

ATL Transformation Examples

The MySQL to KM3 ATL transformation *- version 0.1 -*

November 2005

by

ATLAS group

LINA & INRIA

Nantes

Content

| | | |
|-------------------|--|-----------|
| 1 | Introduction | 1 |
| 2 | Getting an XML description of a MySQL database..... | 1 |
| 3 | The XML cleaning transformation | 1 |
| 3.1 | <i>Rules specification</i> | 1 |
| 3.2 | <i>ATL code</i> | 2 |
| 4 | The XML2MySQL transformation..... | 2 |
| 4.1 | <i>Rules specification</i> | 2 |
| 4.2 | <i>ATL code</i> | 2 |
| 5 | The MySQL2KM3 transformation | 2 |
| 5.1 | <i>Rules specification</i> | 2 |
| 5.2 | <i>ATL code</i> | 3 |
| 5.3 | <i>Transformation overview</i> | 3 |
| 6 | References | 3 |
| Appendix A | The XML metamodel in KM3 format | 4 |
| Appendix B | The MySQL metamodel in KM3 format | 5 |
| Appendix C | The KM3 metamodel in KM3 format..... | 6 |
| Appendix D | The XML2XML ATL code..... | 8 |
| Appendix E | The XML2MySQL ATL code | 10 |
| Appendix F | The MySQL2KM3 ATL code..... | 15 |



1 Introduction

The MySQL to KM3 transformation describes a transformation from the description of a relational database to metamodel semantics. The example aims to demonstrate the possibility to translate data structure description from the RDBMS to the modelling technical space. For this purpose, we have considered the popular open source MySQL RDBMS system [1] as the database platform, and the KM3 notation as the metamodel description tool [2].

This example is composed of three successive transformations:

- The XML cleaning transformation enables to clean an XML model by removing empty Text elements;
- The XML to MySQL transformation produces a MySQL model from an XML model;
- The MySQL to KM3 transformation produces a KM3 model from a MySQL model.

Note that it is possible to obtain an EMF [3] model from the generated KM3 model by using the dedicated injector available with ADT (ATL Development Tools) [4].

2 Getting an XML description of a MySQL database

An *.xml* file encoding the structure of a MySQL database can be obtained using the MyDB Studio tool [5]. This tool enables to export the structure of a table into a dedicated *.xml* file.

A table description is embedded within a WINDEV_TABLE tag. As the name of the exported table does not appear within the generated file, the WINDEV_TABLE tag is enriched with a “name” attribute that encodes the name of the described table.

For the purpose of this transformation example, the descriptions of the different tables of the considered database have to be grouped into a single *.xml* file. In this scope, a new root tag, WINDEV_DATABASE, has to be added to the file in order to embed the different table description tags. As WINDEV_TABLE, the WINDEV_DATABASE tag has a name attribute that enables to specify a name for the database.

Finally, as table references information is not exported, it is assumed that the “comment” field of each column of a table specifies, if necessary, the remote column it refers to. This kind of reference has to be provided in the following format: *table_name:column_name*.

3 The XML cleaning transformation


This transformation accepts an XML model and produces a new XML model. See Appendix A for the XML metamodel in KM3 format.

The input XML model for this transformation is obtained injecting the *.xml* file into an XML model by means of the ADT facilities [4].

3.1 Rules specification

Here are the rules used to achieve XML model cleaning:

- For each Attribute element, a similar Attribute element is generated;
- For each Root element, a Root element is generated. The generated Root is similar to the input one, except its children that do not include any empty Text entity;

| | | |
|---|----------------------------|-----------------|
|  | ATL Transformation Example | |
| | MySQL to KM3 | Date 02/11/2005 |

- For each Element entity which is of type XML!Element, an Element entity is generated. The generated Element is similar to the input one, except its children that do not include any empty Text entity;
- For each Text element which is not empty, a similar Text element is generated.

3.2 ATL code

ATL code for the XML cleaning transformation may be found in Appendix D.

4 The XML2MySQL transformation

This transformation accepts an XML model as input and returns a MySQL model. See Appendix B for the MySQL metamodel in KM3 format.

4.1 Rules specification

Here are the rules used to generate a MySQL model from an XML model:

- For each Root element, a Database element is generated;
- For each Element entity named "WINDEV_Table", a Table element is generated;
- For each Element entity named "TableInfoTable" that encodes a column with an integer type, an IntegerColumn element is generated;
- For each Element entity named "TableInfoTable" that encodes a column with an enumeration type, an EnumColumn element is generated along with its corresponding EnumSet element and its EnumItem elements;
- For each Element entity named "TableInfoTable" that encodes a column which type is neither integer nor enumeration, a Column element is generated.

4.2 ATL code

ATL code for the XML to MySQL transformation may be found in Appendix E.


5 The MySQL2KM3 transformation

This transformation accepts a MySQL model as input and generates a KM3 model. See Appendix C for the KM3 metamodel in KM3 format.

