Use of Persistent Meta-Modeling Systems to Handle Mappings for Ontology Design

Valéry Téguiak
LIAS/ISAE-ENSMA
Futuroscope, France
teguiakh@ensma.fr

Yamine Ait-Ameur
IRIT-ENSEEIHT
Toulouse, France
yamine@enseeiht.fr

Éric Sardet
CRITT Informatique
Futuroscope, France
sardet@ensma.fr

***Abstract*—To enable data intensive application including global information systems with heterogeneous models, the model mapping problem in which a source model is mapped to a target one should be addressed. Current work about mapping provides a finite set of mapping constructors available for writing mappings. In this case, adding a new concept in a meta-model describing mapped schemas could have the effect of building new types of mapping constructors. Thus, this paper attempts to provide a generic and systematic approach for modeling mapping constructors, so that new mapping constructors could be handled efficiently without requiring to rebuild completely the mapping repository system.**

***Keywords*-data integration; mapping; meta-modeling; model transformation; ontology engineering.**

I. Introduction

The huge amount of data created by several application domains and development activities was at the origin of the emergence of several heterogeneous data models and modeling languages. The need of exploiting data and these models in an integrated manner led to several studies on data and model integration and heterogeneous modeling [1], [2]. Two particular and interesting studies are model mappings and mapping languages. Moreover, these approaches have also been developed in the context of the semantic web where model mappings were required for defining transformations, instance migration, etc.

In order to deal with various heterogeneous models used to represent the same real word domain, several mapping languages [8], [12], [20] have been proposed. Their central objective is to establish relationships between models. Most of these approaches run in central memory and do not address the scalability problem when dealing with huge amount of data, instances of those models.

However, many information systems rely on databases to ensure scalability. As a consequence, the need of managing mappings in a persistence context appeared. Therefore, the availability of a repository of mappings is required. Also, a way for exploiting mappings by interpreting, handling and manipulating such mapping operators is also required.

The work of Miller et al. [3] and Ling et al. [4] was precursor. It addressed the problem of mapping management in a database. The main assumption in this work consists in modeling mapping constructors as a finite set of operators. This assumption is acceptable if models to be mapped are encoded in a meta-model that will not evolve dynamically due to the fixed set of mapping constructors.

Nevertheless, offering the capability to manipulate and/or to modify the meta-model could offer more flexibility and extensions capabilities dynamically. Indeed, offering the capability for the meta-model to evolve by supporting the creation of new concepts would also offer the capability to dynamically define on the fly new mapping constructors. As a consequence, the definition of mapping constructors in a generic way becomes possible.

Moreover, because the size of models and instances are growing drastically, the traditional approaches for mapping models need to scale up. Therefore, offering persistent settings for managing such mappings and instances becomes a necessity if one wants to address real sized problems.

This paper focuses on the definition of a generic infrastructure for managing mappings in a database context. It uses a specific database architecture that supports definition of meta-models and their instances. This database infrastructure consists of: (1) a space for representing mapping constructors, (2) a space for representing models and mappings between these models and finally (3) a space for representing data (instances of models).

This paper is organized as follows. Section II outlines related work on mappings. Then, our contribution using constructive data models to model mappings is presented in Section III. In Section IV, we discuss how to represent a graph of mappings in a persistent context. Once our persistent solution for handling mappings in a database structure, through model repositories, is presented in Section V, we briefly present, in Section VI, how this approach has been set up to encode the transformation process for building ontologies starting from texts. This work has been conducted in the context of the DaFOE4App (Differential and Formal Ontologies Editor for Application) project [22]. We finally conclude and give some perspectives of this work.

II. Related Work

Many proposals address model mapping and data translation problems. These proposals can be splitted into two categories: hard encoded and rule-based approaches.

A. *Hard encoded approach*

By the term *hard encoded*, we refer to approaches where both mappings and mapped models representations are hidden within a framework as a program. It means that these representations are not exposed as declarative and user-comprehensible rules. This leads to several difficulties. First, models and mappings can only be extended by the framework designers. Secondly, because of the program-based representation of models and mappings, any extension requires changes at the framework code level. As a consequence, correctness of these representations has to be accepted by users as a dogma. For example, the approach of Papotti and Torlone [14] can be said to be hard encoded. In that context, the expressed transformations are imperative programs, which have the weaknesses described above. The instance translation process is achieved by firstly converting the source data into XML, and then by performing an XML-to-XML translation expressed in XQuery to reshape instances in order to be compatible with the target schema, and finally, by converting the XML representation into the target model.

B. *Rule-based approach*

Weaknesses of the hard encoded approach can be solved using a rule-based approach. This approach attempts to provide a generic way to handle models, mappings and data translation without using a hard encoded program. For example, the approach proposed by Bernstein et al. [13] is a rule-based one. In that approach, they focus on a flexible mapping based on inheritance hierarchies, and in the incremental regeneration of mappings each time the source schema is modified. Other rule-based approaches are driven by a dictionary of schemas, models and translation rules. Among them, we can quote the work of Bowers and Delcambre [11] that proposes Uni-Level Description (UDL). UDL is a meta-model in which models and translations can be described and managed in a uniform process environment for models, schemas and instances. UDL is used to express specific model-to-model translations of both schemas and instances. Like the approach of Atzeni et al. [16], translations are expressed as Datalog rules and the source and target models are stored in a generic relational dictionary.

Our approach is also considered as a rule-based approach. But, compared to the previous quoted approaches, we provide a more abstract level where, in addition, the dictionary is explicitly represented and becomes manageable. Indeed, the dictionary representation according to a meta-meta-model allows the user for example, to modify mapping models without modifying the underlying program.

