INCREMENTAL DESIGN OF ONTOLOGIES
A Model Transformation-based Approach

Henry Valéry Téguiak[1,2], Yamine Ait-Ameur[1], Stéphane Jean[1] and Eric Sardet[2]
[1]*LISI/ENSMA and University of Poitiers, Poitiers, France*
[2]*CRITT Informatique CRCFAO, Chasseneuil, France*

Keywords: Mapping, Meta-modeling, Model transformation, Ontology, Terminology.

Abstract: This paper focuses on Model Driven Engineering (MDE) techniques to build ontology incrementally. The global building process consists of building ontologies from texts through a stepwise approach. Our approach consists of modeling each step independently and defining a mapping model to express relationships between these steps. Results of this work are applied in the ANR DAFOE project, which aims to propose a platform for building ontologies from several kind of resources (texts, terminologies, thesauri or existing ontologies).

1 INTRODUCTION

Although text-based ontology engineering gained much popularity in the last 10 years, very few ontology engineering platforms exploit the full potential of the connection between texts and ontologies. The ontology designers use ontology editors to design their ontologies. Unfortunately, the process leading to this ontology is not recorded and therefore, it is not maintained. As a consequence, traceability is lost.

Thus, we present DAFOE[1], a new platform for building ontologies using different types of linguistic entries (text corpora, results of natural language processing tools, terminologies or thesauri). DAFOE supports knowledge structuring and conceptual modeling from these linguistic entries as well as ontology formalization. The requirements of the platform and its development focus on (1) integrating various kinds of tools usually used within a single modeling platform, (2) guaranteeing persistence and traceability of the whole building process, and (3) developing the platform in an open source paradigm with possible plug-in extensions. Taking into account the diversity of information resources and extraction tools used, it is difficult to represent such a construction process in a single and static model. Indeed, in the scenario of ontologies building from texts, the platform must be able to exploit results coming from various Natural Language Processing (NLP) tools. The difficulty of building such a system consists in the implementation of configuration mechanisms for handle various tools. This implementation requires a bit more abstraction in the design.

Borrowed from the MDE, our approach is based on the definition of model transformations. Indeed, the first model, characterizing terms, is transformed, in a stepwise transformation process, to an ontology. In order to be reusable, this transformation process is applied at the meta-modeling level. This paper focuses on (1) the loosely coupled modeling approach that we suggest to manage each step independently, (2) the meta-modeling based approach for handling models of each step so that the models can evolve independently and autonomously and (3) the models transformation-based approach that we propose for handling relationships between models. After an overview of the methodology led by the DAFOE project, we present our meta-model for encoding objects together with concepts and the transformation meta-model allowing to map concepts to some other ones. We finally conclude and give some perspectives about this work.

2 RELATED WORK

2.1 Ontology Building Problem

The study of current tools for developing ontologies leads to classify these tools into two groups. The first group are ontology editors, which assume that the on-

[1]http://dafoe4app.fr

Valéry Téguiak H., Ait-Ameur Y., Jean S. and Sardet E..
INCREMENTAL DESIGN OF ONTOLOGIES - A Model Transformation-based Approach.
DOI: 10.5220/0003099100940103
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development* (KEOD-2010), pages 94-103
ISBN: 978-989-8425-29-4

tology is already designed (on a paper for example), and therefore it remains only to edit it in order to make it interpretable by computers. The second group consists of tools that aim to design ontologies in a supervised work (semi-automatic) according to one or more design steps. Next sections discuss some methods and tools for building ontologies.

2.1.1 Ontologies Editors

In the domain of ontologies design, there exist several design tools. This section presents some of these tools.

Protégé. Protégé (Knublauch et al., 2004) is a platform for developing OWL ontologies. Its architecture, based on plug-ins, allows a designer to add new features like intelligent reasoning, test, maintenance, etc. Originally designed on a frame-based model, nowadays the most used version is Protégé-OWL. This version consists of a set of plug-ins developed above the Protégé kernel and dedicated for ontology construction according to the OWL ontology model.

PLIBEditor. PLIBEditor supports the creation of ontologies conforming to the PLIB ontology model (Pierra, 2003). This editor has the advantage of persisting ontologies objects in a database designed according to a database architecture called ontology-based database (Dehainsala et al., 2007). This database stores ontologies model, ontologies and their instances.

DOE. Unlike Protégé and PLIBEditor that put a strong emphasis on the formal representation of concepts of an ontology, the DOE editor (Troncy et al., 2003) suggests to structure the informal description of concepts to describe more precisely these concepts. This editor uses a specific semantics called "differential semantics" to document the generalization / specialization hierarchies by applying four basic rules (Bachimont et al., 2002): (1) similarity with parent, (2) difference with parent, (3) similarity with siblings and (4) difference with siblings.
In addition to the tool features, annotation techniques allow the designer to keep for example, relationship between ontologies and annotated documents. But another way to maintain this relationship consists in using texts to build ontologies.