5.1 Rules specification

Here are the rules used to generate a KM3 model from a MySQL model:

- For each DataBase element, a Metamodel element is generated along with two Package elements (one for the Class elements and one for the PrimitiveType elements);
- For each Table element that does not contain any foreign key column, a Class element is generated;
- For each Table element that contains both foreign and non foreign key columns, a Class element is generated;

| | | |
|---|----------------------------|-----------------|
|  | ATL Transformation Example | |
| | MySQL to KM3 | Date 02/11/2005 |

- For each Table element which has more than two columns that are all non foreign key columns, a Class element is generated;
- For each Column element that does not represent neither a foreign key nor a distinct primitive type, an Attribute element is generated;
- For each Column element that does not represent a foreign key but that corresponds to a distinct primitive type, an Attribute element is generated along with a DataType element;
- For each Column element that represents a foreign key and that belongs to a table only composed of non foreign key columns, a Reference element is generated. Such a Reference has no opposite. Such a Reference has no opposite;
- For each Column element that represents a foreign key and that belongs to a two columns table only composed of foreign key columns, a Reference element is generated. Such a Reference has an opposite, the Reference generated for the other column of the considered table;
- For each Column element that represents a foreign key and that belongs to a table that has more than two columns (which are all foreign keys), a couple of Reference elements are generated. Such References do not have any opposite;
- For each EnumSet element representing a distinct enumeration, an Enumeration element is generated;
- For each EnumItem entity, an EnumLiteral element is generated.

5.2 ATL code

ATL code for the MySQL to KM3 transformation may be found in Appendix F.

5.3 Transformation overview

The KM3 to Metrics transformation is a single step transformation that produces a Metrics model from a KM3 model.

6 References

- [1] MySQL web site. <http://www.mysql.com/>.
- [2] KM3 User Manual. The Eclipse Generative Model Transformer (GMT) project, <http://eclipse.org/gmt/>.
- [3] The Eclipse Modeling Framework (EMF), <http://www.eclipse.org/emf/>.
- [4] The ATL Development Tools (ADT). The Eclipse Generative Model Transformer (GMT) project, <http://eclipse.org/gmt/>.
- [5] MyDB Studio web site. <http://www.mydb-studio.com/>.

Appendix A The XML metamodel in KM3 format

```
1  package XML {
2
3      abstract class Node {
4          attribute startLine[0-1] : Integer;
5          attribute startColumn[0-1] : Integer;
6          attribute endLine[0-1] : Integer;
7          attribute endColumn[0-1] : Integer;
8          attribute name : String;
9          attribute value : String;
10         reference parent[0-1] : Element oppositeOf children;
11     }
12
13     class Attribute extends Node {
14     }
15
16     class Text extends Node {
17     }
18
19     class Element extends Node {
20         reference children[*] ordered container : Node oppositeOf parent;
21     }
22
23     class Root extends Element {
24     }
25 }
```

Appendix B The MySQL metamodel in KM3 format

```
1  package MySQL {
2
3      abstract class NamedElement {
4          attribute name : String;
5      }
6
7      class DataBase extends NamedElement {
8          reference tables[*] container : Table oppositeOf database;
9      }
10
11     class Table extends NamedElement {
12         reference columns[*] ordered container : Column oppositeOf table;
13         reference database : DataBase oppositeOf tables;
14     }
15
16     class Column extends NamedElement {
17         attribute type : String;
18         attribute isPrimaryKey : Boolean;
19         attribute null : Boolean;
20         attribute defaultValue : String;
21         attribute comment : String;
22         reference table : Table oppositeOf columns;
23     }
24
25     class IntegerColumn extends Column {
26         attribute isAutoIncrement : Boolean;
27     }
28
29     class EnumColumn extends Column {
30         reference enumSet container : EnumSet;
31     }
32
33     class EnumSet {
34         reference enumItems[*] container : EnumItem oppositeOf enumSet;
35     }
36
37     class EnumItem extends NamedElement {
38         reference enumSet : EnumSet oppositeOf enumItems;
39     }
40
41 }
```

Appendix C The KM3 metamodel in KM3 format

```
1 package KM3 {
2     abstract class LocatedElement {
3         attribute location : String;
4     }
5
6     abstract class ModelElement extends LocatedElement {
7         attribute name : String;
8         reference "package" : Package oppositeOf contents;
9     }
10
11     class Classifier extends ModelElement {}
12
13     class DataType extends Classifier {}
14
15     class Enumeration extends Classifier {          -- extends DataType in Ecore but if so,
16 cannot use an abstract template in TCS
17         reference literals[*] ordered container : EnumLiteral oppositeOf enum;
18     }
19
20     class EnumLiteral extends ModelElement {
21         reference enum : Enumeration oppositeOf literals;
22     }
23
24 -- WARNING, ONLY FOR OCL Standard Library
25     class TemplateParameter extends Classifier {
26     }
27 -- End WARNING
28
29     class Class extends Classifier {
30 -- WARNING, ONLY FOR OCL Standard Library
31         reference parameters[*] ordered container : TemplateParameter;
32 -- End WARNING
33
34         attribute isAbstract : Boolean;
35         reference supertypes[*] : Class;
36         reference structuralFeatures[*] ordered container : StructuralFeature
37 oppositeOf owner;
38         reference operations[*] ordered container : Operation oppositeOf owner;
39     }
40
41     class TypedElement extends ModelElement {
42         attribute lower : Integer;
43         attribute upper : Integer;
44         attribute isOrdered : Boolean;
45         attribute isUnique : Boolean;
46         reference type : Classifier;
47     }
48
49     class StructuralFeature extends TypedElement {
50         reference owner : Class oppositeOf structuralFeatures;
51         reference subsetOf[*] : StructuralFeature oppositeOf derivedFrom;
52         reference derivedFrom[*] : StructuralFeature oppositeOf subsetOf;
53     }
54
55     class Attribute extends StructuralFeature {}
56
57     class Reference extends StructuralFeature {
58         attribute isContainer : Boolean;
59         reference opposite[0-1] : Reference;
```




```
60     }
61
62     class Operation extends TypedElement {
63         reference owner : Class oppositeOf operations;
64         reference parameters[*] ordered container : Parameter oppositeOf owner;
65     }
66
67     class Parameter extends TypedElement {
68         reference owner : Operation oppositeOf parameters;
69     }
70
71     class Package extends ModelElement {
72         reference contents[*] ordered container : ModelElement oppositeOf "package";
73         reference metamodel : Metamodel oppositeOf contents;
74     }
75
76     class Metamodel extends LocatedElement {
77         reference contents[*] ordered container : Package oppositeOf metamodel;
78     }
79 }
```

