ATL Transformation Examples

The MySQL to KM3 ATL transformation

- version 0.1 -

November 2005

by

ATLAS group

LINA & INRIA

Nantes

Content

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 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;

- 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;

- 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

```
package XML {

	abstract class Node {
		attribute startLine[0-1] : Integer;
		attribute startColumn[0-1] : Integer;
		attribute endLine[0-1] : Integer;
		attribute endColumn[0-1] : Integer;
		attribute name : String;
		attribute value : String;
		reference parent[0-1] : Element oppositeOf children;
	}

	class Attribute extends Node {
	}

	class Text extends Node {
	}

	class Element extends Node {
		reference children[*] ordered container : Node oppositeOf parent;
	}

	class Root extends Element {
	}
}
```

Appendix B The MySQL metamodel in KM3 format

```
package MySQL {

	abstract class NamedElement {
		attribute name : String;
	}

	class DataBase extends NamedElement {
		reference tables[*] container : Table oppositeOf database;
	}

	class Table extends NamedElement {
		reference columns[*] ordered container : Column oppositeOf table;
		reference database : DataBase oppositeOf tables;
	}

	class Column extends NamedElement {
		attribute type : String;
		attribute isPrimaryKey : Boolean;
		attribute null : Boolean;
		attribute defaultValue : String;
		attribute comment : String;
		reference table : Table oppositeOf columns;
	}

	class IntegerColumn extends Column {
		attribute isAutoIncrement : Boolean;
	}

	class EnumColumn extends Column {
		reference enumSet container : EnumSet;
	}

	class EnumSet {
		reference enumItems[*] container : EnumItem oppositeOf enumSet;
	}

	class EnumItem extends NamedElement {
		reference enumSet : EnumSet oppositeOf enumItems;
	}

}
```

Appendix C The KM3 metamodel in KM3 format

```
package KM3 {
        abstract class LocatedElement {
                attribute location : String;
        }

        abstract class ModelElement extends LocatedElement {
                attribute name : String;
                reference "package" : Package oppositeOf contents;
        }

        class Classifier extends ModelElement {}

        class DataType extends Classifier {}

        class Enumeration extends Classifier {      -- extends DataType in Ecore but if so,
cannot use an abstract template in TCS
                reference literals[*] ordered container : EnumLiteral oppositeOf enum;
        }

        class EnumLiteral extends ModelElement {
                reference enum : Enumeration oppositeOf literals;
        }

-- WARNING, ONLY FOR OCL Standard Library
        class TemplateParameter extends  Classifier {
        }
-- End WARNING

        class Class extends Classifier {
-- WARNING, ONLY FOR OCL Standard Library
                reference parameters[*] ordered container : TemplateParameter;
-- End WARNING

                attribute isAbstract : Boolean;
                reference supertypes[*] : Class;
                reference structuralFeatures[*] ordered container : StructuralFeature
oppositeOf owner;
                reference operations[*] ordered container : Operation oppositeOf owner;
        }

        class TypedElement extends ModelElement {
                attribute lower : Integer;
                attribute upper : Integer;
                attribute isOrdered : Boolean;
                attribute isUnique : Boolean;
                reference type : Classifier;
        }

        class StructuralFeature extends TypedElement {
                reference owner : Class oppositeOf structuralFeatures;
                reference subsetOf[*] : StructuralFeature oppositeOf derivedFrom;
                reference derivedFrom[*] : StructuralFeature oppositeOf subsetOf;
        }

        class Attribute extends StructuralFeature {}

        class Reference extends StructuralFeature {
                attribute isContainer : Boolean;
                reference opposite[0-1] : Reference;
```

```
		}

		class Operation extends TypedElement {
			reference owner : Class oppositeOf operations;
			reference parameters[*] ordered container : Parameter oppositeOf owner;
		}

		class Parameter extends TypedElement {
			reference owner : Operation oppositeOf parameters;
		}

		class Package extends ModelElement {
			reference contents[*] ordered container : ModelElement oppositeOf "package";
			reference metamodel : Metamodel oppositeOf contents;
		}

		class Metamodel extends LocatedElement {
			reference contents[*] ordered container : Package oppositeOf metamodel;
		}
	}
```