2.1.2 Tools for Supervised Design: Designing Ontologies from Texts

Designing ontologies, on the basis of its consensual nature, is a very difficult task. Some tools however, propose building approaches to lighten this problem. Some of these tools provide an ontology construction starting from texts.

Text2Onto. Developed at the University of Karlsruhe, Text2Onto is a tool that implements text mining algorithms on textual data for building ontologies semi-automatically (Cimiano and Volker, 2005). It includes several data processing: terms extraction using either statistical calculations or regular expressions, identification of relations using lexico-syntactic patterns or proximity computation.

TERMINAE. TERMINAE is a software platform supporting the development of terminology and ontology from texts (Aussenac-Gilles et al., 2008). This tool integrates a terminology learning environment, an environment to assist the conceptualization and an ontology management system. Like Text2Onto, TERMINAE stores the link between the designed ontology and the texts. This link takes into account linguistic phenomena like polysemy or synonymy, and keeps track of designer choices about the organization of the ontology hierarchy.

The approach suggested in this paper has been applied to the DAFOE platform that also ranks among the tools for supervised design of ontologies and focuses on setting the transition rules from one step of modeling to another. The setting process is based on model mapping strategies.
Indeed, Text2Onto and TERMINAE assume that one well knows in advance in which concept of the next step a concept of the previous step will be transformed (e.g., a *Class* resulting from a *Term*). In DAFOE, this matching assumption is not done. Thus, a mapping space is needed to represent transformations so that mapping could be easily improved.

3 PRELIMINARIES

This section provides a terminology with definitions used in this paper, explain the goal of mappings and presents the EXPRESS modeling language that we used to formally implements our approach. Some of these definitions are borrowed from the ontology domain (de Bruijn et al., 2004) and are generalized for models in general.

3.1 Definitions

Mapping. A mapping is a specification of the semantic overlap between two models M_s and M_t. Note that a model mapping is often partial, which means that the mapping does not specify the complete semantics overlap between two models, but just a piece of this overlap which is relevant for the mapping application. Throughout this paper, mapping and transformation term will be used indifferently.

Mapping Language. The mapping language is the language used to formalize mappings. Mappings can also be represented through a model. This model, called a mapping model, manages mappings as objects and provides more flexibility on mapping management.

Mapping Constructor. A mapping constructor can be seen as a template for mappings which occur very often. Constructors can be ranked from very simple (e.g., a mapping between a concept and a relation) to very complex, in which case the constructor captures comprehensive substructures of the models, which are related in a certain way.
More discussions on topics around mapping problems and provided solutions can be founded in (Rahm and Bernstein, 2001), (Yan et al., 2001), (Bernstein, 2003), (Euzenat and Shvaiko, 2007), (Claypool et al., 2001).

3.2 Model Mapping Problem

At the heart of the issue of building a mapping between models is the issue of instance mediation (Euzenat and Shvaiko, 2007). By instance mediation, we understand the issue of identifying a target instance from a source instance, each one described by a model. This includes the specification of a mapping model and the use of these mappings to rewrite queries for instance transformation.

Other work (Wache et al., 2001), (Kalfoglou and Schorlemmer, 2003) address the problem of mapping discovery. Without describing the discovery process, we assume that the discovery process has been made. This work deals with mapping specification and instance mediation.

In instance mediation, if we consider a source model M_s and a target model M_t, the question of instance transformation can be summarized by "how to compute a target instance I_t of M_t from a source instance I_s of M_s using the *MAP* mapping between M_s and M_t?" as represented in Figure 1. The *Generic Transformer* (a model-independent program) read an

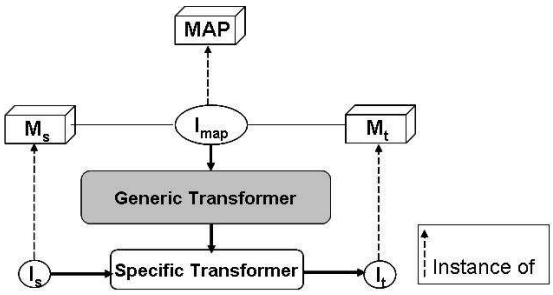

Figure 1: Instance mediation.

instance of mapping (*Imap*) between M_s and M_t and produces a *Specific Transformer* (a program depending on M_s and M_t). This *Specific Transformer* is used to compute I_t from I_s.