Appendix D The XML2XML ATL code

```
1  module XML2XML;
2  create OUT : XML from IN : XML;
3
4
5  -----
6  -- HELPERS -----
7  -----
8
9  -- HELPER:      toKeep
10 -- Returns a boolean stating whether the contextual Node has to be copied from
11 -- the input to the output XML model.
12 -- CONTEXT:     XML!Node
13 -- OUT:         Boolean
14 helper context XML!Node def: toKeep : Boolean =
15     if self.oclIsTypeOf(XML!Text)
16     then
17         self.value.trim() <> ''
18     else
19         false
20     endif;
21
22
23 -----
24 -- RULES -----
25 -----
26
27 -- Rule 'Attribute'
28 -- Copies the input Attribute to the out one.
29 rule Attribute {
30     from
31         i : XML!Attribute
32     to
33         o : XML!Attribute (
34             startLine <- i.startLine,
35             endLine <- i.endLine,
36             startColumn <- i.startColumn,
37             endColumn <- i.endColumn,
38             name <- i.name,
39             value <- i.value,
40             parent <- i.parent
41         )
42 }
43
44 -- Rule 'Text'
45 -- Copies a Text that is not composed of only blank characters.
46 rule Text {
47     from
48         i : XML!Text (
49             i.value.trim() <> ''
50         )
51     to
52         o : XML!Text (
53             startLine <- i.startLine,
54             endLine <- i.endLine,
55             startColumn <- i.startColumn,
56             endColumn <- i.endColumn,
57             name <- i.name,
58             value <- i.value,
59             parent <- i.parent
60         )
61 }
62
63 -- Rule 'Element'
```



ATL Transformation Example

MySQL to KM3

Date 02/11/2005

```
64 -- Copies the input Element to the out one. Children of the generated Element
65 -- are filtered using the toKeep helper.
66 rule Element {
67     from
68         i : XML!Element (
69             i.oclcIsTypeOf(XML!Element)
70         )
71     to
72         o : XML!Element (
73             startLine <- i.startLine,
74             endLine <- i.endLine,
75             startColumn <- i.startColumn,
76             endColumn <- i.endColumn,
77             name <- i.name,
78             value <- i.value,
79             parent <- i.parent,
80             children <- i.children->select(e | e.toKeep)
81         )
82 }
83
84 -- Rule 'Root'
85 -- Copies the input Root to the out one. Children of the generated Element
86 -- are filtered using the toKeep helper.
87 rule Root {
88     from
89         i : XML!Root
90     to
91         o : XML!Root (
92             startLine <- i.startLine,
93             endLine <- i.endLine,
94             startColumn <- i.startColumn,
95             endColumn <- i.endColumn,
96             name <- i.name,
97             value <- i.value,
98             parent <- i.parent,
99             children <- i.children->select(e | e.toKeep)
100         )
101 }
```

