- title
  ),
  title:HTML!TITLE (
    value <- 'Your title here'
  ),
}
```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Table to TabularHTML	Date 2006/08/04

```

    body:HTML!BODY (
      bodyElements <- thisModule.allValidTables->iterate(table; acc:Sequence(
        HTML!TABLE)=Sequence{}|
        acc->including(thisModule.Table2TABLE(table)))
    )
}
--@end entrypoint rule HTML


--@begin lazy rule Table2TABLE
--@comments HTML TABLE are added in the BODY of the only HTML tag
lazy rule Table2TABLE {
  from
    tab:Table!Table
  to
    table:HTML!TABLE (
      border <- '1',
      trs <- firstRow,
      trs <- tab.rows->subSequence(2,tab.rows->size())->iterate(row; acc:
        Sequence(HTML!TR)=Sequence{}|
        acc->including(thisModule.Row2TR(row)))
    ),
    firstRow:HTML!TR (
      tds <- tab.rows->first().cells->iterate(cell; acc:Sequence(HTML!TH)=
        Sequence{}|
        acc->including(thisModule.Cell2TH(cell)))
    )
}
--@end lazy rule Table2TABLE

--@begin lazy rule Cell2TH
lazy rule Cell2TH {
  from
    cell:Table!Cell
  to
    th:HTML!TH (
      value <- cell.roundValue()
    )
}
--@end lazy rule Cell2TH

--@begin lazy rule Row2TR
lazy rule Row2TR {
  from
    row:Table!Row
  to
    tr:HTML!TR (
      tds <- row.cells->iterate(cell; acc:Sequence(HTML!TD)=Sequence{}|
        acc->including(thisModule.Cell2TD(cell)))
    )
}
--@end lazy rule Row2TR

--@begin lazy rule Cell2TD
lazy rule Cell2TD {
  from
    cell:Table!Cell
  to

```

	<b>ATL TRANSFORMATION EXAMPLE</b>	<b>Contributor</b> <b>Éric Vépa</b> <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	<b>Table to TabularHTML</b>	<b>Date 2006/08/04</b>

```

    td:HTML!TD (
        value <- cell.roundValue()
    )
}
--@end lazy rule Cell2TD

```

## 1.4 Extractor

### 1.4.1 Rules specifications

These are the rules to transform an HTML model to an XML model (only the rules needed are implemented).

- For the HTML element, a Root element which name is "HTML" is created.
- For each element from HTML metamodel (except HTML element), an Element element is created, which name is the same as the element from HTML metamodel. For instance, an Element element named "TABLE" is created for a TABLE element.
- For each defined attribute of an element of HTML metamodel, the following elements are created:
  - If the name of the attribute is "value", a Text element, linked to the element created for his owner, is created.
  - Otherwise, an Attribute element, linked to the element created for his owner, which name and value are the same as these of the attribute, is created.

### 1.4.2 ATL code

This ATL code for the HTML2XML transformation consists in 9 rules.

The rule HTML2Root allocates a Root for the HTML element. The rule creates a Root element ("root") which is composed of several Element elements.


The 7 next rules allocates an Element for the element of HTML metamodel. Each rule creates an Element element ("elmt") which is composed of Element and Attribute. A Text element is sometimes created and linked to his parent for an attribute named "value".

The last lazy rule Attribute allocates an Attribute element for attribute (whose name is different from "value") of an element of HTML metamodel. The name of the attribute and his defined or default value are required for this lazy rule.

```

--@name HTML2XML
--@version 1.0
--@domains HTML, Table
--@authors Eric Vepa (eric.vepa <at> gmail.com)
--@date 2006/08/04
--@description XML extractor for HTML tabular representation.

```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Table to TabularHTML	Date 2006/08/04

```

module HTML2XML; -- Module Template
create OUT : XML from IN : HTML;

--@begin rule HTML2Root
rule HTML2Root {
  from
    html:HTML!HTML
  to
    root:XML!Root (
      name <- 'HTML',
      children <- html.head,
      children <- html.body
    )
}
--@end rule HTML2Root

--@begin rule HEAD2Element
rule HEAD2Element {
  from
    head:HTML!HEAD
  to
    elmt:XML!Element (
      name <- 'HEAD',
      children <- head.headElements
    )
}
--@end rule HEAD2Element


--@begin rule TITLE2Element
rule TITLE2Element {
  from
    title:HTML!TITLE
  to
    elmt:XML!Element (
      name <- 'TITLE',
      children <- value
    ),
    value:XML!Text (
      value <- if not title.value.ocllsUndefined() then title.value else ''
      endif
    )
}
--@end rule TITLE2Element