Furthermore, Kalfoglou and Schorlemmer [7] address the problem of mapping discovery which consists of an automatic synthesis of an alignment between models. In our proposal, we assume that the discovery process has already been achieved. Indeed, our work deals with mapping specification and instance mediation in database environment. More discussions on topics around mapping problems and provided solutions can be founded in [4], [9], [15], [18]. As illustrated in Figure 1, mappings can be composed transitively. This requirement has been formalized in [10], [19], where an approach to use composition among models has been proposed. Because this paper focuses on a repository for storing mappings, we do not discuss handling composition between mappings (composition is handled by a query engine in our framework). Furthermore, [3], [4] introduce the notion of *value correspondence* as a proposal of representation for mapping operators.

III. Our approach

In our approach, modeling mapping consists in creating mapping constructors (Model level mapping, Entity level mapping, Attribute level mapping, etc.). In this section, we present a formal model for mapping constructors. Furthermore, before connecting domain models using mappings, these models should be represented in a way allowing them to be managed efficiently. The meta-modeling-based approach is often used for this purpose. We use a meta-model called Entity-Attribute meta-model (E-A meta-model) to handle domain models. Using this E-A meta-model, a model *m* is formally defined by $m = \langle E, A, I, T, dom, range, its\_entity \rangle$ where:

- ***E*** represents the set of entities of the model *m*;
- ***A*** represents the set attributes used to describe entities of the model *m*;
- ***I*** represents the set of entity instances of the model *m*;
- ***T*** is a set of primitive types (Int, String, Boolean, etc.);
- $dom : A \to E$ defines the domain of an attribute;
- $range : A \to E \cup T$ defines the range of an attribute;
- $its\_entity : I \to E$ returns the entity associated to a given instance.

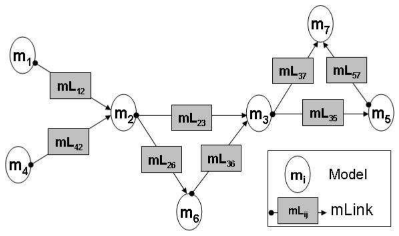


Figure 1. An mLink graph of model mappings.

A. *Model level mapping: mLink*

Correspondences between models are represented by a directed acyclic graph whose nodes are models. Formally,

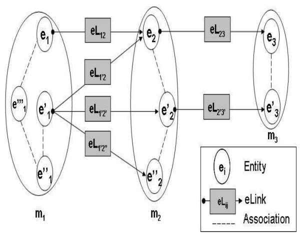


Figure 2. eLink graph.

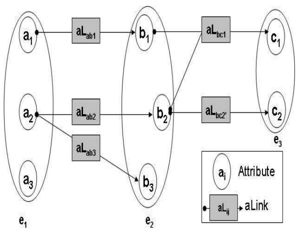


Figure 3. aLink graph.

if ***M*** represents a set of models, the graph G_m of correspondences between models (Cf. Figure 1) is defined by $G_m = (N_m, L_m)$ where:

- $N_m \subseteq$ ***M*** represents the set of nodes of the correspondence graph;
- $L_m = \{(m_s, m_t, \alpha_m) \in N_m \times N_m \times [0; 1]\}$ represents a set of correspondences between a source model (m_s) and a target model (m_t). Here, α_m is a confidence degree of this correspondence.

Through this paper, we will use the term ***mLink***, formally defined as element of L_m, to refer to a correspondence or a mapping between models.

B. Entity level mapping: eLink

Correspondences between entities of the models are represented by a directed acyclic graph whose nodes are entities. Formally, if ***E*** represents a set of entities, the graph G_e of correspondences between entities (Cf. Figure 2) is defined by $G_e = (N_e, L_e)$ where:

- $N_e \subseteq$ ***E*** is the set of nodes of the correspondence graph;
- $L_e = \{(e_s, e_t, \alpha_e, mL) \in N_e \times N_e \times [0; 1] \times L_m\}$ represents correspondences between a source entity (e_s) and a target entity (e_t) built in the context of the *mLink mL* with confidence degree α_e.

Through this paper, we will use the term ***eLink***, formally defined as element of L_e, to refer to a correspondence between entities.

C. Attribute level mapping: aLink

The arity of correspondences between attributes is n:0 reflecting the fact that one needs zero or more attributes of the source entity to compute an attribute of the target entity. Formally, if ***A*** represents the set of attributes, the graph G_a of correspondences between attributes (Cf. Figure 3) is defined by $G_a = (N_a, L_a)$ where:

- $N_a \subseteq$ ***A*** represents the set of nodes of the correspondence graph;
- $L_a = \left\{(A_s, a_t, \alpha_a, \varphi, eL) \in N_a^k \times N_a \times [0; 1] \times \Phi \times L_e\right\}$ represents a set of correspondences from a set of sources attributes $A_s = (a_{s1}, a_{s2}, ..., a_{sk})$ to a target one ($a_t \in N_a$) where:
  - the eLink *eL* represents the context in which the correspondence has been created;
  - φ is an expression used to write a_t in terms of a_{si} ($1 \leq i \leq k$).
  - α_a represents the confidence degree of the correspondence.

Through this paper, we will use the term ***aLink***, formally defined as element of L_a, to refer to a correspondence between attributes.

