International Journal of Engineering &Technology, 7 (4) (2018) 6088-6094

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET
doi: 10.14419/ijet.v7i4.23127
Research paper

Converting Approach from Database Relational Schema into Horn Clauses Predicate Logic

Bilal Hussein [1] *, Firas Abdallah [2], Aref Mehanna [1], Yahia Rabih [2], Shadi Khawandi [2]

[1] Faculty of Technology, Lebanese University, Lebanon
[2] Faculty of Economics and Business Administration–Lebanese University-Beirut, Lebanon
**Corresponding author E-mail: husseinbilal@ul.edu.lb*

Abstract

The aim of this study is to build a transformation model capable to convert a database relational schema into a set of horn clauses predicate logic. Today, databases support trillion and trillion of data megabytes. They provide the main data sources for most of the decision-making systems. Now, organizations give more importance towards knowledge rather than information. But, knowledge cannot be extracted simply from databases. Usually, many complex procedures and techniques of data mining are required. Unlike databases, knowledge bases store knowledge (structured and unstructured information) in a computer-readable form. Moreover, the inference engine, supported by most of the knowledge bases, extracts new knowledge by simply implementing new rules. Based on Model Driven Architecture (MDA) approach and especially the models' transformation, this paper presents a set of mappings which transform a database (metadata and previous data) to knowledgebase expressed into facts and rules. A prototype transformation system is designed and implemented based on UML (Unified Modeling Language) class diagram and ATL (Atlas Transformation Language) which is the tool used for the model transformation. The prototype shows that the output of the transformation process can be operated directly by a logic programming language such as Prolog.

Keywords: *Model Transformation; Model Driven Architecture; Unified Modeling Language; Meta-Model; Atlas Transformation Language.*

1. Introduction

Today, databases support trillion and trillion of data megabytes. They are always considered as the main container of big data. They require new strategies and methods for extracting and analysing data comparing to traditional ones [1]. Analysing data for producing knowledge and decision still a big challenge. In the last years, many methods for extracting knowledge from databases are developed and used. Knowledge cannot be extracted simply from databases. For instance, many complex procedures and techniques of data mining are required. Unlike databases, knowledge bases store knowledge in a computer-readable form and provide the means for the computerized collection, organization, and retrieval of knowledge. Moreover, the inference engine, supported by most of knowledge bases, extracts new knowledge by implementing simply new rules. In this paper, we aim to develop a D2K architecture (Database to Knowledgebase) to transform a Database instance to a Knowledgebase instance. Based on Model Driven Architecture (MDA) approach and especially the models' transformation, this paper presents a set of mappings which transform a database (metadata and previous data) to knowledge base into a set of facts and rules expressed in Horn clauses. The MDA is adopted by the OMG (Object Management Group) [2]. In the MDA software process methodology, a system is developed by refining models. MDA specifies three default models of a system Computation Independent Model (CIM), the Platform Independent Model (PIM) and Platform Specific Model (PSM) [3]. The two key concepts of MDA are models and transformations. Models denoted by source and target models. Transformations are a set of mapping rules that are specified using some language such as Query/View/Transformation (QVT) or Atlas Transformation Language (ATL).

The MDA framework consists of four architecture levels (Fig.1). This architecture is based on the concept of meta-meta-model MOF (MetaObject Facility) [4].

The meta-meta-model level M3 is the most abstract one of this architecture. It defines the basic concepts allowing the representation of lower levels (Meta-Models M2, and Models M1) as well as itself. Today, technologies offer many kinds of meta-meta-models such as MOF (Meta Object facility) standardized by OMG (OMG), eCore defined as part of the Eclipse Modeling Framework [5], UML profiles (Unified Modeling Language), conceptual graphs, sNets, CDIF, OWL [6].

The meta-model level M2 defines the modeling language and the grammar representation of the M1 models while using the vocabulary and grammar specified by the M3 level [7]. Most of the meta-model is described by class diagrams of UML.

The M1 level is the model one. It defines the ontology used for the description of the real world application. While respecting the vocabulary and the grammar specified by its meta-model level M2, an M1 model describes the information of the level M0. Models in M1 are instances of meta-models in M2.

The M0 level is a concrete level of the real-world (any real situation, unique in space and time, represented by a given model from M1).

MDA defines two types of relations between these levels 'conformity' and 'representation of'. For instance, a model at the M1 level is conforming to its meta-model at the M2 level or M0 is a representation of M1. To ensure its usefulness and operationalization by a computer machine, the MDA approach aims the models' validation formally. A model at level M1 is valid if it conforms to

its meta-model at level M2. The conformity rules are described at the meta-meta-model level M3.
Once models are conforming to their meta-models, they can be transformed into another kind of models by specifying the necessary transformation rules. MDA is supported by model transformations. A wide range of languages and tools have been developed [8], for instance, QVT [9] and ATL [10].
In this paper, we design our meta-models (source and target) with UML class diagrams and implement them as Ecore entities [11]. Moreover, the transformation rules are implemented through ATL language.
Our D2K mechanism places a database instance (source model) at the M1 level which conforms to the source meta-model (M2 level) and the knowledge base (target model) at the M1 level which conforms to the target meta-model (M2 level) (Fig. 2).