3.3 The EXPRESS Modeling Language: an Overview

EXPRESS is a data modeling language that combines ideas from the entity-attribute-relationship family of modeling languages with object modeling ideas of the late 1980s. It became an international standard (ISO 10303-11) in 1994.

The major advantage of this language is its capability to describe structural, descriptive and procedural concepts in a common data model and semantics. This integration avoids the use of several models and languages that require bridging over the gap between all the defined models. A data model in EXPRESS is represented by a set of schemas that may refer to each other. Each schema contains two parts. The first part is a set of entities that are structured in an object-oriented approach. The second part contains procedures, functions and global rules used to express constraints on that entities.

3.3.1 Entity Definition

Entities are named and defined by a set of attributes (which may be an empty set) assigned to each entity. A value domain (a data type) is assigned to each attribute. This value domain can be either simple (e.g., integer, string, etc), a collection domain (e.g., list, set, etc) or an entity type meaning that an attribute value is an instance of another entity.

3.3.2 Constraining Entities

EXPRESS makes possible to limit the allowed set of instances of the data models to those instances that satisfy some given constraints. For example, the age of a person must be a positive integer. EXPRESS

uses first order logic which is completely decidable since the set of instances is finite. Constraints are introduced thanks to the WHERE clause that provides for instance invariant, and to the global RULE clause that provides for model invariant. Derivations, inversions and constraints (respectively illustrated by the *DERIVE, INVERSE, and WHERE* key words) are the only places where functions may occur.

3.3.3 Examples

Tables 1 and 2 illustrate entity definitions. Entity B is described using three attributes b_1, b_2 and b_3 whose datatype are respectively real, list of strings and relationship with another entity A, itself being described using a single attribute a_1. Additionally, two attributes a_2 and a_3 are defined. The a_2 attribute is an inverse attribute of the entity A, corresponding to the inverse link defined by attribute b_3 in the entity B. The a_3 attribute is a derived attribute, that means its value is calculated by a function. Instances may be built from entity definition. An identifier is assigned to each instance for referencing purposes. Instances are then described according to the entity definition they are referred to (using the entity name). The description consists of an enumeration of values compliant with each attribute datatype. It can be noticed that inverse attribute are not represented, because they can be computed later on.

Table 1: Definition of entity A.

ENTITY DEFINITION
ENTITY A; a_1:INTEGER; INVERSE a_2: B FOR b_3; DERIVE a_3: INTEGER:= f(a_1); **END_ENTITY**; FUNCTION f(x: INTEGER): INTEGER; return (x*2) END_FUNCTION;
INSTANCE DEFINITION
#1= A(3);

Table 2: Definition of entity B.

ENTITY DEFINITION
ENTITY B; b_1: REAL; b_2: LIST [0:?] OF STRING; b_3: A; WHERE wr: $b_1 > 0.0$; **END_ENTITY**;
INSTANCE DEFINITION
#2= B(4.0,('hello','world'), #1);

An instance definition is provided for illustration purposes. In Table 1, the "#1" identifier is assigned to the instance. The underlying type of the instance is the entity "A". The integer value "3" is assigned to the a_1 attribute. The derived a_2 attribute is not a user assigned value but an automatically assigned value computed using the value of the attribute a_1. In this example, querying the value of a_2 will return here the value "6" according to the derivation function f. In the same manner, in Table 2, the "#2" instance of the entity B, where b_1 is evaluated to 4.0, b_2 is the list ('hello', 'world') and b_3 reference the particular instance "#1" of the entity A.

4 THE DAFOE METHODOLOGY

To face with the difficulty of building ontologies, the DAFOE project proposes a stepwise ontology design approach. As stated in introduction, this methodology consists of 3 main steps: Terminological, Termino-Conceptual and Ontological. This paper does not address the ontology learning problem (Navigli and Velardi, 2004), (Maedche and Staab, 2001). Nevertheless, it provides an infrastructure with model transformation (cf. Section 5) in order to enrich the building process.

4.1 DAFOE Terminological Step

The terminological step takes a corpus of texts as input. This corpus needs to be processed by NLP tools such as term extractors in order to extract terms and their relationships . As an alternative, an ontology designer can use a preexisting terminology. The underlying assumption is threefold: text analysis can extract term candidates that are relevant for a given domain. These terms are likely to be turned into ontology concepts and the distribution of these terms reflects their semantics (Harris, 1968). The result of this step is represented as instance of the terminological model shown in Figure 2. A *Saillance* entity is associated to a Term so that terms can be filter according to some "saillance" criteria like its frequency.