Unlike other work performed on mappings [3], [4], [12], our approach does not presuppose the existence of a finite set of mapping constructors but it aims at providing a formal support to dynamically create new mapping constructors or evolving existing one. Indeed, all mapping constructors presented above use a numeric confidence degree α. This is the most used approach for handling fuzzy mappings. However, what will happen if a user (who is not the framework designer) want to create another fuzzy property for mapping by annotating each mapping with a quality value (“Weak”, “Average”, “Good”, “Very”, “Good”, “Excellent”, etc. for example)? Thus, our approach propose to model mapping constructors so that mapping constructors like *mLink*, *eLink*, *aLink*, etc., could be managed in a generic way and easily extended at runtime. As our work is conducted in a database context, we discuss in the following sections, modeling possibilities of a database repository to store such a graph-based representation of mapping.

IV. Model repositories

In the recent years, several works [6], [17] investigated the problem of representing ontologies and their instances within a database. We reuse this approach for models in general and the resulting database (that we simply call Model Based-Database (MBDB)) can be represented according in three main approaches. In this section, we present a taxonomy of these approaches. Our goal here is to discuss how the graph of mappings presented in Section III can be stored in each of these database types.

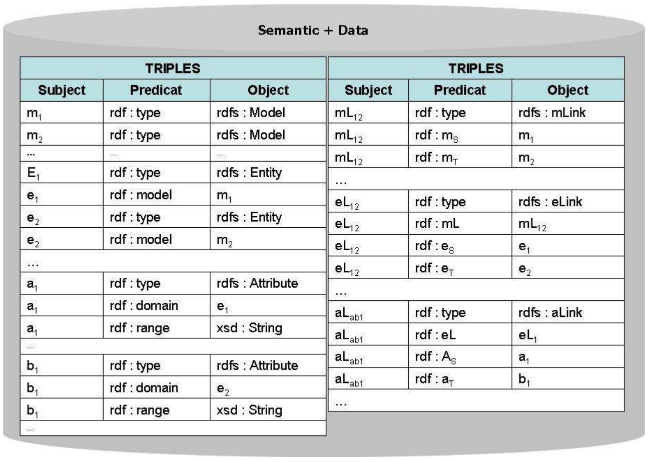
A. MBDB of type 1

In MBDB of type 1, information is represented using a single schema composed of a single table of triples (subject, predicate, object). This table, referred as vertical table [5] is used both for model level data and instance level data. For model level data, the three columns of this table respectively represent the Identifier of an element of the model, a predicate and the value taken by the predicate (Cf. Figure 4). Furthermore, in order to implement our mapping representation proposal with MBDB of type 1, we apply the following rules:

- use RDF Schema as the meta-model for representing domain models. For instance, the triplet (e_1, Type, Entity) means that the concept e_1 is an RDF Class (called Entity in the E-A meta-model);

- extend RDF Schema with new concepts representing mapping constructors (*mLink*, *eLink*, *aLink*);
- extend RDF Schema with new properties representing parameters of L_m, L_e, L_a. For instance, "$rdf : m_s$" is used to represent the source model of a *mLink* while "$rdf : eL$" is used to identify a *mLink* representing the context in which a given *eLink* is created.

Putting all these previous rules together results in a database as illustrated in Figure 4. Unfortunately, it clearly appears that, this approach is not enough scalable because of the number of auto-join operations required on the underlying vertical table.



TRIPLES		
Subject	**Predicat**	**Object**
m_1	rdf : type	rdfs : Model
m_2	rdf : type	rdfs : Model
...	...	...
E_1	rdf : type	rdfs : Entity
e_1	rdf : model	m_1
e_2	rdf : type	rdfs : Entity
e_2	rdf : model	m_2
...		
a_1	rdf : type	rdfs : Attribute
a_1	rdf : domain	e_1
a_1	rdf : range	xsd : String
...		
b_1	rdf : type	rdfs : Attribute
b_1	rdf : domain	e_2
b_1	rdf : range	xsd : String
...		

TRIPLES		
Subject	**Predicat**	**Object**
mL_{12}	rdf : type	rdfs : mLink
mL_{12}	rdf : m_S	m_1
mL_{12}	rdf : m_T	m_2
...		
eL_{12}	rdf : type	rdfs : eLink
eL_{12}	rdf : mL	mL_{12}
eL_{12}	rdf : e_S	e_1
eL_{12}	rdf : e_T	e_2
...		
aL_{ab1}	rdf : type	rdfs : aLink
aL_{ab1}	rdf : eL	eL_1
aL_{ab1}	rdf : A_S	a_1
aL_{ab1}	rdf : a_T	b_1
...		

Figure 4. MBDB of type 1.

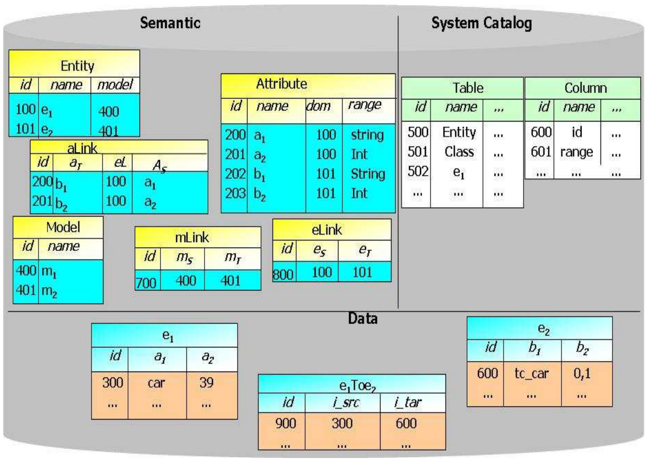
B. *MBDB of type 2*

MBDB of type 2 store separately model level data and the instance level data in two distinct schemes [6]. In classical databases, the system catalog part plays the role of the model level where models are stored as meta-data. The main problem with this representation comes from the fact that the meta-model provided by the DataBase Management System (DBMS) cannot be manipulated. Indeed, this meta-model is frozen and therefore, it cannot be evolved according to some particular requirements. For example, how do we represent mapping concepts like *mLink*, *eLink*, *aLink*, *etc.* (not available in this meta-model) using this type of database? However, as illustrated in Figure 5, one can use the semantic part of the database to represent mappings. Even if this approach provides a solution independent of a particular DBMS, the meta-model of the semantic part is also frozen and cannot be extended at runtime in order to provide new mapping constructors.