--@begin rule BODY2Element
rule BODY2Element {
  from
    body:HTML!BODY
  to
    elmt:XML!Element (
      name <- 'BODY',
      children <- body.bodyElements
    )
}
--@end rule BODY2Element

--@begin rule TABLE2Element

```



	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Table to TabularHTML	Date 2006/08/04

```


rule TABLE2Element {
  from
    table:HTML!TABLE
  to
    elmt:XML!Element (
      name <- 'TABLE',
      children <- thisModule.Attribute('border', if not table.border.
        oclIsUndefined() then table.border else '0' endif),
      children <- table.tr
    )
}
--@end rule TABLE2Element

--@begin rule TR2Element
rule TR2Element {
  from
    tr:HTML!TR
  to
    elmt:XML!Element (
      name <- 'TR',
      children <- tr.tds
    )
}
--@end rule TR2Element


--@begin rule TH2Element
rule TH2Element {
  from
    th:HTML!TH
  to
    elmt:XML!Element (
      name <- 'TH',
      children <- value
    ),
    value:XML!Text (
      value <- if not th.value.oclIsUndefined() then th.value else '' endif
    )
}
--@end rule TH2Element

--@begin rule TD2Element
rule TD2Element {
  from
    td:HTML!TD (
      not td.oclIsTypeOf(HTML!TH)
    )
  to
    elmt:XML!Element (
      name <- 'TD',
      children <- value
    ),
    value:XML!Text (
      value <- if not td.value.oclIsUndefined() then td.value else '' endif
    )
}
--@end rule TD2Element

```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Table to TabularHTML	Date 2006/08/04

```
--@begin lazy rule Attribute
lazy rule Attribute {
  from
    attrName:String,
    attrValue:String
  to
    attr:XML!Attribute (
      name <- attrName,
      value <- attrValue
    )
}
--@end lazy rule Attribute
```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Table to TabularHTML	Date 2006/08/04

## A Appendix: 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;
}
```

## B Appendix : HTML metamodel in KM3 format

```
-- @name    HTML
-- @version 1.0
-- @domains HTML
-- @authors Freddy Allilaire (freddy.allilaire@univ-nantes.fr)
-- @date    2005/04/15
-- @description This basic metamodel describes HyperText Markup Language. HTML
    is the lingua franca for publishing hypertext on the World Wide Web. It is a
    non-proprietary format based upon SGML, and can be created and processed by
    a wide range of tools, from simple plain text editors - you type it in from
    scratch- to sophisticated WYSIWYG authoring tools. HTML uses tags such as <
    h1> and </h1> to structure text into headings, paragraphs, lists, hypertext
    links etc.
-- @see      http://www.w3.org/MarkUp/

package HTML {
    class HTML {
        reference head container : HEAD oppositeOf html;
        reference body container : BODY oppositeOf html;
    }
}
```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Table to TabularHTML	Date 2006/08/04

```

class HTML_Element {
    attribute value : String;
    reference children[*] container : HTML_Element oppositeOf parent
    ;
    reference parent : HTML_Element oppositeOf children;
}

-- @begin HEAD_ELEMENT

class HEAD extends HTML_Element {
    reference headElements[*] container : HEAD_Element oppositeOf
    head;
    reference html : HTML oppositeOf head;
}

abstract class HEAD_Element extends HTML_Element {
    reference head : HEAD oppositeOf headElements;
}

class LINK extends HEAD_Element {
    attribute rel : String;
    attribute title : String;
    attribute ahref : String;
    attribute type : String;
}

class TITLE extends HEAD_Element {
}

-- @end HEAD_ELEMENT

-- @begin BODY_ELEMENT

class BODY extends HTML_Element {
    attribute background : String;
    attribute bgcolor : String;
    attribute text : String;
    attribute link : String;
    attribute alink : String;
    attribute vlink : String;
    reference bodyElements[*] container : BODY_Element oppositeOf
    body;
    reference html : HTML oppositeOf body;
}


abstract class BODY_Element extends HTML_Element {
    reference body : BODY oppositeOf bodyElements;
}

class H1 extends BODY_Element {}

class H2 extends BODY_Element {}

class H3 extends BODY_Element {}