Appendix D The XML2XML ATL code

```
module XML2XML;
create OUT : XML from IN : XML;


-------------------------------------------------------------------------------
-- HELPERS --------------------------------------------------------------------
-------------------------------------------------------------------------------

-- HELPER:     toKeep
-- Returns a boolean stating whether the contextual Node has to be copied from
-- the input to the output XML model.
-- CONTEXT:    XML!Node
-- OUT:        Boolean
helper context XML!Node def: toKeep : Boolean =
	if self.oclIsTypeOf(XML!Text)
	then
		self.value.trim() <> ''
	else
		false
	endif;


-------------------------------------------------------------------------------
-- RULES ----------------------------------------------------------------------
-------------------------------------------------------------------------------

-- Rule 'Attribute'
-- Copies the input Attribute to the out one.
rule Attribute {
	from
	 i : XML!Attribute
	to
		o : XML!Attribute (
			startLine <- i.startLine,
			endLine <- i.endLine,
			startColumn <- i.startColumn,
			endColumn <- i.endColumn,
			name <- i.name,
			value <- i.value,
			parent <- i.parent
	)
}

-- Rule 'Text'
-- Copies a Text that is not composed of only blank characters.
rule Text {
	from
	 i : XML!Text (
			i.value.trim() <> ''
		)
	to
		o : XML!Text (
			startLine <- i.startLine,
			endLine <- i.endLine,
			startColumn <- i.startColumn,
			endColumn <- i.endColumn,
			name <- i.name,
			value <- i.value,
			parent <- i.parent
	)
}

-- Rule 'Element'
-- Copies the input Element to the out one. Children of the generated Element
-- are filtered using the toKeep helper.
rule Element {
	from
	 i : XML!Element (
		i.oclIsTypeOf(XML!Element)
	 )
	to
		o : XML!Element (
			startLine <- i.startLine,
			endLine <- i.endLine,
			startColumn <- i.startColumn,
			endColumn <- i.endColumn,
			name <- i.name,
			value <- i.value,
			parent <- i.parent,
			children <- i.children->select(e | e.toKeep)
	)
}

-- Rule 'Root'
-- Copies the input Root to the out one. Children of the generated Element
-- are filtered using the toKeep helper.
rule Root {
	from
	 i : XML!Root
	to
		o : XML!Root (
			startLine <- i.startLine,
			endLine <- i.endLine,
			startColumn <- i.startColumn,
			endColumn <- i.endColumn,
			name <- i.name,
			value <- i.value,
			parent <- i.parent,
			children <- i.children->select(e | e.toKeep)
	)
}
```