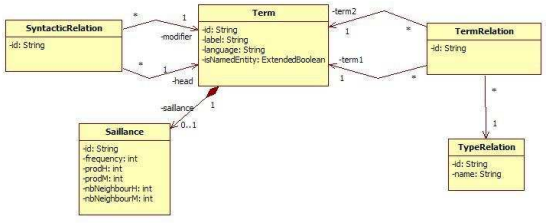

Figure 2: Terminological model: UML diagram excerpt.

4.2 DAFOE Termino-conceptual Step

This step transform the terms resulting from the previous step into a structure of unambiguous termino-concepts (TC) and termino-conceptual relations (TCR). As an alternative, the knowledge engineer may build that step by importing a preexisting termino-conceptual resource such as a thesaurus or results of the analysis of the terminological step. In this case, he/she analyses the meanings of terms and relations that appear at the terminological step with respect to each other by taking into account their occurrences. Then, he/she can cluster terms and relationships that have the same meaning, distinguish the various meanings of ambiguous terms, compare the contexts in which they are used.

The termino-conceptual step is central for transforming linguistic elements into conceptual ones and for tracing the ontology concept back to the linguistic elements they are coming from. *TCs* are associated with differentiation criteria, for describing them by differences and similarities as proposed by the Archonte method (Bachimont et al., 2002). Four textual fields make explicit semantic differences and similarities of a *TC* with its father and siblings. This traceability improves ontology readability and maintenance. Terms and terminological relations that are connected to termino-conceptual elements are said ”conceptualised”. The result of this step is represented as instance of the Termino-conceptual model shown in Figure 3.

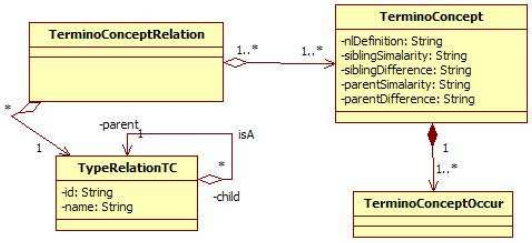

Figure 3: Termino-conceptual model: UML diagram excerpt.

4.3 DAFOE Ontological Step

The ontology data model formalizes *TCs* and *TCRs* in a formal language equivalent to OWL-DL. Concepts are described as classes, individuals as instances of classes, properties between classes as object properties or datatype properties. An automatic process translates *TCs* and *TCRs* into formal concepts in a hierarchy with inherited properties subsumption. This translation exploits the structure of the semantic network represented in the termino-conteptual step and the differential criteria associated with *TCs* and *TCRs*. The result of this step is represented as instance of the Ontological model shown in Figure 4.

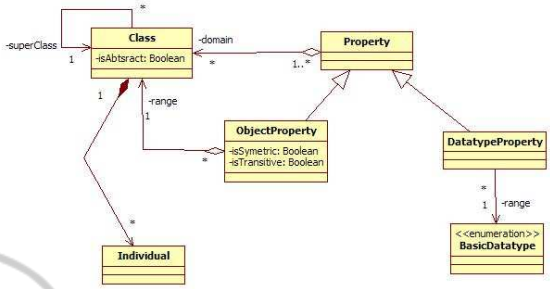

Figure 4: Ontological model: UML diagram excerpt.

The next section presents the two main approaches to express transformation between models and explains our proposal for representing relationship between steps of the DAFOE methodology.

5 MODELING RELATIONSHIPS BETWEEN STEPS

Due to the numerous types of information managed in each step of the methodology led by DAFOE, we explicitly separate the modeling of each step and then bind these steps through a transformation layer. This loosely coupled modeling of the DAFOE methodology eases model evolution of each step independently. Figure 5 presents an overview of this process.

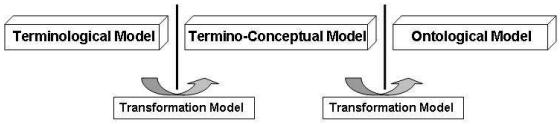

Figure 5: Process for modeling relationships between steps.

When encoding this transformation process, two approaches are possible: (1) hard encoding of the transformation or (2) specification of the transformation as an instance of a transformation meta-model. These two approaches are discussed below.

5.1 A First Attempt: Hard Encoded Transformation

The first approach consists in hard encoding the mappings between the concepts of the source and target models.

5.1.1 Managing Transformations

Representing transformations between models requires to be careful about transformation arities. For example, the 1:1 arity means that one source entity is mapped to a single target entity, the N:1 (respectively 1:N) arity means that many sources entities (respectively one source entity) may be transformed together to one target entity (respectively many target entities). It is also possible to have a N:M arity.