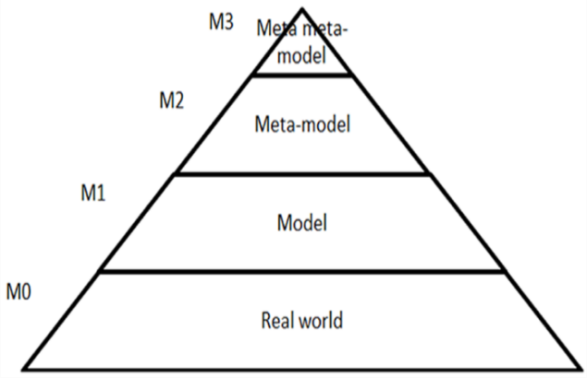

Fig. 1: MDA Framework Architecture Levels.

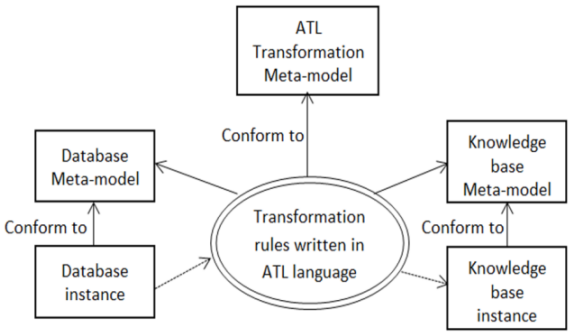

Fig. 2: D2K Architecture.

2. Related works

This section presents related works for transforming the database to the knowledge base or ontology base. Most research focused on transforming database data into Resource Description Framework (RDF) such as [12] and Web Ontology Language (OWL) such as [13 - 17] [23] [24]. Some others generate semantic data from RDB [15] [18]. Few studies have taken into consideration the RDB schema [14] [16] [19]. Another study proposes modeling language for a direct mapping from a database web form to OWL or RDFS [17]. A research conducted by Ishan Rastogi has discussed a method that transforms a database and especially its data into a knowledge base. This transformation uses at first step data sanitization and clustering in order to prepare the information base then creating the knowledge base using Shannon's theorem of entropy [20].
Model transformation techniques of the MDA approach can be implemented to automatically generate parts of the system from models. Models are no longer limited to the documentation of a system but can be part of its structure and definition.
The advantage of these techniques over the methods mentioned above is very effective for the exogenous transformation between two different technological spaces. It is in this sense that we have chosen this approach to transform a database model managed by a database management system into a knowledge base model managed by a knowledge base management system.

3. Materials and methods

This study was conducted at Lebanese University at the faculty of Technology as a part of a project titled 'A prototype of an Online Expert System Based on Knowledge Meta-Model for student guidance and advising'.

3.1. Model transformation from database to knowledge base

3.1.1. Meta-models

A meta-model is a domain specific language (DSL) which is used to express the common concepts for models in the same area. It is built from informal models, recommendations in a natural language and semi-formal models usually written in the Unified Modeling Language (UML). DSLs are specifically designed to answer a particular problem or application domain; they make knowledge explicit. They allow domain experts, understanding, validating, modifying, testing, and sometimes even developing DSL programs [21].
Models transformation is a process that aims to produce a valid model (which conforms to its target meta-model), called target model, from a valid model (which conforms to its source meta-model) called source model. According to MDA, this transformation process consists of two steps. The first step is the specification of the transformation rules describing the correspondence between the concepts of the source and target meta-models. The second step is to apply these transformation rules to all elements of the source model in order to generate a target model.
Our transformation process is based on two meta-models denoted Database Meta-Model (DBMM) and Knowledge Base Meta-Model (KBMM). The DBMM describes a database by its metadata and tuples (Fig. 3), while the KBMM describes a knowledge base by a set of facts and rules specified as Horn clauses (Fig. 4).

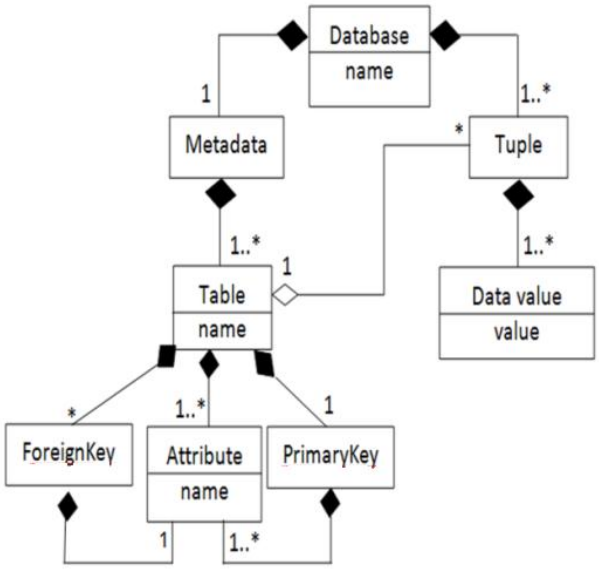

Fig. 3: Database Meta-Model.