Appendix E The XML2MySQL ATL code

```
module XML2MySQL;
create OUT : MySQL from IN : XML;


-------------------------------------------------------------------------------
-- HELPERS --------------------------------------------------------------------
-------------------------------------------------------------------------------

-- HELPER:     rootElt
-- Returns the root Root element of the XML input model.
-- CONTEXT:    thisModule
-- OUT:        XML!Root
helper def: rootElt : XML!Root =
       XML!Root.allInstances()->asSequence()->first();

-- HELPER:     getAttrVal
-- Returns a string corresponding to the value of the attribute (identified by
-- the string passed as parameter) of the contextual XML!Element.
-- CONTEXT:    XML!Element
-- IN:         String
-- OUT:        String
helper context XML!Element def: getAttrVal(name : String) : String =
    self.children
               ->select(c | c.oclIsKindOf(XML!Attribute) and c.name = name)
               ->first().value;

-- HELPER:     getElementsByName
-- Returns the XML!Element corresponding to the children (identified by the
-- string passed as parameter) of the contextual XML!Element.
-- CONTEXT:    XML!Element
-- IN:         String
-- OUT:        Set(XML!Element)
helper context XML!Element
       def: getElementsByName(name : String) : Set(XML!Element) =
       self.children->select(c | c.oclIsKindOf(XML!Element) and c.name = name);

-- HELPER:     getFirstElementByName
-- Returns the XML!Element corresponding to the first child (identified by the
-- string passed as parameter) of the contextual XML!Element.
-- CONTEXT:    XML!Element
-- IN:         String
-- OUT:        XML!Element
helper context XML!Element
       def: getFirstElementByName(name : String) : XML!Element =
       self.getElementsByName(name)->first();

-- HELPER:     getTextValue()
-- Returns a string contraining the value of the Text which is the child of the
-- contextual XML!Element.
-- CONTEXT:    XML!Element
-- OUT:        String
helper context XML!Element def: getTextValue() : String =
       if self.children->isEmpty()
       then
              ''
       else
              if self.children->first().oclIsUndefined()
              then
                     ''
              else
                     self.children->first().value
              endif
       endif;
```

```

-- HELPER:     isIntegerType()
-- Returns a boolean stating whether the contextual String encodes a MySQL
-- integer type.
-- CONTEXT:    String
-- OUT:        Boolean
helper context String def: isIntegerType() : Boolean =
	self.startsWith('tinyint') or self.startsWith('int');

-- HELPER:     getItemListRec
-- Returns a sequence of strings corresponding to the different EnumItems
-- encoded within the contextual String.
-- The String passed as parameter contains the EnumItem being parsed.
-- CONTEXT:    String
-- IN:         String
-- OUT:        Sequence(String)
helper context String def: getItemListRec(it : String) : Sequence(String) =
	let char : String = self.substring(1, 1) in
	if self.size() = 1
	then
		Sequence{}
	else
		if char = ','
		then
			self.substring(2, self.size()).getItemListRec('')
		else
			if char = '\''
			then
				if it = ''
				then
					self.substring(2, self.size()).getItemListRec('')
				else
					Sequence{
						it,
						self.substring(2, self.size()).getItemListRec('')
					}->flatten()
				endif
			else
				self.substring(2, self.size()).getItemListRec(it.concat(char))
			endif
		endif
	endif;

-- HELPER:     getItemList
-- Returns a sequence of strings corresponding to the different EnumItems encoded
-- within the contextual String.
-- CONTEXT:    String
-- OUT:        Sequence(String)
helper context String def: getItemList() : Sequence(String) =
	let list : String = self.substring(6, self.size()) in
	list.getItemListRec('');

-- HELPER:     getTypeNameRec
-- Returns a string containing the name of the type encoded by the contextual
-- string (recursive helper).
-- CONTEXT:    String
-- OUT:        String
helper context String def: getTypeNameRec() : String =
	let char : String = self.substring(1, 1) in
	if self.size() = 1
	then
		''
	else
		if char = '(' or char = ' '
		then
			''
		else
			char.concat( self.substring(2, self.size()).getTypeNameRec() )
		endif
```

```
        endif;

-- HELPER:     getTypeName()
-- Returns a String encoding the name of the type that is contained within the
-- contextual Sring.
-- CONTEXT:    String
-- OUT:        String
helper context String def: getTypeName() : String =
        self.concat('#').getTypeNameRec();


--------------------------------------------------------------------------------
-- RULES -----------------------------------------------------------------------
--------------------------------------------------------------------------------

-- Rule 'DataBase'
-- Creates a DataBase from the root Root element.
rule DataBase {
        from
         i : XML!Root
        to
                o : MySQL!DataBase (