C. *MBDB of type 3*

MBDB of type 3 [17] propose to add another schema to MBDB of type 2. This schema stores the E-A meta-model in a reflexive meta-meta-model. Thus, for the meta-model, the meta-meta-model plays the same role as the one played by the system catalog. Compared to MBDB of type 2, adding a meta-meta-model in MBDB of type 3 provides possibility to extend the meta-model. So, thanks to that meta-meta-model, MBDB of type 3 could be reused to reach our goal related to the representation of mapping concepts in database. Figure 6 illustrates the OntoBD [17] architecture as an example of a MBDB of type 3. OntoDB is based on 4 main parts:

- **the meta-base part:** it corresponds to the catalog system of databases. It contains system tables used to manage all the data contained in the database;
- **the data part:** it represents domain objects also called data;
- **the semantic part:** it contains models defining the semantics of data. More precisely, domain models of information system stored in the semantic part;
- **the meta-schema part:** the meta-schema part records the E-A meta-model.



Entity		
id	*name*	*model*
100	e_1	400
101	e_2	401

aLink			
id	a_T	*eL*	A_S
200	b_1	100	a_1
201	b_2	100	a_2

Attribute			
id	*name*	*dom*	*range*
200	a_1	100	string
201	a_2	100	Int
202	b_1	101	String
203	b_2	101	Int

Model	
id	*name*
400	m_1
401	m_2

mLink		
id	m_S	m_T
700	400	401

eLink		
id	e_S	e_T
800	100	101

Table		
id	*name*	...
500	Entity	...
501	Class	...
502	e_1	...
...	...	...

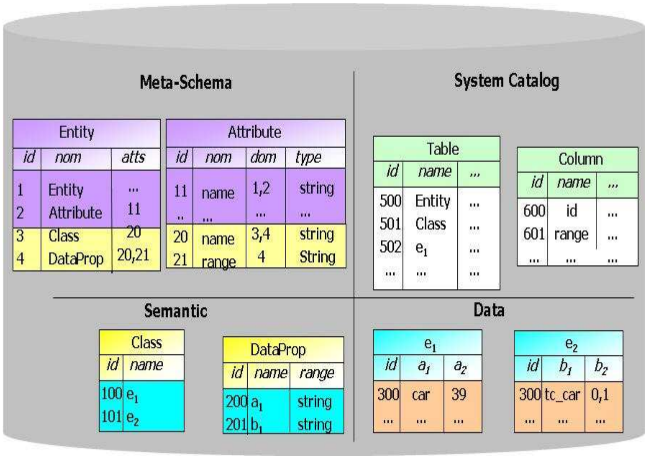
Column		
id	*name*	...
600	id	...
601	range	...
...	...	...

e_1		
id	a_1	a_2
300	car	39
...	...	...

e_1Toe_2		
id	*i_src*	*i_tar*
900	300	600
...	...	...

e_2		
id	b_1	b_2
600	tc_car	0,1
...	...	...

Figure 5. MBDB of type 2.



Entity		
id	*nom*	*atts*
1	Entity	...
2	Attribute	11
3	Class	20
4	DataProp	20,21

Attribute			
id	*nom*	*dom*	*type*
11	name	1,2	string
..	...	...	...
20	name	3,4	string
21	range	4	String

Table		
id	*name*	...
500	Entity	...
501	Class	...
502	e_1	...
...	...	...

Column		
id	*name*	...
600	id	...
601	range	...
...	...	...

Class	
id	*name*
100	e_1
101	e_2

DataProp		
id	*name*	*range*
200	a_1	string
201	b_1	string

e_1		
id	a_1	a_2
300	car	39
...	...	...

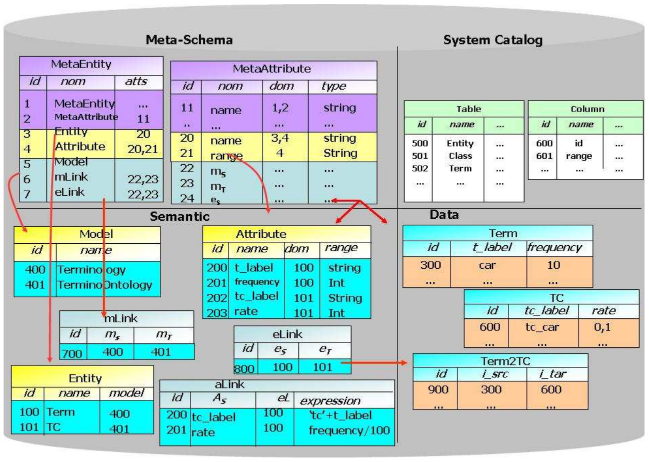
e_2		
id	b_1	b_2
300	tc_car	0,1
...	...	...

Figure 6. OntoDB: a MBDB of type 3.

The next Section presents how the approach led by MBDB of type 3 has been used to persist the graph of mappings.