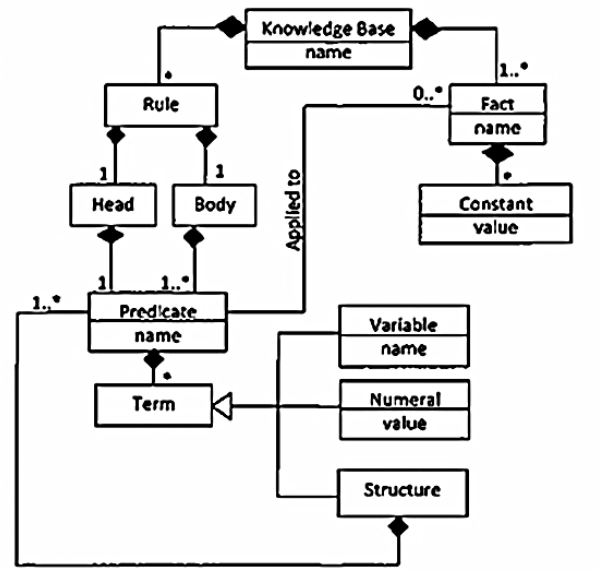

Fig. 4: Knowledge Base Meta-Model.

3.1.2. Mapping database to a knowledgebase

The key concept of our approach transforms each table of the database to a predicate expression which consists of a predicate's name followed by zero or more arguments enclosed in parentheses and separated by commas predicateName(arg1, arg2, …) where the predicate's name is the same as the table's name, and each predicate's argument corresponds to a table's attribute.
Moreover, mapping a database to a knowledge base requires four transformation rules denoted R1, R2, R3, and R4 (Table 1). These rules are applied to each table of the database instance one by one.

Table 1: Transformation Rules

Transformation Rule	Generated Output
R1: transforms the primary key of a table (source) to a knowledge base rule (target) written in the form of a Horn clause	primaryKeyName([K1,K2,..]):- tableName([K1,K2,…],[list of hidden attributes]). Where: primaryKeyName: name of the primary key tableName: name of the table containing the primary key to be transformed [K1,K2,…]: table's primary key consisting of attributes K1,K2,…. [list of hidden attributes] : list of all other attributes of the table foreignKeyName([K1,K2,…],[F1,F2,…]):-childTableName([K1,K2,…],[list of hidden attributes], [F1,F2,…],[list of hidden attributes]), parentTableName([K'1,K'2,…],[list of hidden attributes]), [F1,F2,…]=[K'1,K'2,…]
R2: transforms each foreign key of a table (source) to a knowledge base rule (target) written in the form of a Horn clause	Where: [K1,K2,…]: the primary key of the child's table [F1,F2,…]: foreign key of the child's table [K'1,K'2,…]: the primary key of the parent's table [F1,F2,…]=[K'1,K'2,…]: the relationship between the foreign key of the child's table and the primary key of the parent's table [list of hidden attributes]: list of all other non-key attributes of the table.
R3: transforms each non-key attribute of a table (source) to a knowledge base rule (target) written in the form of a Horn clause	AttributeName([K1,K2,…],A):- tableName([K1,K2,…],[list of hidden attributes] ,A,[list of hidden attributes]). Where: [K1,K2,…]: the primary key of the table tableName. A: attribute's name inserted in its place according to the attribute list of the table [list of hidden attributes]: list of other attribute members of the table before/after the attribute A in the list of attributes
R4: transforms each tuple of a table (i.e. table row) (source) to a knowledge base fact (target)	tableName([list of tuple's data]) Where: [list of tuple's data]: list of attributes values of a tuple in a table

3.1.3. Example of use

The example shows the results of applying rules R1, R2, R3, and R4 on the following database schema (Fig. 5):

Table : Student

Scode	Sname	Saddress	Mcode
100	hachem salem	beirut	1
101	sami audi	saida	2
102	naji touma	tvr	1
103	salim mourad	beirut	3

Table: Major

Mcode	Mname
1	manage ment
2	accounting
3	finance

Fig. 5: Simple Student-Major Database.

This simple database example consists of two tables Student and Major. Student table has a primary key Scode, two non-key attributes Sname and Saddress and one foreign key Mcode. The 'Major' table has a primary key Mcode and a non-key attribute Mname. A relationship allows an integrity reference between the tables Student and Major.