                        name <- i.getAttrVal('name'),
                        tables <- XML!Element.allInstances()
                                                ->select(e | e.name = 'WINDEV_TABLE')
                )
}


-- Rule 'Table'
-- Creates a Table from an XML!Element named 'WINDEV_TABLE'.
rule Table {
        from
         i : XML!Element (
                i.name = 'WINDEV_TABLE'
         )
        to
                o : MySQL!Table (
                        name <- i.getAttrVal('name'),
                        columns <-
                                i.getElementsByName('TableInfoTable')->asSequence()
                                        ->select(e |

        e.getFirstElementByName('Type').getTextValue().startsWith('tinyint')
                                        ),
                        database <- thisModule.rootElt
                )
}


-- Rule 'IntegerColumn'
-- Creates an IntegerColumn from an XML!Element named 'TableInfoTable' having
-- an integer type.
rule IntegerColumn {
        from
         i : XML!Element (
                if i.name = 'TableInfoTable'
                        then
                                i.getFirstElementByName('Type').getTextValue().isIntegerType()
                        else
                                false
                        endif
         )
        to
                o : MySQL!IntegerColumn (
                        name <- i.getFirstElementByName('Field').getTextValue(),
                        type <-
                                i.getFirstElementByName('Type').getTextValue().getTypeName(),
                        isPrimaryKey <-
```

```
                                i.getFirstElementByName('Key').getTextValue() = 'PRI',
                        null <- i.getFirstElementByName('Null').getTextValue() = 'YES',
                        defaultValue <- i.getFirstElementByName('Default').getTextValue(),
                        comment <- i.getFirstElementByName('Comment').getTextValue(),
                        isAutoIncrement <-
                                i.getFirstElementByName('Extra').getTextValue() =
'auto_increment',
                        table <- i.parent
                )
}


-- Rule 'EnumColumn'
-- Creates an EnumColumn from an XML!Element named 'TableInfoTable' having
-- an enumeration type.
rule EnumColumn {
        from
         i : XML!Element (
                if i.name = 'TableInfoTable'
                        then

        i.getFirstElementByName('Type').getTextValue().startsWith('enum')
                        else
                                false
                        endif
         )
        using {
        items : Sequence(String) =
                i.getFirstElementByName('Type').getTextValue().getItemList();
        }
        to
                o : MySQL!EnumColumn (
                        name <- i.getFirstElementByName('Field').getTextValue(),
                        type <- 'enum',
                        isPrimaryKey <-
                                i.getFirstElementByName('Key').getTextValue() = 'PRI',
                        null <- i.getFirstElementByName('Null').getTextValue() = 'YES',
                        defaultValue <- i.getFirstElementByName('Default').getTextValue(),
                        comment <- i.getFirstElementByName('Comment').getTextValue(),
                        table <- i.parent,
                        enumSet <- e1
                ),
                e1 : MySQL!EnumSet (
                        enumItems <- e2
                ),
                e2 : distinct MySQL!EnumItem foreach(i in items) (
                        name <- i,
                        enumSet <- e1
                )
}


-- Rule 'Column'
-- Creates a Column from an XML!Element named 'TableInfoTable' having neither
-- an integer nor an enumeration type.
rule Column {
        from
         i : XML!Element (
                if i.name = 'TableInfoTable'
                        then
                                let type : String =
                                        i.getFirstElementByName('Type').getTextValue() in
                                not type.isIntegerType() and not type.startsWith('enum')
                        else
                                false
                        endif
         )
        to
                o : MySQL!Column (
                        name <- i.getFirstElementByName('Field').getTextValue(),
                        type <-
                                i.getFirstElementByName('Type').getTextValue().getTypeName(),
                        isPrimaryKey <-
                                i.getFirstElementByName('Key').getTextValue() = 'PRI',
                        null <- i.getFirstElementByName('Null').getTextValue() = 'YES',
                        defaultValue <- i.getFirstElementByName('Default').getTextValue(),
                        comment <- i.getFirstElementByName('Comment').getTextValue(),
                        table <- i.parent
        )
}
```