The following EXPRESS code represents an illustration of modeling transformations. Derivation functions are defined: they produce target entities from source entities. We just present a simple part with a 1:1 transformation between *TERM* and *TC* (in the entity *E_TERM_to_TC_1_1*) or between *TC* and *CLASS* (in the entity *E_TC_to_CLASS_1_1*). In this approach, the transformation from the source element to the target element is embedded in a program, here a derivation function (build_target function). This function shows for example that the property *id* of a *TERM* matches with the property *id* of a *TC*. In the same way, the label of a *TC* is the concatenation of the String 'tc_' with the *TERM.label*.

Table 3: Hard_encoded_transformation_TM1.

```
SCHEMA hard_encoded_transformation_TM1;
ENTITY E_Term_to_TC_1_1;
      source: TERM;
      DERIVE
      target: TC:= build_target(SELF.source);
END_ENTITY;
FUNCTION build_target(t: TERM): TC;
      LOCAL;
      tc:= !TC;
      END_LOCAL;
      tc.id:= source.id;
      tc.label:= 'tc_'+source.label;
      ...;
      return (tc);
END_FUNCTION;
      ...;
END_SCHEMA;
```

5.1.2 Limitations

This solution is not satisfactory. Indeed, assume that one changes the structural representation of the model of one step (for example, the entity *TERM* is modified). In this case, the build_target function of the entity *E_TERM_to_TC_1_1* must be rewritten. Another drawback of this approach is the necessity to master in advance the different possible transformations before building a sequence of transformations between the entities of each model. So, the "hard encoded" transformation approach is model-dependent. It is then not recommended in a context of heterogeneous models with various transformation rules. Indeed, the transformation is highly tiled to structural representation of the models at a given time. Consequently, one needs to manipulate models and their transformation at each evolution. Terminae, Text2Onto,... are example of such tools.

Table 4: Hard_encoded_transformation_TM2.

```
SCHEMA hard_encoded_transformation_TM2;
ENTITY E_TC_to_CLASS_1_1;
      source: TC;
      DERIVE
      target: CLASS:= build_target(SELF.source);
END_ENTITY;
FUNCTION build_target(tc: TC): CLASS;
      LOCAL;
      cls:= !CLASS;
      END_LOCAL;
      ...;
      return (cls);
END_FUNCTION;
      ...;
END_SCHEMA;
```

5.2 Our Approach: Transformation as Instances

The main idea behind our approach is, on the one side, to deal with the numerous transformation programs to be written and on the other side, to offer a configurable space for setting up correspondences between steps. Furthermore, one does not need to known in advance all the possible transformations. Indeed, if we look closely at the *hard_encoded_transformation_TM1* and *hard_encoded_transformation_TM2* , we notice a set of similarities like:

s1) Both programs transform a source entity of a source model in a target entity of a target model.

s2) Inside both programs, we find code statements (the *build_target* function) that transform a kind of data to a kind of another data (*TERM* to *TC* or *TC* to *CLASS* for example).

The objective of a data model based transformation is to abstract the previous similarities by offering the capability to replay a single transformation on different model instances. In this case, meta-modeling techniques are required not only for representing data model instances, but also the transformation function. Indeed, functions are reified by data models that may be instantiated several times. A single interpretation function is associated for code generation. This process has been adopted on the DAFOE platform, it is described below. Using a more abstract specification, one can handle these similarities in a generic

way. Thus, as presented in Figure 6 we use a meta-model for representing the models of each steps and we also use a meta-model of transformation for building transformation dynamically.

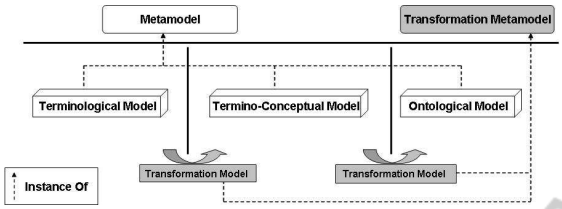

Figure 6: Our approach: a generic process.

5.2.1 A Metamodel for Concept Modeling

Meta-modeling appears as the use of a modeling formalism to represent a modeling formalism. This vision of models has two fundamental advantages: (1) it is possible to represent in the same way models and instances and (2) it is also possible to write programs based on both metamodel and models.

The Entity-Attribute Meta-model. For handling model life cycle during their transformations, we need a model to express models (the metamodel). For us, a model will consist of a set of entities, instances of these entities, and attributes to describe these instances (cf. Figure 7). Instances representation is also based on a model (instance model). This instance model is used for example, to migrate instances of a source model to a target model using mappings between these two models.
For our experiments in ontology design, the entity-attribute model has proved to be powerful enough to play the role of meta-model for Terminological, Termino-conceptual and ontological models.