The rule R1 provides two Horn clauses (knowledge base rules):
Scode(K) :- Student(K,_,_,_,_)
Mcode(K) :- Major(K,_,_,_)

The rule R2 provides one Horn clause:
Mcode(K,F):-Student(K,_,_,F), Major(F',_) , F=F'

The rule R3 provides three Horn clauses
Sname (K, A) :- Student(K,A,_,_)
// A is the student's name of the student's code K
Saddress (K, A) :- Student(K,_,A,_)
// A is the student's address of the student's code K
Mname(K,A) :- Major(K,A)
// A is the major's name of the major's code K

The rule R4 provides seven facts (total number of rows in both tables):
Student(100, hachem salem,beirut,1)
Student(101, sami audi,saida,2)
Student(102, naji touma, tyr,1)
Student(103, salim mourad,beirut,3)
Major(1,management)
Major(2,accounting)
Major(3,finance)

Therefore, the transformation process provides the following knowledge base instance:
Scode(K) :- Student(K,_,_,_,_)
Mcode(K) :- Major(K,_,_,_,_)
Mcode(K,F):-Student(K,_,_,F), Major(F',_) , F=F'
Sname(K,A) :- Student(K,A,_,_)
Saddress(K,A) :- Student(K,_,A,_)
Mname(K,A) :- Major(K,A)
Student(100, 'hachem salem',beirut,1)
Student(101, 'sami audi',saida,2)
Student(102, 'naji touma', tyr,1)
Student(103, 'salim mourad',beirut,3)
Major(1,management)
Major(2,accounting)
Major(3,finance)

4. Result and discussion

In this subsection, we present the implementation of our two meta-models Database Meta-Model (DBMM) and Knowledge Base Meta-Model (KBMM) and their associated transformation rules using the Eclipse Modeling Framework (EMF) and the ATL language. Figure 6 provides an overview of the Database to the Knowledgebase ATL transformation. It introduces the name of the files that are going to encode the models (SchoolDB.xmi, SchoolKB.xmi), the meta-models (DBMM.ecore, KBMM.ecore) and the ATL transformation (DB2KB.atl) that will be handled during the design of the database to the knowledgebase ATL transformation. It is important to note that the transformation rules defined in (DB2KB.atl) are conforming to the ATL transformation meta-model. The ATL syntax is detailed in Atlas Transformation Language User Manual [22].

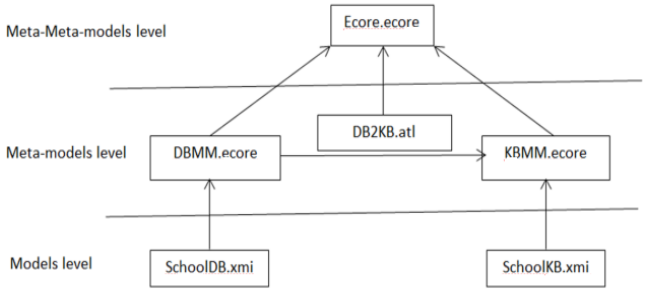

Fig. 6: Overview of the Database to Knowledgebase ATL Transformation.

The implementation of the database to knowledgebase ATL Transformation consists of four steps:

1) Designing the meta-models (source and target) using the Ecore editor. EMF generates an Ecore file holding a meta-model (file.ecore). In our case, we designed the source meta-model DBMM.ecore and the target meta-model KBMM.ecore.
2) Creating transformation rules using the syntax of the ATL language. All rules are stored in an atl file DB2KB.atl.
3) Providing XMI file SchoolDB.xmi as an input model that conforms to the designed input meta-model DBMM.ecore.
4) Generating XMI file SchoolKB.xmi as an output model that conforms to the designed output meta-model KBMM. This file is generated when we launch the execution of the DB2KB.atl file.

a) Designing meta-models

Both meta-models (DBMM and KBMM) are based on the same meta-meta-model Ecore. Firstly, we created the database meta-model DBMM as an Ecore file DBMM.ecore and the knowledge base meta-model KBMM as an Ecore file KBMM.ecore. The Ecore editor allows the describing of Ecore models in a tree-based view (classes, interfaces, attributes, and associations) (Fig. 7):