Appendix F The MySQL2KM3 ATL code

```
module MySQL2KM3;
create OUT : KM3 from IN : MySQL;


------------------------------------------------------------------------------
-- HELPERS -------------------------------------------------------------------
------------------------------------------------------------------------------

-- HELPER:     databaseElt
-- Returns the root Database entity of the input MySQM model.
-- CONTEXT:    thisModule
-- OUT:        MySQL!DataBase
helper def: dataBaseElt : MySQL!DataBase =
       MySQL!DataBase.allInstances()->asSequence()->first();

-- HELPER:     isStringType()
-- Returns a boolean stating whether the contextual string encodes a KM3 String
-- type.
-- CONTEXT:    String
-- OUT:        Boolean
helper context String def: isStringType() : Boolean =
       self = 'varchar';

-- HELPER:     isIntegerType()
-- Returns a boolean stating whether the contextual string encodes a KM3
-- Integer type.
-- CONTEXT:    String
-- OUT:        Boolean
helper context String def: isIntegerType() : Boolean =
       self = 'tinyint' or self = 'int';

-- HELPER:     isDoubleType()
-- Returns a boolean stating whether the contextual string encodes a KM3 Double
-- type.
-- CONTEXT:    String
-- OUT:        Boolean
helper context String def: isDoubleType() : Boolean =
       self = 'float' or self = 'double';

-- HELPER:     isUnsupportedType()
-- Returns a boolean stating whether the contextual string encodes a KM3
-- Unsupported type.
-- CONTEXT:    String
-- OUT:        Boolean
helper context String def: isUnsupportedType() : Boolean =
       self = 'date' or self = 'time' or self = 'blob' or self = 'longblob';

-- HELPER:     km3TypeExistsIn
-- Returns a boolean stationg whether the KM3 type encoded by the contextual
-- MySQL!Column is already defined within the set passed as parameter.
-- CONTEXT:    MySQL!Column
-- IN:         Set(MySQL!Column)
-- OUT:        Boolean
helper context MySQL!Column
               def: km3TypeExistsIn(s: Set(MySQL!Column)) : Boolean =
       s->iterate(e; res: Boolean = false |
               if self.type.isStringType()
               then
                       if e.type.isStringType() or e.type.isUnsupportedType()
                       then
                               true
                       else
                               res
```

```
                        endif
                else
                        if self.type.isIntegerType()
                        then
                                if e.type.isIntegerType()
                                then
                                        true
                                else
                                        res
                                endif
                        else
                                if self.type.isDoubleType()
                                then
                                        if e.type.isDoubleType()
                                        then
                                                true
                                        else
                                                res
                                        endif
                                else
                                        if self.type.isUnsupportedType()
                                        then
                                                if e.type.isStringType() or
e.type.isUnsupportedType()
                                                then
                                                        true
                                                else
                                                        res
                                                endif
                                        else
                                                res
                                        endif
                                endif
                        endif
                endif
        );

-- HELPER:      isForeignKey
-- Returns a boolean stating whether the contextual MySQL!Column is a foreign
-- key.
-- CONTEXT:     MySQL!Column
-- OUT:         Boolean
helper context MySQL!Column def: isForeignKey : Boolean =
        self.comment.size() <> 0;

-- HELPER:      isDefinedIn
-- Returns a boolean stating whether the contextual MySQL!EnumItem is also
-- defined within the set passed as parameter.
-- CONTEXT:     MySQL!EnumItem
-- IN:          Set(MySQL!EnumItem)
-- OUT:         Boolean
helper context MySQL!EnumItem
                def: isDefinedIn(s: Set(MySQL!EnumItem)) : Boolean =
        s->iterate(i; res: Boolean = false |
                if self.name = i.name
                then
                        true
                else
                        res
                endif
        );

-- HELPER:      isEquivalentTo
-- Returns a boolean stating whether the contextual MySQL!EnumSet is equivalent to
-- the MySQL!EnumSet passed as parameter.
-- CONTEXT:     MySQL!EnumSet
-- IN:          MySQL!EnumSet
-- OUT:         Boolean
helper context MySQL!EnumSet def: isEquivalentTo(e: MySQL!EnumSet) : Boolean =
    if self.enumItems->size() <> e.enumItems->size()
    then