V. HANDLING MAPPINGS IN A DATABASE

In this section, we give some details about the implementation of our proposal. Thanks to its extension facility, MBDB of type 3 are therefore better suited to implement the mapping approach presented in this paper. Indeed, a type 3 MBDB provides enough flexibility for the extension of both the E-A meta-model and the mapping model. It is therefore possible, for example, to create new mapping constructors. The resulting database repository is illustrated in Figure 7 where labels purposes (instead of object identifier) are used only for more readability. Compared to Figure 6, we have extended the semantic part with mapping constructors. All these mapping constructors are created as instances of the meta-schema. That provides for creating dynamically new mapping constructors. The resulting infrastructure is obtained in 3 steps.



MetaEntity		
id	*nom*	*atts*
1	MetaEntity	...
2	MetaAttribute	11
3	Entity	20
4	Attribute	20,21
5	Model	
6	mLink	22,23
7	eLink	22,23

MetaAttribute			
id	*nom*	*dom*	*type*
11	name	1,2	string
..	...	...	...
20	name	3,4	string
21	range	4	String
22	m_S	...	...
23	m_T	...	...
24	e_S	...	...

Table		
id	*name*	*...*
500	Entity	...
501	Class	...
502	Term	...
...	...	...

Column		
id	*name*	*...*
600	id	...
601	range	...
...	...	...

Model	
id	*name*
400	Terminology
401	TerminoOntology

Attribute			
id	*name*	*dom*	*range*
200	t_label	100	string
201	frequency	100	Int
202	tc_label	101	String
203	rate	101	Int

mLink		
id	m_s	m_T
700	400	401

eLink		
id	e_S	e_T
800	100	101

Entity		
id	*name*	*model*
100	Term	400
101	TC	401

aLink			
id	A_S	*eL*	*expression*
200	tc_label	100	'tc'+t_label
201	rate	100	frequency/100

Term		
id	*t_label*	*frequency*
300	car	10
...	...	...

TC		
id	*tc_label*	*rate*
600	tc_car	0,1
...	...	...

Term2TC		
id	*i_src*	*i_tar*
900	300	600
...	...	...

Figure 7. MBDB with mappings.

1) **Setup of the mapping management infrastructure.** This infrastructure consists in building a repository for mapping constructors. After creating tables in the meta-schema part, we populate them. For example, the statements of Table I insert new rows in tables *MetaEntity* and *MetaAttribute* tables of Figure 7. As a consequence, a physical repository for each constructor is automatically built in the semantic part. A table is created for each meta-entity with its attribute found in the meta-attribute table. At this level, we use classical SQL queries and the designer needs to know the meta-schema tables structure. The MQL language [21] provides the user with high level operators that hide implementation details available in such SQL queries. We do not give details of this language to keep this paper in reasonable size. Notice that, for readability purposes all statement examples provided

Table I
CREATION OF THE E-A META-MODEL

INSERT INTO MetaEntity (label) VALUES(“mLink”)
INSERT INTO MetaEntity (label) VALUES(“eLink”)
INSERT INTO MetaEntity (label) VALUES(“aLink”)
INSERT INTO MetaEntity (label) VALUES(“Model”)
...
INSERT INTO MetaAttribute (label, domain, type) VALUES(“m_s”,”mLink”,”Model”)
...
INSERT INTO MetaAttribute (label, domain, type) VALUES(“m_t”,”mLink”,”Model”)
...
INSERT INTO MetaAttribute (label, domain, type) VALUES(“α”,”mLink”,INT)
...

in this paper use *‘‘label”* as foreign key instead of URI.

2) **Model creation.** The models creation task consists in populating the *Entity*, *Attribute* and *Model* tables of the E-A meta-model (Cf. Figure 7). For each entity of the E-A meta-model, the physical structure (e_i tables) for storing data is created in the Data part.

3) **Mapping creation.** Creating a mapping consists in populating the *mLink*, *eLink*, *aLink* tables materializing mapping constructors (Cf. Figure 7). This explicit representation keeps traceability of mappings. However, it also keeps traceability between instances. Indeed, instance level mappings results from the instantiation of *eLink*. This instance mapping constructor is handled by creating, for each *eLink* a table ($e_1 TO e_2$ for example) storing those instances of the source entity that have been used to build the instance of the target entity.

An example of both of model and mapping creation tasks is given in Section VI, where the case study of modeling an ontology building process is detailed.

Notice that we have defined directed links from models and concepts to others. The reverse links can be easily built, if needed, using the same process. This capability will offer full traversals in the database from models to others. As a consequence, it is possible to trace the source concept used to produce target ones. The MQL language [21] offers high level operators for such traversals.

VI. APPLICATION

This section summarizes the use of the approach developed above in a particular context: building ontologies starting from text analysis.

A. Overview

The proposal of this paper has been applied in the DAFOE platform, a platform led by the ANR DaFOE4App project. This platform provides a stepwise methodology where the first step is dedicated to linguistic analysis (called Terminology Step) in which users manage linguistic information (terms and relations between terms) extracted with natural language processing tools (NLP). After the linguistic analysis, the step for structuring linguistic information (called TerminoOntology step), in order to avoid possible ambiguity of terms, is performed. Finally, a formalization step (called Ontology Step) allows users to create *classes* and *properties* of the ontologies and to populate created classes. Each of these autonomous steps has been modeled as illustrated in Figure 8, 9 and 10 respectively. Notice that the main goal of the DaFOE platform is not to populate classes but to build ontologies that are intended to be exported into other systems that provide instance management facilities. Thus, instance management is out of the scope of the application domain.