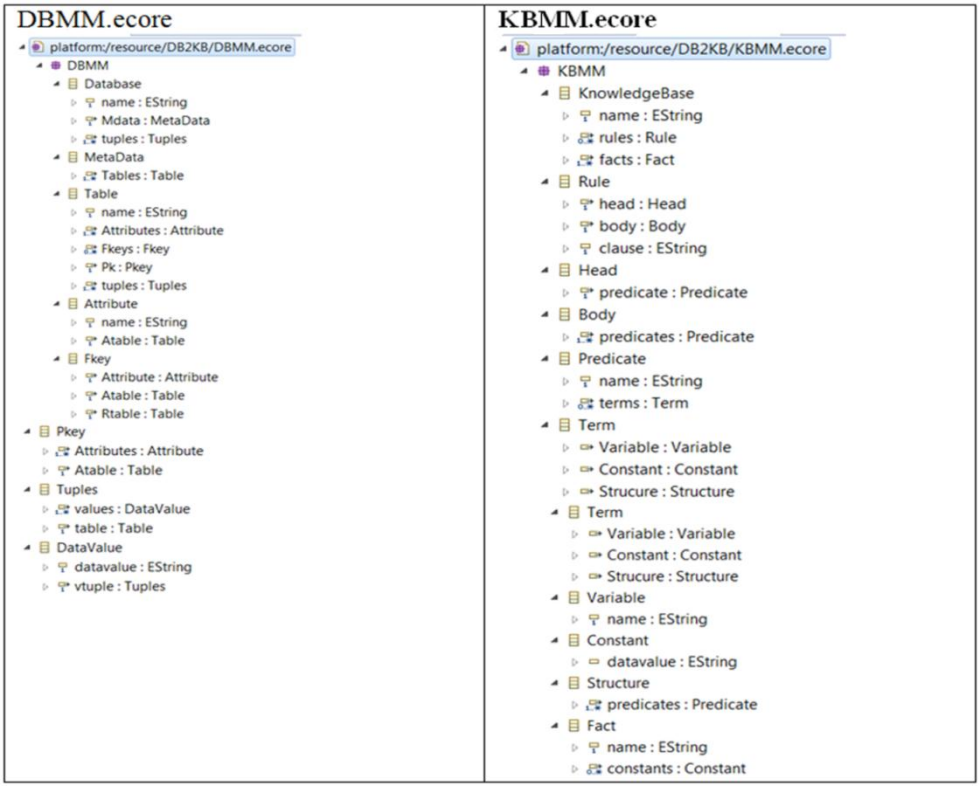

Fig. 7: Database and Knowledgebase Meta-Models.

b) Creating transformation rules

In this section, we list the implementation of rules which describe all different types of transformation presented in our approach. ATL language developed on top of the Eclipse platform that allows programmatically writing and storing all transformation rules in an atl file. Below a short atl file DB2KB.atl that describes some transformation rules.

DB2KB.atl

```
module DB2KB;
create OUT : KBMM from IN : DBMM;
rule Database2KnowledgeBase{
from
s: DBMM!Database
to
q: KBMM!KnowledgeBase (
name<-s.name,
rules<-Sequence{
```

```
thisModule.getTables(s.Mdata)-> collect(e|thisModule.PrimaryKey2Rule(e)),
thisModule.getTables(s.Mdata)->collect(e |e.Attributes->union(e.Fkeys ->
collect(e1| e1.Attribute))->
collect(e2|thisModule.Attribute2Rule(e,e2))),
thisModule.getTables(s.Mdata)->collect(e |e.Fkeys-> asSet()->
collect(e1|thisModule.ForeignKey2Rule(e1)))
},
facts<-Sequence{
thisModule.getTuples(s)-> collect(e|thisModule.Tuple2Fact(e))})
}
lazy rule PrimaryKey2Rule {. . . . }
lazy rule AttributeToTerm {. . . . }
lazy rule Attribute2Rule {. . . . }
lazy rule ForeignKey2Rule {. . . . }
lazy rule Tuple2Fact {. . . . }
lazy rule ConstantToValue {. . . . }
```

c) Providing source model in a xmi file

An input model (source) schooldb.xmi that conforms to the DBMM meta-model is created in an xmi file format as follow:

SchoolDB.xmi

```
<Database name="DBTOKB">
<Mdata>
 <Tables name="Student">
 <Pk>
 <Attributes name="Scode">
 <Atable name="Student"/>
 </Attributes>
 <Atable name="Student"/>
 </Pk>
 <Fkeys>
 <Attribute name="Mcode">
 <Atable name="Student"/>
 </Attribute>
 <Atable name="Student"/>
 <Rtable name="Major"/>
 </Fkeys>
 <Attributes name="Sname">
 <Atable name="Student"/>
 </Attributes>
 <Attributes name="Saddresse">
 <Atable name="Student"/>
 </Attributes>
 <Attributes name="Sphone">
 <Atable name="Student"/>
 </Attributes>
 </Tables>
 <Tables name="Major">
 <Pk>
 <Attributes name="Mcode">
 <Atable name="Major"/>
 </Attributes>
 </Pk>
 <Attributes name="Mname">
 <Atable name="Major"/>
 </Attributes>
 </Tables>
</Mdata>
 <tuples>
 <values datavalue='1'/>
 <values datavalue='khalil'/>
 <values datavalue="beirut"/>
 <values datavalue="03999999"/>
 <values datavalue="1"/>
 <table name="Student"/>
 </tuples>
 <tuples>
 <values datavalue='2'/>
 <values datavalue='nabil'/>
 <values datavalue='saida'/>
 <values datavalue='03888888'/>
 <values datavalue='2'/>
 <table name='Student'/>
 </tuples>
 <tuples>
 <values datavalue='3'/>
 <values datavalue='talal'/>
 <values datavalue='tyr'/>
 <values datavalue='03777777'/>
 <values datavalue='3'/>
 <table name='Student'/>
 </tuples>
 <tuples>
 <values datavalue='1'/>
 <values datavalue='IAG'/>
 <table name='Major'/>
 </tuples>
 <tuples>
 <values datavalue='2'/>
 <values datavalue='GRIT'/>
 <table name='Major'/>
 </tuples>
 <tuples>
 <values datavalue='3'/>
 <values datavalue='GIM'/>
 <table name='Major'/>
 </tuples>
</Database>
```