5.2.2 Examples

The previous Entity-Attribute meta-model can be instantiated in order to describe the terminological, termino-conceptual and ontological models.

5.2.3 Encoding Transformations

In the same manner, we propose a meta-model supporting the description of transformations. This meta-model is composed of two main parts: the structural mapping model and the procedural mapping model.

The Structural Mapping Model. This model records the different mappings that could be set up between entities of the source model and the ones of

Table 5: Instantiate the terminological model.

```
– Create Terminological model ;
#1=MODEL(1, 'Terminological');
...
– Create entities
#4=ENTITY('Term', #1, $);
#5=ENTITY('TermRelation', #1, $);
...
– Create simple attribute type
#8=STRING_TYPE('String');
#9=INT_TYPE('Integer');
...
– Create attributes of Term
#10=ATTRIBUTE('id', #4, #9);
#11=ATTRIBUTE('label', #4, #8);
#12=ATTRIBUTE('language', #4, #8);
...
– Create attributes of TermRelation
#15=ATTRIBUTE('id', #5, #8);
#16=ATTRIBUTE('term1', #5, #4);
#17=ATTRIBUTE('term2', #5, #4);
```

Table 6: Instantiate the TerminoConceptual model.

```
– Create TerminoConceptual model
#20=MODEL(20, 'TerminoConceptual');
...
– Create entities
#21=ENTITY('TerminoConcept', #1, $);
#22=ENTITY('TerminoConceptRelation', #1, $);
...
– Create attributes of TerminoConcept
#23=ATTRIBUTE('id', #21, #9);
#24=ATTRIBUTE('label', #21, #8);
#25=ATTRIBUTE('dialect', #21, #8);
...
```

the target model. Cardinalities and typing constraints are identified in such model. An extract of the structural mapping model is described on Figure 8.

The Procedural Mapping Model. This model supports the description of the derivation functions and of the constraint expressions. In other words, this model provides for any resources needed to build either expressions for computing target concepts from source ones, or to define boolean expressions that describe the constraints that may apply on a given concept. Figure 9 illustrates the simplified expression model that we used. Any kind of expression is a subtype of an abstract *Generic Expression* that we represent by an acyclic graph. Since we provide a formal approach with model checking principle, a consistence rule ensure that an expression is acyclic. Furthermore, this expression model is not only limited to boolean, numeric or string expression. It can be extended for handling numerous types of expression. Once the data model describing the transformation is set up, MDE

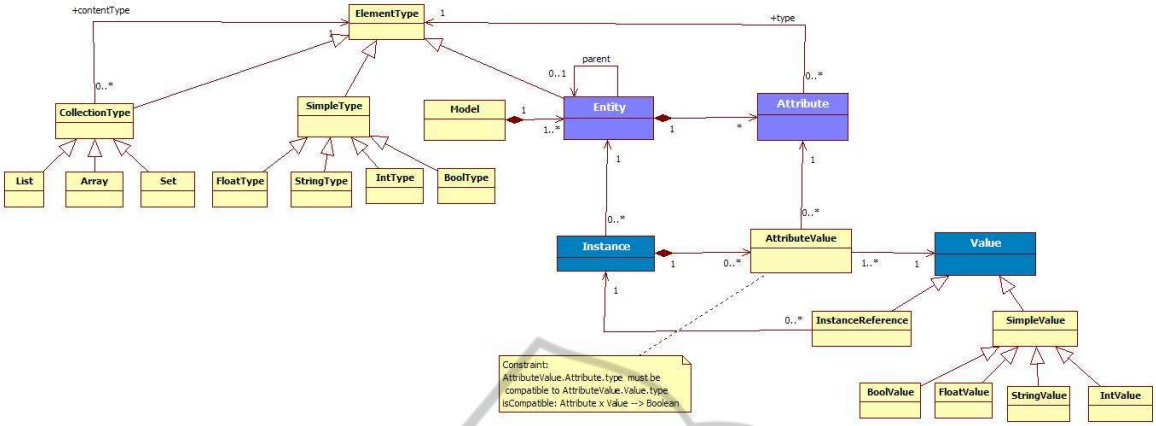

Figure 7: The Entity-Attribute Metamodel.

techniques are used. Then, either the EXPRESS data modeling language is used to run transformations, or a specific program interpreting instances is written. In our case, we have chosen the second option.

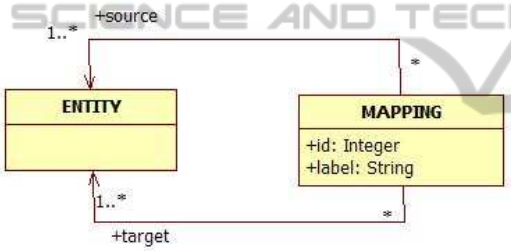

Figure 8: Structural Mapping Metamodel.