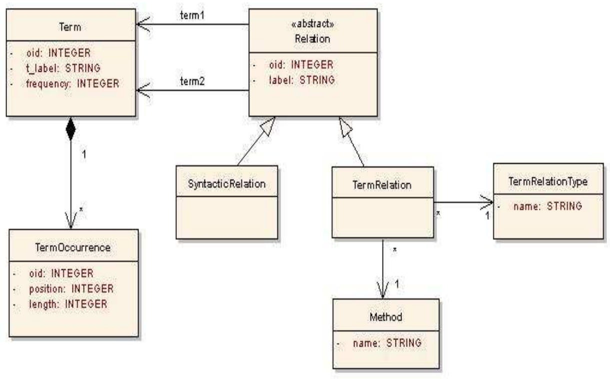


Figure 8. A simplified representation of the Terminology model.

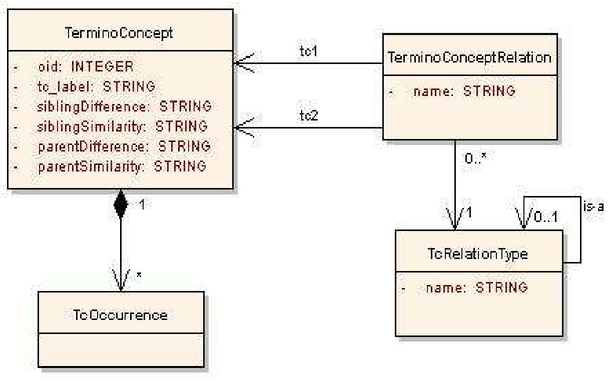


Figure 9. A simplified representation of the TerminoOntology model.

B. Setting up our approach

Applying our approach leads to the persistent infrastructure represented in Figure 11. It consists in writing correspondences between elements of the model of each step using mapping constructors. The developed approach in the

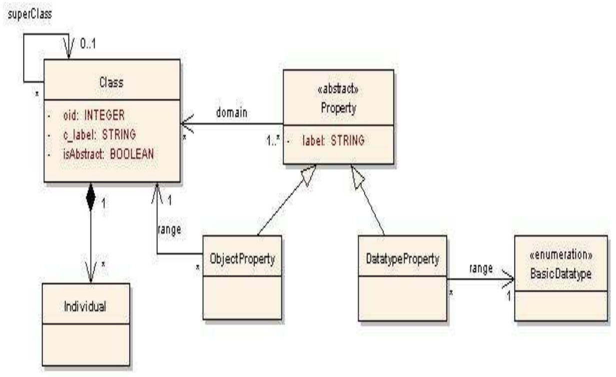


Figure 10. A simplified representation of the Ontology model.

DaFOE4App project has identified two bridges for switching between steps: a first one for producing termino-ontology concepts from texts and a second one for producing ontology concepts from termino-ontology concepts. According to our approach, bridge means the creation of *mLink, eLink* and *aLink* respectively after setting the models to be mapped. These two bridges are detailed below.

1) **Bridge 1. Terminology to TerminoOntology step.** Considering both *Terminology* and *TerminoOntology* steps through their models (Cf. Figure 8 and 9 respectively), a simplified mapping between these steps consists in:
mLink creation. The statement S_1 of Table II creates a *mLink* from the *Terminology* model to the *TerminoOntology* model. As a result, row 700 is inserted in table *mLink* (Figure 11).
eLink creation. The statement S_3 of Table III creates a *eLink* from the *Term* entity to the *TerminoConcept* entity to express that instances of the *Term* entity will be transformed as instances of the *TerminoConcept* entity. As a result of this statement, row 800 is inserted in the table eLink as illustrated in Figure 11. A *eLink* from the *TermRelation* entity of the *Terminology* model and the *TerminoConceptRelation* entity of the *TerminoOntology* model is also available.
aLink creation. The statement S_5 of table IV shows a *aLink* expresing that an instance of the *TerminoConcept* entity has the same *label* as its corresponding instance in the *Term* entity prefixed by *'tc_'*. Another *aLink* expresses that the rate of an instance of *TerminoConcept* entity equals to the frequency of corresponding instances in *Term* entity divided by 100. As a result, rows 200 and 201 are inserted in table *aLink* (Figure 11).
2) **Bridge 2. TerminoOntology to Ontology step.** For *TerminoOntology* and *Ontology* steps, a simplified mapping between their respective models consists in:
mLink creation. The statement S_2 of Table II creates a *mLink* from the *TerminoOntology* model to the *Ontology* model. As a result, row 701 is inserted in

Table II
STATEMENTS FOR MLINKS CREATION

Statement S_1–
INSERT INTO mLink (label, m_s, m_t, α_m)
VALUES(“Terminology2TerminoOntology”, “Terminology”, “TerminoOntology”, 0.8);

Statement S_2–
INSERT INTO mLink (label, m_s, m_t, α_m)
VALUES(“TerminoOntology2Ontology”, “TerminoOntology”, “Ontology”, 0.9);
...

Table III
STATEMENTS FOR ELINKS CREATION

Statement S_3–
INSERT INTO eLink (label, e_s, e_t, α_e, mL)
VALUES(“Term2TerminoConcept”, “Term”, “TerminoConcept”, 0.8, Terminology2TerminoOntology);

Statement S_4–
INSERT INTO eLink (label, e_s, e_t, α_e, mL)
VALUES(“TerminoConcept2Class”, “TerminoConcept”, “Class”, 0.8, “TerminoOntology2Ontology”);
...

table *mLink* of Figure 11.

eLink creation. In the context of the previous created *mLink*, a *eLink* is created from the *TerminoConcept* entity to the *Class* entity to express that instances of the *TerminoConcept* entity will be transformed as instances of the *Class* entity. This *eLink* is created using statement S_4 represented Table III. A *eLink* from *TerminoConceptRelation* of the *TerminoOntology* model and the *Property* entity of the Ontology model is also available. As a result, row 801 is inserted in table *eLink* (Figure 11).

aLink creation. The statement S_7 of Table IV shows a *aLink* expressing that an instance of the *Class* entity has the same *label* as its corresponding instance in *TerminoConcept* entity. As result, rows 202 is inserted in table *aLink* (Figure 11).