d)

e) Generating a target model in xmi file

Once the source model is created in an xmi file, we execute the transformation process by running the atl program DB2KB.atl.

The output model (target) is generated in an xmi file SchoolKB.xmi that conforms to the KBMM meta-model.

SchoolKB.xmi

```
<KnowledgeBase name="DBTOKB">
<rules clause="Scode(A):-Student(A,_,_,_,_)">
 <head>
 <predicate name="Scode(A)">
 <terms>
 <Variable name="A"/>
 </terms>
 </predicate>
 </head>
 <body>
 <predicates name="Student(A,_,_,_,_)">
 <terms>
 <Variable name="A"/>
 </terms>
 <terms>
 <Variable name="B"/>
 </terms>
 <terms>
 <Variable name="C"/>
 </terms>
 <terms>
 <Variable name="D"/>
 </terms>
 <terms>
 <Variable name="E"/>
 </terms>
 </predicates>
 </body>
</rules>
<rules clause="Mcode(A):-Major(A,_)">
 <head>
 <predicate name="Mcode(A)">
 <terms>
 <Variable name="A"/>
 </terms>
 </predicate>
 </head>
 <body>
 <predicates name="Major(A,_)">
 <terms>
 <Variable name="A"/>
 </terms>
 <terms>
 <Variable name="B"/>
 </terms>
 </predicates>
 </body>
</rules>
<rules clause="Sname(A,B):-Student(A,_,_,B,_)">
 <head>
 <predicate name="Sname(A,B)">
 <terms>
 <Variable name="A"/>
 </terms>
 <terms>
 <Variable name="B"/>
 </terms>
 </predicate>
 </head>
<body>
 <predicates name="Student(A,_,_,B,_)">
 <terms>
 <Variable name="A"/>
 </terms>
 <terms>
 <Variable name="B"/>
 </terms>
 <terms>
 <Variable name="C"/>
 </terms>
 <terms>
 <Variable name="D"/>
 </terms>
 <terms>
 <Variable name="E"/>
 </terms>
 </predicates>
 </body>
</rules>
<rules clause="Saddresse(A,B):-Student(A,_,_,_,B)">
 <head>
 <predicate name="Saddresse(A,B)">
 <terms>
 <Variable name="A"/>
 </terms>
 <terms>
 <Variable name="B"/>
 </terms>
 </predicate>
 </head>
 <body>
 <predicates name="Student(A,_,_,_,B)">
 <terms>
 <Variable name="A"/>
 </terms>
 <terms>
 <Variable name="B"/>
 </terms>
 <terms>
 <Variable name="C"/>
 </terms>
 <terms>
 <Variable name="D"/>
 </terms>
 <terms>
 <Variable name="E"/>
 </terms>
 </predicates>
 </body>
</rules>
<rules clause="Sphone(A,B):-Student(A,_,B,_,_)">
 <head>
 <predicate name="Sphone(A,B)">
 <terms>
 <Variable name="A"/>
 </terms>
 <terms>
 <Variable name="B"/>
 </terms>
 </predicate>
 </head>
 <body>
 <predicates name="Student(A,_,B,_,_)">
 <terms>
 <Variable name="A"/>
 </terms>
 <terms>
 <Variable name="B"/>
 </terms>
 <terms>
 <Variable name="C"/>
 </terms>
 <terms>
 <Variable name="B"/>
 </terms>
 <terms>
 <Variable name="D"/>
 </terms>
 <terms>
```

```
<Variable name="B"/>
</terms>
<terms>
<Variable name="A"/>
</terms>
</predicate>
</head>
<body>
<predicates name="Student(A,B,_,_,_)">
<terms>
```

```
 <Variable name="E"/>
 </terms>
 </predicates>
 </body>
</rules>
<rules clause="Mcode(A,B):-
Student(A,B,_,_,_)">
 <head>
 <predicate
name="Mcode(A,B)">
 <terms>
 <Variable name="A"/>
 </terms>
 <terms>
 <Variable name="B"/>
 </terms>
 </predicate>
 </head>
 <body>
 <predicates
name="Student(A,B,_,_,_)">
 <terms>
 <Variable name="A"/>
 </terms>
 <terms>
 <Variable name="B"/>
 </terms>
 <terms>
 <Variable name="C"/>
 </terms>
 <terms>
 <Variable name="D"/>
 </terms>
 <terms>
 <Variable name="E"/>
 </terms>
 </predicates>
 </body>
</rules>
<rules clause="Mname(A,B):-
Major(A,B)">
 <head>
 <predicate
name="Mname(A,B)">
 <terms>
 <Variable name="A"/>
 </terms>
 <terms>
 <Variable name="B"/>
 </terms>
 </predicate>
 </head>
 <body>
 <predicates
name="Major(A,B)">
 <terms>
 <Variable name="A"/>
 </terms>
 <terms>
 <Variable name="B"/>
 </terms>
 </predicates>
 </body>
</rules>
<rules clause="Mcode(A,B):-
Stu-
dent(A,B,_,_,_),Major(C,_),B=
C">
 <head>
 <predicate
name="Mcode(A,B)">
 <terms>
 <Variable name="A"/>
 </terms>
 <terms>
 <Variable name="B"/>
 </terms>
 <terms>
 <Variable name="C"/>
 </terms>
 <terms>
 <Variable name="D"/>
 </terms>
 <terms>
 <Variable name="E"/>
 </terms>
 </predicates>
 <predicates name="Major(C,_)">
 <terms>
 <Variable name="C"/>
 </terms>
 <terms>
 <Variable name="D"/>
 </terms>
 </predicates>
 <predicates name="B=C">
 <terms>
 <Variable name="B"/>
 </terms>
 <terms>
 <Variable name="C"/>
 </terms>
 </predicates>
 </body>
</rules>
<facts
name="Student(1,khalil,beirut,039999
99,1)">
 <constants datavalue="1"/>
 <constants datavalue="khalil"/>
 <constants datavalue="beirut"/>
 <constants datavalue="03999999"/>
 <constants datavalue="1"/>
</facts>
<facts
name="Student(2,nabil,saida,0388888
8,2)">
 <constants datavalue="2"/>
 <constants datavalue="nabil"/>
 <constants datavalue="saida"/>
 <constants datavalue="03888888"/>
 <constants datavalue="2"/>
</facts>
<facts
name="Student(3,talal,tyr,03777777,3
)">
 <constants datavalue="3"/>
 <constants datavalue="talal"/>
 <constants datavalue="tyr"/>
 <constants datavalue="03777777"/>
 <constants datavalue="3"/>
</facts>
<facts name="Major(1,IAG)">
 <constants datavalue="1"/>
 <constants datavalue="IAG"/>
</facts>
<facts name="Major(2,GRIT)">
 <constants datavalue="2"/>
 <constants datavalue="GRIT"/>
</facts>
<facts name="Major(3,GIM)">
 <constants datavalue="3"/>
 <constants datavalue="GIM"/>
 </facts>
</KnowledgeBase>
```