Appendix E The XML2MySQL ATL code

```
1  module XML2MySQL;
2  create OUT : MySQL from IN : XML;
3
4
5  -----
6  -- HELPERS -----
7  -----
8
9  -- HELPER:      rootElt
10 -- Returns the root Root element of the XML input model.
11 -- CONTEXT:     thisModule
12 -- OUT:         XML!Root
13 helper def: rootElt : XML!Root =
14     XML!Root.allInstances()->asSequence()->first();
15
16 -- HELPER:      getAttrVal
17 -- Returns a string corresponding to the value of the attribute (identified by
18 -- the string passed as parameter) of the contextual XML!Element.
19 -- CONTEXT:     XML!Element
20 -- IN:          String
21 -- OUT:         String
22 helper context XML!Element def: getAttrVal(name : String) : String =
23     self.children
24         ->select(c | c.oclIsKindOf(XML!Attribute) and c.name = name)
25         ->first().value;
26
27 -- HELPER:      getElementsByName
28 -- Returns the XML!Element corresponding to the children (identified by the
29 -- string passed as parameter) of the contextual XML!Element.
30 -- CONTEXT:     XML!Element
31 -- IN:          String
32 -- OUT:         Set(XML!Element)
33 helper context XML!Element
34     def: getElementsByName(name : String) : Set(XML!Element) =
35         self.children->select(c | c.oclIsKindOf(XML!Element) and c.name = name);
36
37 -- HELPER:      getFirstElementByName
38 -- Returns the XML!Element corresponding to the first child (identified by the
39 -- string passed as parameter) of the contextual XML!Element.
40 -- CONTEXT:     XML!Element
41 -- IN:          String
42 -- OUT:         XML!Element
43 helper context XML!Element
44     def: getFirstElementByName(name : String) : XML!Element =
45         self.getElementsByName(name)->first();
46
47 -- HELPER:      getTextValue()
48 -- Returns a string containing the value of the Text which is the child of the
49 -- contextual XML!Element.
50 -- CONTEXT:     XML!Element
51 -- OUT:         String
52 helper context XML!Element def: getTextValue() : String =
53     if self.children->isEmpty()
54     then
55         ''
56     else
57         if self.children->first().oclIsUndefined()
58         then
59             ''
60         else
61             self.children->first().value
62         endif
63     endif;
```

```

64
65 -- HELPER:      isIntegerType()
66 -- Returns a boolean stating whether the contextual String encodes a MySQL
67 -- integer type.
68 -- CONTEXT:    String
69 -- OUT:        Boolean
70 helper context String def: isIntegerType() : Boolean =
71     self.startsWith('tinyint') or self.startsWith('int');
72
73 -- HELPER:      getItemListRec
74 -- Returns a sequence of strings corresponding to the different EnumItems
75 -- encoded within the contextual String.
76 -- The String passed as parameter contains the EnumItem being parsed.
77 -- CONTEXT:    String
78 -- IN:         String
79 -- OUT:        Sequence(String)
80 helper context String def: getItemListRec(it : String) : Sequence(String) =
81     let char : String = self.substring(1, 1) in
82     if self.size() = 1
83     then
84         Sequence{}
85     else
86         if char = ','
87         then
88             self.substring(2, self.size()).getItemListRec('')
89         else
90             if char = '\'
91             then
92                 if it = ''
93                 then
94                     self.substring(2, self.size()).getItemListRec('')
95                 else
96                     Sequence{
97                         it,
98                         self.substring(2, self.size()).getItemListRec('')
99                     }->flatten()
100             endif
101         else
102             self.substring(2, self.size()).getItemListRec(it.concat(char))
103         endif
104     endif
105     endif;
106
107 -- HELPER:      getItemList
108 -- Returns a sequence of strings corresponding to the different EnumItems encoded
109 -- within the contextual String.
110 -- CONTEXT:    String
111 -- OUT:        Sequence(String)
112 helper context String def: getItemList() : Sequence(String) =
113     let list : String = self.substring(6, self.size()) in
114     list.getItemListRec('');
115
116 -- HELPER:      getTypeNameRec
117 -- Returns a string containing the name of the type encoded by the contextual
118 -- string (recursive helper).
119 -- CONTEXT:    String
120 -- OUT:        String
121 helper context String def: getTypeNameRec() : String =
122     let char : String = self.substring(1, 1) in
123     if self.size() = 1
124     then
125         ''
126     else
127         if char = '(' or char = ' '
128         then
129             ''
130         else
131             char.concat( self.substring(2, self.size()).getTypeNameRec() )
132         endif

```

```

133         endif;
134
135     -- HELPER:     getTypeName()
136     -- Returns a String encoding the name of the type that is contained within the
137     -- contextual String.
138     -- CONTEXT:    String
139     -- OUT:        String
140     helper context String def: getTypeName() : String =
141         self.concat('#').getTypeNameRec();
142
143
144     -----
145     -- RULES -----
146     -----
147
148     -- Rule 'DataBase'
149     -- Creates a DataBase from the root Root element.
150     rule DataBase {
151         from
152             i : XML!Root
153         to
154             o : MySQL!DataBase (
155                 name <- i.getAttrVal('name'),
156                 tables <- XML!Element.allInstances()
157                                     ->select(e | e.name = 'WINDEV_TABLE')
158             )
159     }
160
161
162     -- Rule 'Table'
163     -- Creates a Table from an XML!Element named 'WINDEV_TABLE'.
164     rule Table {
165         from
166             i : XML!Element (
167                 i.name = 'WINDEV_TABLE'
168             )
169         to
170             o : MySQL!Table (
171                 name <- i.getAttrVal('name'),
172                 columns <-
173                     i.getElementsByName('TableInfoTable')->asSequence()
174                                     ->select(e |
175
176                     e.getFirstElementByName('Type').getTextValue().startsWith('tinyint')
177                                     ),
178                 database <- thisModule.rootElt
179             )
180     }
181
182
183     -- Rule 'IntegerColumn'
184     -- Creates an IntegerColumn from an XML!Element named 'TableInfoTable' having
185     -- an integer type.
186     rule IntegerColumn {
187         from
188             i : XML!Element (
189                 if i.name = 'TableInfoTable'
190                 then
191                     i.getFirstElementByName('Type').getTextValue().isIntegerType()
192                 else
193                     false
194                 endif
195             )
196         to
197             o : MySQL!IntegerColumn (
198                 name <- i.getFirstElementByName('Field').getTextValue(),
199                 type <-
200                     i.getFirstElementByName('Type').getTextValue().getTypeName(),
201                 isPrimaryKey <-