        false
    else
        self.enumItems->iterate(i; res: Boolean = true |
            if i.isDefinedIn( e.enumItems )
            then
                res
            else
                false
            endif
        )
    endif;

-- HELPER:     enumExistsIn
-- Returns a boolean stating whether the contextual MySQL!EnumSet appears in
-- the sequence passed as parameter.
-- CONTEXT:    MySQL!EnumSet
-- IN:         Sequence(MySQL!EnumSet)
-- OUT:        Boolean
helper context MySQL!EnumSet
        def: enumExistsIn(s: Sequence(MySQL!EnumSet)) : Boolean =
    s->iterate(e; res: Boolean = false |
        if e.isEquivalentTo(self)
        then
            true
        else
            res
        endif
    );

-- HELPER:     enumSet
-- Returns a sequence of MySQL!EnumSet that contains one exemplary of the
-- different EnumSet defined in the input MySQL model.
-- CONTEXT:    thisModule
-- OUT:        Sequence(MySQL!EnumSet)
helper def: enumSet : Sequence(MySQL!EnumSet) =
    MySQL!EnumSet.allInstances()
        ->asSet()
        ->iterate(e; acc: Sequence(MySQL!EnumSet) = Sequence{} |
            if not e.enumExistsIn(acc)
            then
                acc.append(e)
            else
                acc
            endif
        );

-- HELPER:     dbTypeSet
-- Returns a set of MySQL!Column that contains one column of the different MySQL
-- datatypes present in the input MySQL model.
-- CONTEXT:    thisModule
-- OUT:        Set(MySQL!Column)
helper def: dbTypeSet : Set(MySQL!Column) =
    MySQL!Column.allInstances()
        ->select(c | c.type <> 'enum' and not c.isForeignKey)
        ->asSet();

-- HELPER:     km3TypeSet
-- Returns a set of MySQL!Column that contains one column of the different KM3
-- datatypes corresponding to the MySQL datatypes present in the input MySQL
-- model.
-- CONTEXT:    thisModule
-- OUT:        Set(MySQL!Column)
helper def: km3TypeSet : Set(MySQL!Column) =
    thisModule.dbTypeSet
        ->iterate(c; acc: Set(MySQL!Column) = Set{} |
            if not c.km3TypeExistsIn(acc)
            then
```

```
                                acc.including(c)
                        else
                                acc
                        endif
                );

-- HELPER:      getTableNameRec()
-- Returns a string containing the name of the Table encoded by the contextual
-- string (recursive helper).
-- CONTEXT:     String
-- OUT:         String
helper context String def: getTableNameRec() : String =
        let char : String = self.substring(1 ,1) in
        if char = ':'
        then
                ''
        else
                char.concat( self.substring(2, self.size()).getTableNameRec() )
        endif;

-- HELPER:      getTableName()
-- Returns a string encoding the name of a Table from the contextual string
-- that contains the Comment property of a MySQL!Column.
-- CONTEXT:     String
-- OUT:         String
helper context String def: getTableName() : String =
        self.getTableNameRec();

-- HELPER:      getReferredTable
-- Returns the MySQL!Table that contains the Column that is referred by the
-- contexual MySQL!Column.
-- CONTEXT:     MySQL!Column
-- OUT:         MySQL!Table
helper context MySQL!Column def: getReferredTable : MySQL!Table =
        let t_name : String = self.comment.getTableName() in
        MySQL!Table.allInstances()
                ->select(t | t.name = t_name)
                ->asSequence()->first();

-- HELPER:      getKM3TypeName()
-- Returns a string encoding the KM3 type corresponding to the type encoded by
-- the contextual string.
-- CONTEXT: String
-- OUT:         String
helper context String def: getKM3TypeName() : String =
        if self.isStringType()
        then
                'String'
        else
                if self.isIntegerType()
                then
                        'Integer'
                else
                        if self.isDoubleType()
                        then
                                'Double'
                        else
                                -- Default
                                'String'
                        endif
                endif
        endif;


-------------------------------------------------------------------------------
-- RULES ----------------------------------------------------------------------
-------------------------------------------------------------------------------

-- Rule 'Metamodel'
-- Creates a Metamodel, a 'PrimitiveTypes' Package, and an empty Package from
-- the input Database element.
rule Metamodel {
    from
     i : MySQL!DataBase
    to
        o : KM3!Metamodel (
            location <- '',
            contents <- Sequence{p, pt}
        ),
        p : KM3!Package (
            location <- '',
            name <- i.name,
            package <- OclUndefined,
            metamodel <- o,
            contents <- Sequence{}
        ),
        pt : KM3!Package (
            location <- '',
            name <- 'PrimitiveTypes',
            package <- OclUndefined,
            metamodel <- o,
            contents <-
                thisModule.km3TypeSet