Notice that with the Expression model, we are already able to write mappings. These syntactic mappings can be seen as a repository of mapping patterns. It remains to create a semantic environment to bind variables and constants of syntactic mappings to concepts of the meta-model. The semantic mapping results to the projection of syntactic mapping in a semantic environment. Figure 10 presents the projection model used to add semantics to mappings. It is important to note that unlike SWRL (Semantic Web Rule Language) (Horrocks et al., 2004), the projection model is not only used for writing formal parameters. A syntactic mapping can be reused in a new environment with a new semantics. As example, the computation of a *TC* from a *Term* is illustrated in examples below (cf. Section 5.2.4).

5.2.4 Examples

The following EXPRESS instances shows a partial view of the mapping between TERMs and TCs. In this example, we present how to express that *TC.label = 'tc_'+TERM.label*.

Table 7: Instantiate the TerminoConceptual model.

```
– create variables and constant
#30= ElementVar(3O, 'X');
#31= ElementVar(31, 'Y');
#33= CONSTANT('tc_');
– create the concatenation #34= ConcatExp(#33, #31);
#35= FunctionalRule(#31, #34);
#40= MAPPING('TERM_TO_TC_1_1, #35);
...
– #50= VariableEnvironment(#11, #30);
#51= VariableEnvironment(#24, #31);
#60= SEMANTICS(#40, (#50, #51));
...
```

6 CONCLUSIONS

This paper presents a stepwise ontology design methodology. The proposed approach uses MDE approaches in order to define model transformations. It consists in identifying the meta-models of the manipulated (transformed) models and the one of the possible mappings. It has been fully formalized with the EXPRESS modeling language and a prototype has been developed on the top of the OntoDB ontology based database in order to persist instances resulting from models instantiations. It has been also integrated in the DAFOE platform, an open source platform for building ontologies from texts, terminologies, thesauri or existing ontologies. The proposed mapping approach has been experimented by the DAFOE consortium. In the medical course for example, encouraging results has been obtained when building ontology of pneumonology from a corpus of the same domain.

Currently, mappings defined with our transformation meta-model are unidirectional and we are still working on making them bidirectional. Another chal-

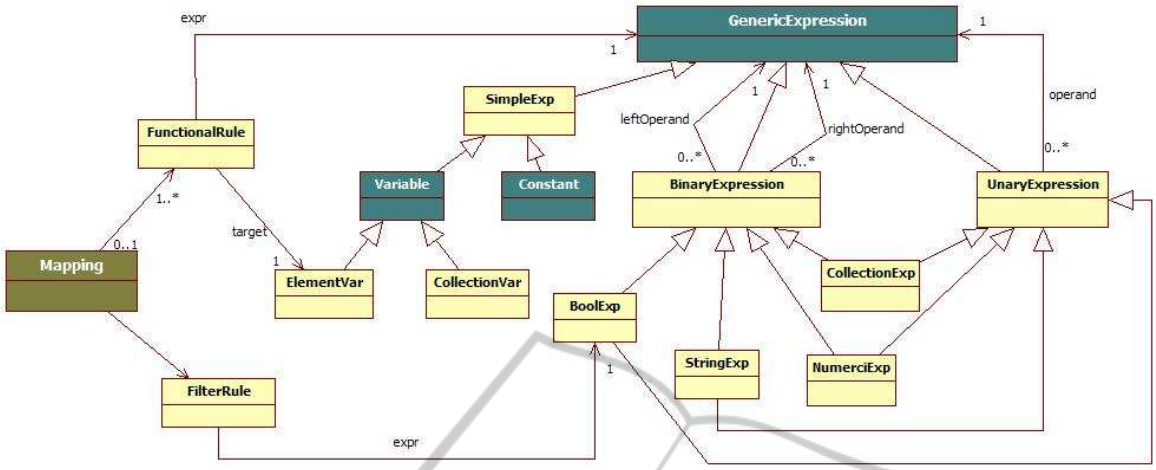

Figure 9: Simplified model for mapping and expression.

lenge will be to define a mapping language in a database persistent context supporting our mapping meta-model. Indeed, most of the current mapping languages use a XML representation for the description of mappings. However, to ensure scalability of our approach, manipulated data are stored in a database. For this reason, it would be interesting to have a mapping constructors integrated in the query language of underlying database environment. A possible issue will be to look closely at the Ontology Query Language (OntoQL) (Jean et al., 2006), the exploitation language associated with OntoDB to manage both ontologies and data. We will study how to extend OntoQL with mapping operators so that it will be possible to query both created models and mappings in a database environment.