```

```

202         i.getFirstElementByName('Key').getTextValue() = 'PRI',
203         null <- i.getFirstElementByName('Null').getTextValue() = 'YES',
204         defaultValue <- i.getFirstElementByName('Default').getTextValue(),
205         comment <- i.getFirstElementByName('Comment').getTextValue(),
206         isAutoIncrement <-
207             i.getFirstElementByName('Extra').getTextValue() =
208 'auto_increment',
209         table <- i.parent
210     )
211 }
212
213
214 -- Rule 'EnumColumn'
215 -- Creates an EnumColumn from an XML!Element named 'TableInfoTable' having
216 -- an enumeration type.
217 rule EnumColumn {
218     from
219         i : XML!Element (
220             if i.name = 'TableInfoTable'
221             then
222
223             i.getFirstElementByName('Type').getTextValue().startsWith('enum')
224             else
225                 false
226             endif
227         )
228     using {
229         items : Sequence(String) =
230             i.getFirstElementByName('Type').getTextValue().getItemList();
231     }
232     to
233         o : MySQL!EnumColumn (
234             name <- i.getFirstElementByName('Field').getTextValue(),
235             type <- 'enum',
236             isPrimaryKey <-
237                 i.getFirstElementByName('Key').getTextValue() = 'PRI',
238             null <- i.getFirstElementByName('Null').getTextValue() = 'YES',
239             defaultValue <- i.getFirstElementByName('Default').getTextValue(),
240             comment <- i.getFirstElementByName('Comment').getTextValue(),
241             table <- i.parent,
242             enumSet <- e1
243         ),
244         e1 : MySQL!EnumSet (
245             enumItems <- e2
246         ),
247         e2 : distinct MySQL!EnumItem foreach(i in items) (
248             name <- i,
249             enumSet <- e1
250         )
251     }
252
253
254 -- Rule 'Column'
255 -- Creates a Column from an XML!Element named 'TableInfoTable' having neither
256 -- an integer nor an enumeration type.
257 rule Column {
258     from
259         i : XML!Element (
260             if i.name = 'TableInfoTable'
261             then
262                 let type : String =
263                     i.getFirstElementByName('Type').getTextValue() in
264                     not type.isIntegerType() and not type.startsWith('enum')
265             else
266                 false
267             endif
268         )
269     to
270         o : MySQL!Column (

```



ATL Transformation Example

MySQL to KM3

Date 02/11/2005

```
271         name <- i.getFirstElementByName('Field').getTextValue(),
272         type <-
273             i.getFirstElementByName('Type').getTextValue().getTypeName(),
274         isPrimaryKey <-
275             i.getFirstElementByName('Key').getTextValue() = 'PRI',
276         null <- i.getFirstElementByName('Null').getTextValue() = 'YES',
277         defaultValue <- i.getFirstElementByName('Default').getTextValue(),
278         comment <- i.getFirstElementByName('Comment').getTextValue(),
279         table <- i.parent
280     )
281 }
```


Appendix F The MySQL2KM3 ATL code

```
1  module MySQL2KM3;
2  create OUT : KM3 from IN : MySQL;
3
4
5  -----
6  -- HELPERS -----
7  -----
8
9  -- HELPER:      databaseElt
10 -- Returns the root Database entity of the input MySQL model.
11 -- CONTEXT:     thisModule
12 -- OUT:         MySQL!DataBase
13 helper def: databaseElt : MySQL!DataBase =
14     MySQL!DataBase.allInstances()->asSequence()->first();
15
16 -- HELPER:      isStringType()
17 -- Returns a boolean stating whether the contextual string encodes a KM3 String
18 -- type.
19 -- CONTEXT:     String
20 -- OUT:         Boolean
21 helper context String def: isStringType() : Boolean =
22     self = 'varchar';
23
24 -- HELPER:      isIntegerType()
25 -- Returns a boolean stating whether the contextual string encodes a KM3
26 -- Integer type.
27 -- CONTEXT:     String
28 -- OUT:         Boolean
29 helper context String def: isIntegerType() : Boolean =
30     self = 'tinyint' or self = 'int';
31
32 -- HELPER:      isDoubleType()
33 -- Returns a boolean stating whether the contextual string encodes a KM3 Double
34 -- type.
35 -- CONTEXT:     String
36 -- OUT:         Boolean
37 helper context String def: isDoubleType() : Boolean =
38     self = 'float' or self = 'double';
39
40 -- HELPER:      isUnsupportedType()
41 -- Returns a boolean stating whether the contextual string encodes a KM3
42 -- Unsupported type.
43 -- CONTEXT:     String
44 -- OUT:         Boolean
45 helper context String def: isUnsupportedType() : Boolean =
46     self = 'date' or self = 'time' or self = 'blob' or self = 'longblob';
47
48 -- HELPER:      km3TypeExistsIn
49 -- Returns a boolean stating whether the KM3 type encoded by the contextual
50 -- MySQL!Column is already defined within the set passed as parameter.
51 -- CONTEXT:     MySQL!Column
52 -- IN:         Set(MySQL!Column)
53 -- OUT:         Boolean
54 helper context MySQL!Column
55     def: km3TypeExistsIn(s: Set(MySQL!Column)) : Boolean =
56     s->iterate(e; res: Boolean = false |
57         if self.type.isStringType()
58         then
59             if e.type.isStringType() or e.type.isUnsupportedType()
60             then
61                 true
62             else
63                 res
```

```

64         endif
65     else
66         if self.type.isIntegerType()
67         then
68             if e.type.isIntegerType()
69             then
70                 true
71             else
72                 res
73             endif
74         else
75             if self.type.isDoubleType()
76             then
77                 if e.type.isDoubleType()
78                 then
79                     true
80                 else
81                     res
82                 endif
83             else
84                 if self.type.isUnsupportedType()
85                 then
86                     if e.type.isStringType() or
87 e.type.isUnsupportedType()
88                     then
89                         true
90                     else
91                         res
92                     endif
93                 else
94                     res
95                 endif
96             endif
97         endif
98     endif
99 );
100
101 -- HELPER:    isForeignKey
102 -- Returns a boolean stating whether the contextual MySQL!Column is a foreign
103 -- key.
104 -- CONTEXT:   MySQL!Column
105 -- OUT:       Boolean
106 helper context MySQL!Column def: isForeignKey : Boolean =
107     self.comment.size() <> 0;
108
109 -- HELPER:    isDefinedIn
110 -- Returns a boolean stating whether the contextual MySQL!EnumItem is also
111 -- defined within the set passed as parameter.
112 -- CONTEXT:   MySQL!EnumItem
113 -- IN:        Set(MySQL!EnumItem)
114 -- OUT:       Boolean
115 helper context MySQL!EnumItem
116 def: isDefinedIn(s: Set(MySQL!EnumItem)) : Boolean =
117     s->iterate(i; res: Boolean = false |
118         if self.name = i.name
119         then
120             true
121         else
122             res
123         endif
124     );
125
126 -- HELPER:    isEquivalentTo
127 -- Returns a boolean stating whether the contextual MySQL!EnumSet is equivalent to
128 -- the MySQL!EnumSet passed as parameter.
129 -- CONTEXT:   MySQL!EnumSet
130 -- IN:        MySQL!EnumSet
131 -- OUT:       Boolean
132 helper context MySQL!EnumSet def: isEquivalentTo(e: MySQL!EnumSet) : Boolean =