                    ->collect(e | thisModule.resolveTemp(e, 'd'))
        )
}


-- Rule 'Class1'
-- Creates a Class from a Table that contains no foreign key.
rule Class1 {
    from
     i : MySQL!Table (
        not i.columns->exists(c | c.isForeignKey)
     )
    to
        o : KM3!Class (
            location <- '',
            name <- i.name,
            package <- thisModule.resolveTemp(thisModule.dataBaseElt, 'p'),
            isAbstract <- false,
            supertypes <- Set{},
            structuralFeatures <-
                Sequence{
                    i.columns->select(e | not e.isForeignKey),
                    MySQL!Column.allInstances()
                        ->select(c |
                            c.isForeignKey and
                            not c.table.columns
                                ->exists(e | not e.isForeignKey)
and
                            c.table.columns->size() > 2)
                        ->select(c | c.getReferredTable = i)
                        ->collect(r | thisModule.resolveTemp(r, 'o2')),
                    MySQL!Column.allInstances()
                        ->select(c |
                            c.isForeignKey and
                            not c.table.columns->exists(e | not
e.isForeignKey) and
                            c.table.columns->size() = 2)
                        ->select(c | c.getReferredTable = i)
                }->flatten(),
            operations <- Sequence{}
        )
}


-- Rule 'Class2'
-- Creates a Class from a Table that contains both foreign key and non foreign
-- key columns.
rule Class2 {
        from
         i : MySQL!Table (
                i.columns->exists(c | c.isForeignKey) and
                i.columns->exists(c | not c.isForeignKey)
         )
        to
                o : KM3!Class (
                        location <- '',
                        name <- i.name,
                        package <- thisModule.resolveTemp(thisModule.dataBaseElt, 'p'),
                        isAbstract <- false,
                        supertypes <- Set{},
                        structuralFeatures <-
                                Sequence{
                                        i.columns,
                                        MySQL!Column.allInstances()
                                                ->select(c |
                                                        c.isForeignKey and
                                                        not c.table.columns
                                                                ->exists(e | not e.isForeignKey)
and
                                                        c.table.columns->size() > 2)
                                                ->select(c | c.getReferredTable = i)
                                                ->collect(r | thisModule.resolveTemp(r, 'o2')),
                                        MySQL!Column.allInstances()
                                                ->select(c |
                                                        c.isForeignKey and
                                                        not c.table.columns->exists(e | not
e.isForeignKey) and
                                                        c.table.columns->size() = 2)
                                                ->select(c | c.getReferredTable = i)
                                }->flatten(),
                        operations <- Sequence{}
                )
}


-- Rule 'Class3'
-- Creates a Class from a Table that contains only no foreign key columns, and
-- whose columns number is > 2 .
rule Class3 {
        from
         i : MySQL!Table (
                        not i.columns->exists(c | not c.isForeignKey) and
                        i.columns->size() > 2
         )
        to
                o : KM3!Class (
                        location <- '',
                        name <- i.name,
                        package <- thisModule.resolveTemp(thisModule.dataBaseElt, 'p'),
                        isAbstract <- false,
                        supertypes <- Set{},
                        structuralFeatures <- i.columns,
                        operations <- Sequence{}
                )
}


-- Rule 'Attribute1'
-- Creates an Attribute from a Column that is not a foreign key and that does
-- not belong to thisModule.km3TypeSet.
rule Attribute1 {
        from
         i : MySQL!Column (
                not i.isForeignKey and
```

```
                        not thisModule.km3TypeSet->exists(c | c = i)
         )
        to
                o : KM3!Attribute (
                        location <- '',
                        name <- i.name,
                        package <- OclUndefined,
                        lower <- 1,
                        upper <- 1,
                        isOrdered <- false,
                        isUnique <- false,
                        type <-
                                if i.type = 'enum'
                                then
                                        thisModule.enumSet
                                                ->select(e | e.isEquivalentTo(i.enumSet))
                                                ->asSequence()->first()