Putting these mappings all together results in a stepwise design methodology for a database recording manipulated data and produced to build an ontology from texts according to the process defined in the DaFOE4App project.

VII. CONCLUSION

In this paper, we presented an approach for persisting mappings. Focusing on a specific type of databases i.e persistent meta-modeling systems, we proposed an extensible infrastructure for mapping management. Rather than freezing all the mapping constructors in a database, we have proposed to represent them as a model. This model is then used on the one hand to create new mapping constructors and on the other hand for the automatic generation of a persistent

Table IV
STATEMENTS FOR ALINKS CREATION

Statement S_5–
INSERT INTO aLink (label, A_s, a_t, α_a, φ, eL)
VALUES(“TermLabel2TcLabel”, (“term_label”), “tc_label”, 0.8, “tc_label= “tc_” + term_label”, “Term2TerminoConcept”);

Statement S_6–
INSERT INTO aLink (label, A_s, a_t, α_a, φ, eL)
VALUES(“TermLabel2TcLabel”, (“frequency”), “rate”, 0.8, “rate= “frequency/100”, “Term2TerminoConcept”);

Statement S_7–
INSERT INTO aLink (label, A_s, a_t, α_a, φ, eL)
VALUES(“TcLabel2ClassLabel”, (“tc_label”), “tc_label”, 0.8, “class_label= tc_label”, “TerminConcept2Class”);
...

repository for mappings to ensure their traceability. As an assessment, our approach has been deployed and then implemented for the modeling process of building ontologies from texts in the context of the DaFOE4App project.

Furthermore, once models and mappings are created and models are populated with data, it would be interesting for example, to exploit these mappings when querying data. Indeed, because our mapping modeling is applied to models that represent the same real world domain, the domain related retrieving process needs to interpret mappings between models. Unfortunately, the resulting mapping graph as presented in this paper may be complex to manage with the classical SQL queries. For instance, as mappings are transitive thanks to mapping composition, one would want to use this capability to retrieve data transitively. Writing such a query could become complex. Thus, in continuity to this work, we have defined a SQL-like management and query language, namely MQL [21], that provides high level operators that makes easier querying data using mappings between models. This language, that hides implementation details regarding the database structure will be benchmarked in the context of engineering data retrieving where response time may be critical because of the huge amount of the underlying data. In this particular case, we are planing to improve performance analysis of the MQL query language.

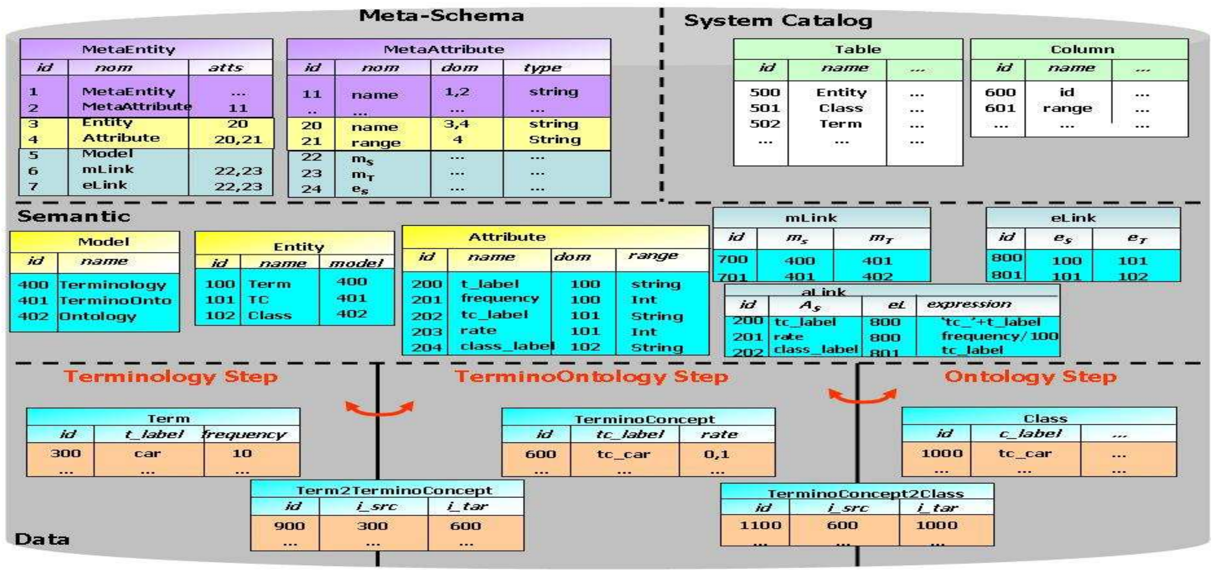
ACKNOWLEDGMENT

The authors would like to thanks the partners of the ANR DaFOE4App project for their contribution.