```

```

133         if self.enumItems->size() <> e.enumItems->size()
134         then
135             false
136         else
137             self.enumItems->iterate(i; res: Boolean = true |
138                 if i.isDefinedIn( e.enumItems )
139                 then
140                     res
141                 else
142                     false
143                 endif
144             )
145         endif;
146
147 -- HELPER:     enumExistsIn
148 -- Returns a boolean stating whether the contextual MySQL!EnumSet appears in
149 -- the sequence passed as parameter.
150 -- CONTEXT:   MySQL!EnumSet
151 -- IN:        Sequence(MySQL!EnumSet)
152 -- OUT:       Boolean
153 helper context MySQL!EnumSet
154     def: enumExistsIn(s: Sequence(MySQL!EnumSet)) : Boolean =
155         s->iterate(e; res: Boolean = false |
156             if e.isEquivalentTo(self)
157             then
158                 true
159             else
160                 res
161             endif
162         );
163
164 -- HELPER:     enumSet
165 -- Returns a sequence of MySQL!EnumSet that contains one exemplary of the
166 -- different EnumSet defined in the input MySQL model.
167 -- CONTEXT:   thisModule
168 -- OUT:       Sequence(MySQL!EnumSet)
169 helper def: enumSet : Sequence(MySQL!EnumSet) =
170     MySQL!EnumSet.allInstances()
171     ->asSet()
172     ->iterate(e; acc: Sequence(MySQL!EnumSet) = Sequence{} |
173         if not e.enumExistsIn(acc)
174         then
175             acc.append(e)
176         else
177             acc
178         endif
179     );
180
181 -- HELPER:     dbTypeSet
182 -- Returns a set of MySQL!Column that contains one column of the different MySQL
183 -- datatypes present in the input MySQL model.
184 -- CONTEXT:   thisModule
185 -- OUT:       Set(MySQL!Column)
186 helper def: dbTypeSet : Set(MySQL!Column) =
187     MySQL!Column.allInstances()
188     ->select(c | c.type <> 'enum' and not c.isForeignKey)
189     ->asSet();
190
191 -- HELPER:     km3TypeSet
192 -- Returns a set of MySQL!Column that contains one column of the different KM3
193 -- datatypes corresponding to the MySQL datatypes present in the input MySQL
194 -- model.
195 -- CONTEXT:   thisModule
196 -- OUT:       Set(MySQL!Column)
197 helper def: km3TypeSet : Set(MySQL!Column) =
198     thisModule.dbTypeSet
199     ->iterate(c; acc: Set(MySQL!Column) = Set{} |
200         if not c.km3TypeExistsIn(acc)
201         then
    
```



```
202             acc.including(c)
203         else
204             acc
205         endif
206     );
207
208 -- HELPER:     getTableNameRec()
209 -- Returns a string containing the name of the Table encoded by the contextual
210 -- string (recursive helper).
211 -- CONTEXT:   String
212 -- OUT:       String
213 helper context String def: getTableNameRec() : String =
214     let char : String = self.substring(1,1) in
215     if char = ':'
216     then
217         ''
218     else
219         char.concat( self.substring(2, self.size()).getTableNameRec() )
220     endif;
221
222 -- HELPER:     getTableName()
223 -- Returns a string encoding the name of a Table from the contextual string
224 -- that contains the Comment property of a MySQL!Column.
225 -- CONTEXT:   String
226 -- OUT:       String
227 helper context String def: getTableName() : String =
228     self.getTableNameRec();
229
230 -- HELPER:     getReferredTable
231 -- Returns the MySQL!Table that contains the Column that is referred by the
232 -- contextual MySQL!Column.
233 -- CONTEXT:   MySQL!Column
234 -- OUT:       MySQL!Table
235 helper context MySQL!Column def: getReferredTable : MySQL!Table =
236     let t_name : String = self.comment.getTableName() in
237     MySQL!Table.allInstances()
238         ->select(t | t.name = t_name)
239         ->asSequence()->first();
240
241 -- HELPER:     getKM3TypeName()
242 -- Returns a string encoding the KM3 type corresponding to the type encoded by
243 -- the contextual string.
244 -- CONTEXT:   String
245 -- OUT:       String
246 helper context String def: getKM3TypeName() : String =
247     if self.isStringType()
248     then
249         'String'
250     else
251         if self.isIntegerType()
252         then
253             'Integer'
254         else
255             if self.isDoubleType()
256             then
257                 'Double'
258             else
259                 -- Default
260                 'String'
261             endif
262         endif
263     endif;
264
265 -----
266 -- RULES -----
267 -----
268 -----
269
270 -- Rule 'Metamodel'
```

```

271 -- Creates a Metamodel, a 'PrimitiveTypes' Package, and an empty Package from
272 -- the input Database element.
273 rule Metamodel {
274   from
275     i : MySQL!DataBase
276   to
277     o : KM3!Metamodel (
278       location <- '',
279       contents <- Sequence{p, pt}
280     ),
281     p : KM3!Package (
282       location <- '',
283       name <- i.name,
284       package <- OclUndefined,
285       metamodel <- o,
286       contents <- Sequence{}
287     ),
288     pt : KM3!Package (
289       location <- '',
290       name <- 'PrimitiveTypes',
291       package <- OclUndefined,
292       metamodel <- o,
293       contents <-
294         thisModule.km3TypeSet
295         ->collect(e | thisModule.resolveTemp(e, 'd'))
296     )
297 }
298
299
300 -- Rule 'Class1'
301 -- Creates a Class from a Table that contains no foreign key.
302 rule Class1 {
303   from
304     i : MySQL!Table (
305       not i.columns->exists(c | c.isForeignKey)
306     )
307   to
308     o : KM3!Class (
309       location <- '',
310       name <- i.name,
311       package <- thisModule.resolveTemp(thisModule.dataBaseElt, 'p'),
312       isAbstract <- false,
313       supertypes <- Set{},
314       structuralFeatures <-
315         Sequence{
316           i.columns->select(e | not e.isForeignKey),
317           MySQL!Column.allInstances()
318             ->select(c |
319               c.isForeignKey and
320               not c.table.columns
321                 ->exists(e | not e.isForeignKey)
322             and
323               c.table.columns->size() > 2)
324             ->select(c | c.getReferredTable = i)
325             ->collect(r | thisModule.resolveTemp(r, 'o2')),
326           MySQL!Column.allInstances()
327             ->select(c |
328               c.isForeignKey and
329               not c.table.columns->exists(e | not
330 e.isForeignKey) and
331               c.table.columns->size() = 2)
332             ->select(c | c.getReferredTable = i)
333           }->flatten(),
334       operations <- Sequence{}
335     )
336 }
337
338
339 -- Rule 'Class2'