```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Table to TabularHTML	Date 2006/08/04

```

class H4 extends BODYElement {}

class EM extends BODYElement {}

class STRONG extends BODYElement {}

class B extends BODYElement {}

class I extends BODYElement {}

class TT extends BODYElement {}

class PRE extends BODYElement {}

class BIG extends BODYElement {}

class SMALL extends BODYElement {}

class SUB extends BODYElement {}

class SUP extends BODYElement {}

class STRIKE extends BODYElement {}

class FONT extends BODYElement {
    attribute color : String;
    attribute face : String;
    attribute size : String;
}

-- @begin IMG


class IMG extends BODYElement {
    attribute src : String;
    attribute width : String;
    attribute height : String;
    attribute alt : String;
    attribute align : String;
    attribute vspace : String;
    attribute hspace : String;
    attribute ismap : String;
    attribute usemap : String;
    attribute border : String;
}

class BR extends BODYElement {
    attribute clear : String;
}

class MAP extends BODYElement {}

class AREA extends BODYElement {
    attribute shape : String;
    attribute coords : String;

```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Table to TabularHTML	Date 2006/08/04

```

        attribute ahref : String;
    }

-- @end IMG

class STYLE extends BODYElement {}

class EMBED extends BODYElement {
    attribute src : String;
    attribute width : String;
    attribute height : String;
    attribute align : String;
    attribute vspace : String;
    attribute hspace : String;
    attribute border : String;
}

class NOEMBED extends BODYElement {}

class SPAN extends BODYElement {
    attribute style : String;
}

class A extends BODYElement {
    attribute ahref : String;
    attribute name : String;
    attribute id : String;
}

class DIV extends BODYElement {
    attribute align : String;
}

class P extends BODYElement {
}


-- @begin TABLE

abstract class TABLEElement extends BODYElement {
    attribute bgcolor : String;
    attribute background : String;
}

class TABLE extends TABLEElement {
    attribute border : String;
    attribute width : String;
    attribute cellspacing : String;
    attribute cellpadding : String;
    reference trs[*] container : TR oppositeOf table;
}

class TR extends TABLEElement {
    attribute valign : String;
    attribute align : String;
    reference table : TABLE oppositeOf trs;

```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Table to TabularHTML	Date 2006/08/04

```

        reference tds[*] container : TD oppositeOf tr;
    }

    class TD extends TABLEElement {
        attribute colspan : String;
        attribute rowspan : String;
        attribute valign : String;
        attribute align : String;
        attribute width : String;
        reference tr : TR oppositeOf tds;
    }

    class TH extends TD {}

    -- @begin TABLE

    -- @end BODY ELEMENT

    -- @begin FORM

    class FORM {
        attribute action : String;
        attribute method : String;
    }

    class INPUT {
        attribute align : String;
        attribute maxlength : String;
        attribute size : String;
        attribute checked : String;
        attribute src : String;
        attribute inputValue : String;
        attribute name : String;
        attribute type : String;
    }


    class TEXTAREA {
        attribute name : String;
        attribute rows : String;
        attribute cols : String;
    }

    class SELECT {
        attribute multiple : String;
        attribute size : String;
        attribute name : String;
    }

    class OPTION {
        attribute selected : String;
        attribute optionValue : String;
    }

    -- @end FORM

```

	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Table to TabularHTML	Date 2006/08/04

```
-- @begin LIST

abstract class ListElement {
    attribute type : String;
}

    class OL extends ListElement {
        attribute start : String;
    }

    class UL extends ListElement {}

    class LI extends ListElement {
        attribute liValue : String;
    }

    class DL {}

    class DT {}

    class DD {}

-- @begin LIST

-- @begin APPLET

    class APPLET {
        attribute applet : String;
        attribute "class" : String;
        attribute src : String;
        attribute align : String;
        attribute width : String;
        attribute height : String;
    }

class PARAM {
    attribute name : String;
    attribute paramValue : String;
}


class OBJECT {
    attribute classid : String;
    attribute id : String;
    attribute data : String;
    attribute type : String;
    attribute standby : String;
}

-- @end APPLET

-- @begin FRAME

class FRAMESET {
    attribute rows : String;
    attribute cols : String;
```



	ATL TRANSFORMATION EXAMPLE	Contributor Éric Vépa <a href="mailto:eric.vepa@gmail.com">eric.vepa@gmail.com</a>
	Table to TabularHTML	Date 2006/08/04

```

    attribute framespacing : String;
    attribute frameborder : String;
    attribute border : String;
}
class FRAME {
    attribute src : String;
    attribute name : String;
    attribute marginwidth : String;
    attribute marginheight : String;
    attribute scrolling : String;
    attribute noresize : String;
}
class NOFRAME {}

class IFRAME extends FRAME {}

-- @end FRAME
}

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

```