REFERENCES

[1] Michael Genesereth, Arthur Keller and Oliver Duschka. Infomaster: an information integration system. In Proceedings of the ACM SIGMOD Record, pp. 539–542, 1997

[2] Pepijn Visser, Martin Beer, Trevor Bench-Capon, B. M. Diaz and Michael Shave. Resolving Ontological Heterogeneity in the KRAFT Project. In Proceedings of DEXA’99, pp. 668–677, 1999



MetaEntity

id	nom	atts
1	MetaEntity	...
2	MetaAttribute	11
3	Entity	20
4	Attribute	20,21
5	Model	
6	mLink	22,23
7	eLink	22,23

MetaAttribute

id	nom	dom	type
11	name	1,2	string
..	...	...	...
20	name	3,4	string
21	range	4	String
22	m_S	...	...
23	m_T	...	...
24	e_S	...	...

Table

id	name	...
500	Entity	...
501	Class	...
502	Term	...
...	...	...

Column

id	name	...
600	id	...
601	range	...
...	...	...

Model

id	name
400	Terminology
401	TerminoOnto
402	Ontology

Entity

id	name	model
100	Term	400
101	TC	401
102	Class	402

Attribute

id	name	dom	range
200	t_label	100	string
201	frequency	100	Int
202	tc_label	101	String
203	rate	101	Int
204	class_label	102	String

mLink

id	m_S	m_T
700	400	401
701	401	402

eLink

id	e_S	e_T
800	100	101
801	101	102

aLink

id	A_S	eL	expression
200	tc_label	800	'tc_'+t_label
201	rate	800	frequency/100
202	class_label	801	tc_label

Term

id	t_label	frequency
300	car	10
...	...	...

TerminoConcept

id	tc_label	rate
600	tc_car	0,1
...	...	...

Class

id	c_label	...
1000	tc_car	...
...	...	...

Term2TerminoConcept

id	i_src	i_tar
900	300	600
...	...	...

TerminoConcept2Class

id	i_src	i_tar
1100	600	1000
...	...	...

Figure 11. Mapping management in the DaFOEApp project.

[3] Renée Miller, Laura Haas and Mauricio Hernández. Schema Mapping as Query Discovery. In Proceedings of VLDB'00, pp. 77–88, 2000

[4] Yan Ling, Miller Renée, Haas Laura and Fagin Ronald. Data-driven understanding and refinement of schema mappings. In SIGMOD Record, pp. 485–496, 2001

[5] Rakesh Agrawal, Amit Somani and Yirong Xu. Storage and Querying of E-Commerce Data. In Proceedings of VLDB'01, pp. 149–158. Rome, Italy 2001

[6] Jeen Broekstra, Arjohn Kampman and Frank van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In Proceedings of ISWC'02, pp. 54-68, 2002

[7] Yannis Kalfoglou and Marco Schorlemmer. IF-Map: an ontology mapping method based on information flow theory. In JoDS'03, pp. 98–127, 2003

[8] Sergey Melnik, Erhard Rahm and Philip Bernstein. Developing metadata-intensive applications with Rondo. In Journal of Semantic Web, pp. 47–74, 2003

[9] Philip Bernstein. Applying Model Management to Classical Meta Data Problems. In SIGMOD Record, pp. 209-220. 2003

[10] Madhavan Jayant and Halevy Alon. Composing mappings among data sources. In Proceedings of VLDB'03, pp. 572–583, 2003

[11] S. Bowers and L. M. L. Delcambre. The Uni-Level Description: A uniform framework for representing information in multiple data models. In Proceedings of ER'03, pp. 45-58, 2003.

[12] Ian Horrocks, Peter Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof and Mike Dean. SWRL: a semantic web rule language combining OWL and RuleML, 2004, http://www.w3.org/Submission/SWRL/,2012-04-09 06:00:05 +0100

[13] Philip Bernstein, Sergey Melnik and Peter Mork. Interactive schema translation with instance-level mappings, In Proceedings of VLDB'05, pp. 1283–1286, 2005

[14] P. Papotti, R. Torlone. Heterogeneous data translation through XML conversion. J. Web Eng., pp. 189-204, 2005.

[15] Choi Namyoun, Song Il-Yeol and Han Hyoil. A survey on ontology mapping. In SIGMOD Record, pp. 34–41. New York, USA 2006

[16] Paolo Atzeni, Paolo Cappellari and Philip A. Bernstein. MIDST: model independent schema and data translation, In SIGMOD Record, pp. 1134-1136, 2007

[17] Hondjack Dehainsala, Guy Pierra and Ladjel Bellatreche. OntoDB: An ontology-based database for intensive applications. In Proceedings of DASFAA'07, pp. 497–506, 2007

[18] Jérôme Euzenat, Pavel Shvaiko. Ontology matching. In Springer-Verlag(eds.), 2007

[19] Alan Nash, Philip Bernstein and Sergey Melnik. Composition of Mappings Given by Embedded Dependencies. In ACM TODS'07, Volume 32 Issue 1, pp. 172–183, 2007

[20] Naouel Moha, Sagar Sen, Cyril Faucher, Olivier Barais and Jean-Marc Jézéquel: Evaluation of Kermeta for solving graph-based problems. In STTT'10 Journal, pp. 273–285, 2010

[21] Valéry Téguiak, Yamine Ait-Ameur, Éric Sardet and Ladjel Bellatreche. MQL: an extension of SQL for mappings manipulation. Internal report, LIAS/ISAE-ENSMA, 2011, http://www.lisi.ensma.fr/ftp/pub/documents/papers/2011/2011-Report-Teguiak_1.pdf, 2012-04-09 06:00:05 +0100

[22] Projet DaFOE4App. État de l'art et étude des besoins pour une plate-forme de construction d'ontologies. Livrable de projet, 2007, ftp://ftp.irit.fr/IRIT/IC3/Dafoe-livrableA.0.1.pdf, 2012-04-09 06:00:05 +0100