```

```

340 -- Creates a Class from a Table that contains both foreign key and non foreign
341 -- key columns.
342 rule Class2 {
343   from
344     i : MySQL!Table (
345       i.columns->exists(c | c.isForeignKey) and
346       i.columns->exists(c | not c.isForeignKey)
347     )
348   to
349     o : KM3!Class (
350       location <- '',
351       name <- i.name,
352       package <- thisModule.resolveTemp(thisModule.dataBaseElt, 'p'),
353       isAbstract <- false,
354       supertypes <- Set{},
355       structuralFeatures <-
356         Sequence{
357           i.columns,
358           MySQL!Column.allInstances()
359             ->select(c |
360               c.isForeignKey and
361               not c.table.columns
362                 ->exists(e | not e.isForeignKey)
363 and
364                 c.table.columns->size() > 2)
365             ->select(c | c.getReferredTable = i)
366             ->collect(r | thisModule.resolveTemp(r, 'o2')),
367           MySQL!Column.allInstances()
368             ->select(c |
369               c.isForeignKey and
370               not c.table.columns->exists(e | not
371 e.isForeignKey) and
372                 c.table.columns->size() = 2)
373             ->select(c | c.getReferredTable = i)
374           }->flatten(),
375           operations <- Sequence{}
376         )
377   }
378
379
380 -- Rule 'Class3'
381 -- Creates a Class from a Table that contains only no foreign key columns, and
382 -- whose columns number is > 2 .
383 rule Class3 {
384   from
385     i : MySQL!Table (
386       not i.columns->exists(c | not c.isForeignKey) and
387       i.columns->size() > 2
388     )
389   to
390     o : KM3!Class (
391       location <- '',
392       name <- i.name,
393       package <- thisModule.resolveTemp(thisModule.dataBaseElt, 'p'),
394       isAbstract <- false,
395       supertypes <- Set{},
396       structuralFeatures <- i.columns,
397       operations <- Sequence{}
398     )
399   }
400
401
402 -- Rule 'Attribute1'
403 -- Creates an Attribute from a Column that is not a foreign key and that does
404 -- not belong to thisModule.km3TypeSet.
405 rule Attribute1 {
406   from
407     i : MySQL!Column (
408       not i.isForeignKey and