Parsing the SchoolKB.xmi file provides the following text file that describes an instance of a knowledge base written as a set of rules and facts:

```
Scode(A):-Student(A,_,_,_,_)
Mcode(A):-Major(A,_)
Sname(A,B):-Student(A,_,_,B,_)
Saddresse(A,B):-Student(A,_,_,_,B)
Sphone(A,B):-Student(A,_,B,_,_)
Mcode(A,B):-Student(A,B,_,_,_)
Mname(A,B):-Major(A,B)
Mcode(A,B):-Student(A,B,_,_,_),Major(C,_),B=C
Student(1,khalil,beirut,03999999,1)
Student(2,nabil,saida,03888888,2)
Student(3,talal,tyr,03777777,3)
Major(1,IAG)
Major(2,GRIT)
Major(3,GIM)
```

5. Conclusion

In this paper, we have presented transformation rules capable of transforming a database instance into a knowledge base instance. Based on MDA architecture, our approach provides two meta-models (DBMM and KBMM) to conduct this transformation. A prototype transformation model is designed and implemented using ATL language. Data-base and knowledge base instances are described by xmi files. In the future, we plan to construct a tool for integrating the knowledge base xmi file automatically into a knowledge base of an expert system.

References

[1] Sagiroglu, S. and D. Sinanc, Big data: A review, 2013 International Conference on Collaboration Technologies and Systems (CTS), San Diego, CA, 2013, pp. 42-47, (2013). https://doi.org/10.1109/CTS.2013.6567202.

[2] Mukerji, J. and J. Miller, Technical Guide to Model Driven Architecture: The MDA Guide v1.0.1. Technical report (2003).

[3] Truyen, F., The Fast Guide to Model Driven Architecture, The Basics of Model Driven Architecture, WHITEPAPER, Cephas Consulting Corp. Architecture Oriented Services, January (2006).