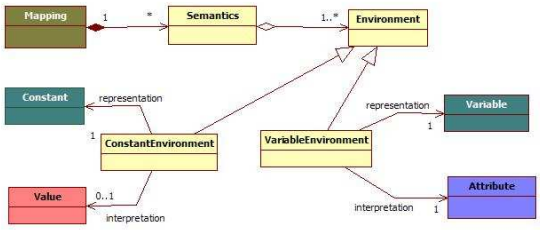

Figure 10: Model for semantic interpretation.

ACKNOWLEDGEMENTS

The authors would like to thank the partners of the ANR DAFOE project for their contribution.

REFERENCES

Aussenac-Gilles, N., Despres, S., and Szulman, S. (2008). The TERMINAE method and platform for ontology engineering from texts. In *Bridging the Gap between Text and Knowledge*. IOS Press.

Bachimont, B., Troncy, R., and Isaac, A. (2002). Semantic commitment for designing ontologies: a proposal. In *European Conference on Knowledge Modelling and Knowledge Management, Ekaw 2002*, Sigenza, Spain. Springer Verlag, LNCS.

Bernstein, P. A. (2003). Applying model management to classical meta data problems. In *CIDR*, pages 209–220. CIDR SIGMOD Record.

Cimiano, P. and Volker, J. (2005). Text2onto - a framework for ontology learning and data-driven change discovery. In Montoyo, A., Munoz, R., and Metais, E., editors, *Proc. of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB)*, volume 3513 of *Lecture Notes in Computer Science*, pages 227–238, Alicante, Spain. Springer.

Claypool, K. T., Rundensteiner, E. A., Zhang, X., Su, H., Kuno, H. A., Lee, W.-C., and Mitchell, G. (2001). Gangam - a solution to support multiple data models, their mappings and maintenance. In *SIGMOD Conference*, page 606.

de Bruijn, J., Foxvog, D., and Zimmerman, K. (2004). Ontology mediation patterns library. Deliverable D4.3.1, SEKT.

Dehainsala, H., Pierra, G., and Bellatreche, L. (2007). Ontodb: An ontology-based database for intensive applications. In *12th International Conference on Database Systems for Advanced Applications(DASFAA'07)*, pages 497–508.

Euzenat, J. and Shvaiko, P. (2007). *Ontology matching*. Springer-Verlag, Heidelberg (DE).

Harris, Z. (1968). *Mathematical Structures of Language*. Interscience Publishers.

Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004). SWRL: a semantic web rule language combining OWL and RuleML. http://www.w3.org/Submission/SWRL/.

Jean, S., Ait-Ameur, Y., and Pierra, G. (2006). Querying ontology based database. the ontoql proposal. In *IN: 18 th International Conference on Software Engineering and Knowledge Engeeniring*, pages 166–171.

Kalfoglou, Y. and Schorlemmer, M. (2003). IF-Map: an ontology mapping method based on information flow theory. *Journal on Data Semantics*, I:98–127.

Knublauch, H., Fergerson, R. W., Noy, N. F., and Musen, M. A. (2004). The protg owl plugin: An open development environment for semantic web applications. In *International Semantic Web Conference*, pages 229–243. Springer.

Maedche, E. and Staab, S. (2001). Ontology learning for the semantic web. *IEEE Intelligent Systems*, 16:72–79.

Navigli, R. and Velardi, P. (2004). Learning domain ontologies from document warehouses and dedicated web sites roberto navigli and paola velardi. *Computational Linguistics*, 30:2004.

Pierra, G. (2003). Context-explication in conceptual ontologies: The plib approach,. In R. Jardim-Gonsalves, J. C. and Steimer-Garo, A., editors, *Proc. of 10th ISPE International Conf. on Concurrent Engineering: Research and Applications (CE'03) : Special Track on Data Integration in Engineering*, pages 243–254, Madeira, Portugal. UNINOVA.

Rahm, E. and Bernstein, P. (2001). A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350.

Troncy, R., Isaac, A., and Malais, V. (2003). Using xslt for interoperability: Doe and the travelling domain experiment. In *Proc. 2nd workshop on evaluation of ontology-based tools (EON), Sanibel Island (FL US)*, pages 92–102.

Wache, H., Voegele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., and Hübner, S. (2001). Ontology-based integration of information – a survey of existing approaches. In *Proc. IJCAI Workshop on Ontologies and Information Sharing*, pages 108–117, Seattle (WA US).

Yan, L. L., Miller, R. J., Haas, L. M., and Fagin, R. (2001). Data-driven understanding and refinement of schema mappings. In *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 485–496, New York, NY, USA. ACM.