```

```

409         not thisModule.km3TypeSet->exists(c | c = i)
410     )
411     to
412     o : KM3!Attribute (
413         location <- '',
414         name <- i.name,
415         package <- OclUndefined,
416         lower <- 1,
417         upper <- 1,
418         isOrdered <- false,
419         isUnique <- false,
420         type <-
421             if i.type = 'enum'
422             then
423                 thisModule.enumSet
424                 ->select(e | e.isEquivalentTo(i.enumSet))
425                 ->asSequence()->first()
426             else
427                 thisModule.resolveTemp(
428                     thisModule.km3TypeSet
429                     ->select(e |
430                         e.type.getKM3TypeName() =
431 i.type.getKM3TypeName())
432                     ->asSequence()->first(),
433                     'd'
434                 )
435             endif,
436         owner <- i.table,
437         subsetOf <- Set{},
438         derivedFrom <- Set{}
439     )
440 }
441
442
443 -- Rule 'Attribute2'
444 -- Creates an Attribute and a DataType from a Column that is not a foreign key
445 -- but that belongs to thisModule.km3TypeSet.
446 rule Attribute2 {
447     from
448     i : MySQL!Column (
449         not i.isForeignKey and
450         thisModule.km3TypeSet->exists(c | c = i)
451     )
452     to
453     o : KM3!Attribute (
454         location <- '',
455         name <- i.name,
456         package <- OclUndefined,
457         lower <- 1,
458         upper <- 1,
459         isOrdered <- false,
460         isUnique <- false,
461         type <- d,
462         owner <- i.table,
463         subsetOf <- Set{},
464         derivedFrom <- Set{}
465     ),
466     d : KM3!DataType (
467         location <- '',
468         name <- i.type.getKM3TypeName(),
469         package <- thisModule.resolveTemp(thisModule.dataBaseElt, 'pt')
470     )
471 }
472
473
474 -- Rule 'Reference1'
475 -- Creates a Reference from a foreign key Column embedded in a Table that also
476 -- contains non foreign key columns.
477 rule Reference1 {

```



ATL Transformation Example

MySQL to KM3

Date 02/11/2005

```
478         from
479         i : MySQL!Column (
480             i.isForeignKey and
481             i.table.columns->exists(c | not c.isForeignKey)
482         )
483         to
484         o : KM3!Reference (
485             location <- '',
486             name <- i.name,
487             package <- OclUndefined,
488             lower <- 1,
489             upper <- 1,
490             isOrdered <- false,
491             isUnique <- false,
492             type <- i.getReferredTable,
493             owner <- i.table,
494             subsetOf <- Set{},
495             derivedFrom <- Set{},
496             isContainer <- false,
497             opposite <- OclUndefined
498         )
499     }
500
501
502 -- Rule 'Reference2'
503 -- Creates a Reference from a foreign key Column embedded in a 2 columns Table
504 -- that only contains foreign key columns.
505 rule Reference2 {
506     from
507     i : MySQL!Column (
508         i.isForeignKey and
509         not i.table.columns->exists(c | not c.isForeignKey) and
510         i.table.columns->size() = 2
511     )
512     to
513     o : KM3!Reference (
514         location <- '',
515         name <- i.name,
516         package <- OclUndefined,
517         lower <- 0,
518         upper <- 0-1,
519         isOrdered <- false,
520         isUnique <- false,
521         type <- i.getReferredTable,
522         owner <-
523             i.table.columns
524                 ->select(c | c <> i)
525                 ->asSequence()->first().getReferredTable,
526         subsetOf <- Set{},
527         derivedFrom <- Set{},
528         isContainer <- false,
529         opposite <-
530             i.table.columns->select(c | c <> i)->asSequence()->first()
531     )
532 }
533
534
535 -- Rule 'Reference3'
536 -- Creates a couple of References from a foreign key Column embedded in a Table
537 -- with more than 2 columns, and that only contains foreign key columns (such
538 -- tables are created by rule 'Class3').
539 rule Reference3 {
540     from
541     i : MySQL!Column (
542         i.isForeignKey and
543         not i.table.columns->exists(c | not c.isForeignKey) and
544         i.table.columns->size() > 2
545     )
546     to
```



```

547         -- Reference owned by the Table only composed of foreign keys
548     o1 : KM3!Reference (
549         location <- '',
550         name <- i.name,
551         package <- OclUndefined,
552         lower <- 0,
553         upper <- 0-1,
554         isOrdered <- false,
555         isUnique <- false,
556         type <- i.getReferredTable,
557         owner <- i.table,
558         subsetOf <- Set{},
559         derivedFrom <- Set{},
560         isContainer <- false,
561         opposite <- o2
562     ),
563     -- Reference owned by the referred Table
564     o2 : KM3!Reference (
565         location <- '',
566         name <- i.table.name,
567         package <- OclUndefined,
568         lower <- 0,
569         upper <- 0-1,
570         isOrdered <- false,
571         isUnique <- false,
572         type <- i.table,
573         owner <- i.getReferredTable,
574         subsetOf <- Set{},
575         derivedFrom <- Set{},
576         isContainer <- false,
577         opposite <- o1
578     )
579 }
580
581 -- Rule 'Enumeration'
582 -- Creates an Enumeration from an EnumSet that belongs to thisModule.enumSet.
583 rule Enumeration {
584     from
585         i : MySQL!EnumSet (
586             thisModule.enumSet->exists(e | e = i)
587         )
588     to
589         o : KM3!Enumeration (
590             location <- '',
591             name <- 'Enum_'.concat(thisModule.enumSet->indexOf(i).toString()),
592             package <- thisModule.resolveTemp(thisModule.dataBaseElt, 'p'),
593             literals <- i.enumItems
594         )
595 }
596
597 -- Rule 'EnumLiteral'
598 -- Creates an EnumLiteral from an EnumItem defined within an EnumSet that
599 -- belongs to thisModule.enumSet.
600 rule EnumLiteral {
601     from
602         i : MySQL!EnumItem (
603             thisModule.enumSet->exists(e | e = i.enumSet)
604         )
605     to
606         o : KM3!EnumLiteral (
607             location <- '',
608             name <- i.name,
609             package <- OclUndefined
610         )
611 }
612
613 }

```