[4] Bezivin, J. and O. Gerbé, Towards a precise definition of the omg/mda framework, ASE'01, Automated Software Engineering, San Diego, USA, Nov 26 – 29, (2001).

[5] Budinsky, F., D. Steinberg, R. Ellersick, T. Grose and E. Merks, Eclipse Modeling Framework: A Developer's Guide, Addison-Wesley Professional, (2004).

[6] Dinh, T. L. A., O. Gerbé and H. Sahraoui, Un méta-métamodèle pour la gestion de modèles. In IDM 06 Actes des 2èmes Journées sur l'Ingénierie Dirigée par les Modèles, Lille, France, (2006).

[7] Thi-Lan-anh, D., G. Olivier and S. Houari, Gestion de modèles : définitions, besoins et revue de littérature. In Premières Journées sur l'Ingénierie Dirigée par les Modèles, pages 1–15, Paris, France, 30 Juin- 1 Juillet (2005).

[8] Czarnecki, K. and S. Helsen, Feature-based survey of model transformation approaches. IBM Systems Journal, 45(3):621–645 (2006). https://doi.org/10.1147/sj.453.0621.

[9] QVT, Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Version 1.2, Needham, MA, formal/2015-02-01 edition, February (2015).

[10] Jouault, F., F. Allilaire, J. Bézivin and L. Kurtev, ATL: A model transformation tool. Science of Computer Programming, 72:31 – 39, Special Issue on Second issue of experimental software and toolkits (EST) (2008). https://doi.org/10.1016/j.scico.2007.08.002.

[11] Steinberg, D., F. Budinsky, M. Paternostro, E. Merks, R. Ellersick and T. J. Rose, EMF Eclipse Modeling Framework. The Eclipse Series. Boston, MA, 2nd edition (2009).

[12] Arenas M., A. Bertails, E. Prud'hommeaux, J. Sequeda, A Direct Mapping of Relational Data to RDF, W3C Recommendation 2012, http://www.w3.org/TR/rdb-direct-mapping, (2012).

[13] Pankowski, T., Using Data-to-Knowledge Exchange for Transforming Relational Databases to Knowledge Bases. In: Bikakis A., Giurca A. (eds) Rules on the Web: Research and Applications. RuleML 2012, LNCS,vol 7438, Springer, Berlin, Heidelberg, (2012). https://doi.org/10.1007/978-3-642-32689-9_21.

[14] Dadjoo, M. and E. Kheirkhah, An Approach for Transforming of Relational Databases to OWL Ontology, International Journal of Web & Semantic Technology (IJWesT) Vol.6, No.1, January (2015). https://doi.org/10.5121/ijwest.2015.6102.
[15] Gherabi, N., K. Addakiri and M. Bahaj, Mapping relational database into OWL Structure with data semantic Preservation, International Journal of Computer Science and Information Security, Vol. 10, No. 1, January (2012).
[16] Guoqiang, Z. and J. Suling, Ontology-based knowledge extraction for relational database schema, 2009 Second International Symposium on Electronic Commerce and Security (2009).
[17] Lei, C., Xu. Zhuoming and Ni. Lixian, WF2OML: A Modeling Language for Mapping Web Forms to Ontology, 2013 10th Web Information System and Application Conference, (2013).
[18] Jeong, C.-H., S.P. Choi, S.-H. Shin, S. Lee, H. Jung, S.-Y. Kim and P. Kim, Creating Semantic Data from Relational Database. Proceeding of IEEE International Conference on Social Computing, Washington, D.C, (2013). https://doi.org/10.1109/SocialCom.2013.174.
[19] Gui-hyun, B., K. Su-kyoung and A. Ki-hong, Framework for Automatically Construct Ontology Knowledge Base from Semi-structured Datasets, the 10th International Conference for Internet Technology and Secured Transactions (ICITST-2015).
[20] Ishan, R., S. Anurag and C. Adesh, DB2KB: A model to transform database to knowledgebase, 2013 International Conference on Information Systems and Computer Networks (ISCON), (2013).
[21] Strembeck, M. and U. Zdun, An approach for the systematic development of domain-specific languages, Software – Practice AND Experience, Vol. 39, Issue15, pp. 253-1292, October (2009). https://doi.org/10.1002/spe.936.
[22] ATLAS group. LINA and INRIA Nantes, ATL: Atlas Transformation Language, ATL User Manual, version 0.7, February (2006).
[23] JungHyen An and Young B. Park, Methodology for Automatic Ontology Generation Using Database Schema Information, Mobile Information Systems, vol. 2018, Article ID 1359174, 13 pages, (2018). https://doi.org/10.1155/2018/1359174.
[24] Ahn J.-H. and Park Y. B., Rule extraction ontology generation from an adaptive IoT ecosystem database, in Proceedings of the International Conference on ICT Convergence, Jeju Island, Korea, October (2017). https://doi.org/10.1109/ICTC.2017.8190808.