                                else
                                        thisModule.resolveTemp(
                                                thisModule.km3TypeSet
                                                        ->select(e |
                                                                e.type.getKM3TypeName() =
i.type.getKM3TypeName())
                                                        ->asSequence()->first(),
                                                'd'
                                        )
                                endif,
                        owner <- i.table,
                        subsetOf <- Set{},
                        derivedFrom <- Set{}
                )
}


-- Rule 'Attribute2'
-- Creates an Attribute and a DataType from a Column that is not a foreign key
-- but that belongs to thisModule.km3TypeSet.
rule Attribute2 {
        from
         i : MySQL!Column (
                not i.isForeignKey and
                        thisModule.km3TypeSet->exists(c | c = i)
         )
        to
                o : KM3!Attribute (
                        location <- '',
                        name <- i.name,
                        package <- OclUndefined,
                        lower <- 1,
                        upper <- 1,
                        isOrdered <- false,
                        isUnique <- false,
                        type <- d,
                        owner <- i.table,
                        subsetOf <- Set{},
                        derivedFrom <- Set{}
                ),
                d : KM3!DataType (
                        location <- '',
                        name <- i.type.getKM3TypeName(),
                        package <- thisModule.resolveTemp(thisModule.dataBaseElt, 'pt')
                )
}


-- Rule 'Reference1'
-- Creates a Reference from a foreign key Column embedded in a Table that also
-- contains non foreign key columns.
rule Reference1 {
		from
		 i : MySQL!Column (
			i.isForeignKey and
				i.table.columns->exists(c | not c.isForeignKey)
		 )
		to
			o : KM3!Reference (
				location <- '',
				name <- i.name,
				package <- OclUndefined,
				lower <- 1,
				upper <- 1,
				isOrdered <- false,
				isUnique <- false,
				type <- i.getReferredTable,
				owner <- i.table,
				subsetOf <- Set{},
				derivedFrom <- Set{},
				isContainer <- false,
				opposite <- OclUndefined
			)
	}


	-- Rule 'Reference2'
	-- Creates a Reference from a foreign key Column embedded in a 2 columns Table
	-- that only contains foreign key columns.
	rule Reference2 {
		from
		 i : MySQL!Column (
			i.isForeignKey and
				not i.table.columns->exists(c | not c.isForeignKey) and
				i.table.columns->size() = 2
		 )
		to
			o : KM3!Reference (
				location <- '',
				name <- i.name,
				package <- OclUndefined,
				lower <- 0,
				upper <- 0-1,
				isOrdered <- false,
				isUnique <- false,
				type <- i.getReferredTable,
				owner <-
					i.table.columns
						->select(c | c <> i)
						->asSequence()->first().getReferredTable,
				subsetOf <- Set{},
				derivedFrom <- Set{},
				isContainer <- false,
				opposite <-
					i.table.columns->select(c | c <> i)->asSequence()->first()
			)
	}


	-- Rule 'Reference3'
	-- Creates a couple of References from a foreign key Column embedded in a Table
	-- with more than 2 columns, and that only contains foreign key columns (such
	-- tables are created by rule 'Class3').
	rule Reference3 {
		from
		 i : MySQL!Column (
			i.isForeignKey and
				not i.table.columns->exists(c | not c.isForeignKey) and
				i.table.columns->size() > 2
		 )
		to
```

```
			-- Reference owned by the Table only composed of foreign keys
			o1 : KM3!Reference (
				location <- '',
				name <- i.name,
				package <- OclUndefined,
				lower <- 0,
				upper <- 0-1,
				isOrdered <- false,
				isUnique <- false,
				type <- i.getReferredTable,
				owner <- i.table,
				subsetOf <- Set{},
				derivedFrom <- Set{},
				isContainer <- false,
				opposite <- o2
			),
			-- Reference owned by the referred Table
			o2 : KM3!Reference (
				location <- '',
				name <- i.table.name,
				package <- OclUndefined,
				lower <- 0,
				upper <- 0-1,
				isOrdered <- false,
				isUnique <- false,
				type <- i.table,
				owner <- i.getReferredTable,
				subsetOf <- Set{},
				derivedFrom <- Set{},
				isContainer <- false,
				opposite <- o1
			)
}


-- Rule 'Enumeration'
-- Creates an Enumeration from an EnumSet that belongs to thisModule.enumSet.
rule Enumeration {
	from
	 i : MySQL!EnumSet (
		thisModule.enumSet->exists(e | e = i)
	 )
	to
		o : KM3!Enumeration (
			location <- '',
			name <- 'Enum_'.concat(thisModule.enumSet->indexOf(i).toString()),
			package <- thisModule.resolveTemp(thisModule.dataBaseElt, 'p'),
			literals <- i.enumItems
		)
}


-- Rule 'EnumLiteral'
-- Creates an EnumLiteral from an EnumItem defined within an EnumSet that
-- belongs to thisModule.enumSet.
rule EnumLiteral {
	from
	 i : MySQL!EnumItem (
		thisModule.enumSet->exists(e | e = i.enumSet)
	 )
	to
		o : KM3!EnumLiteral (
			location <- '',
			name <- i.name,
			package <- OclUndefined
		)
}
```