Using Natural Language to Improve the Generation of Model Transformation in Software Design

Jean-Remi Falleri
Univ. Montpellier 2
and LIRMM-CNRS
161 Ada Street F34392 Montpellier
France
falleri@lirmm.frl

Violaine Prince
Univ. Montpellier 2
and LIRMM-CNRS
161 Ada Street F34392 Montpellier
France prince@lirmm.fr

Mathieu Lafourcade
Univ. Montpellier 2
and LIRMM-CNRS
161 Ada Street F34392 Montpellier
France
lafourcade@lirmm.fr

Michel Dao
Orange Labs
Issy Les Moulineaux
France
michel.dao@orange-ftgroup.com

Marianne Huchard
Univ. Montpellier 2
and LIRMM-CNRS
161 Ada Street F34392 Montpellier
France
huchard@lirmm.fr

Clementine Nebut
Univ. Montpellier 2
and LIRMM-CNRS
161 Ada Street F34392 Montpellier
France
nebut@lirmm.fr

***Abstract*—Among the present crucial issues in UML Modeling, one of the most common is about the fusion of similar models coming from various sources. Several similar models are created in Software Engineering and it is of primary interest to compare them and, when possible, to craft a general model including a specific one, or just identify models that are in fact equivalent. Most present approaches are based on model structure comparison and alignment on strings for attributes and classe names. This contribution evaluates the added value of several combined NLP techniques based on lexical networks, POS tagging, and Dependency Rules application, and how they might improve the fusion of models. Topics : use of NLP techniques in practical applications.**

I. Introduction

Natural Language Processing (NLP) is more and more a topic of interest for Model Driven Engineering in Software Design. Software is designed worldwide for almost every type of task involving information, and has to be exchanged between different teams geographically and temporally distant. For the same kind of applications, one might find several 'metamodels' (*i.e.* abstract 'meta' specifications) independently developed, and also several versions of the same metamodel with different names and designations. Since those abstract structures generate various software specifications (called models), then compatibility needs to be ensured. Usually this problem is solved using manually written and *ad hoc* model transformations. The latter are not difficult to write per se, but are so numerous that they heavily impact the project work load. Members Sotfware Engineering community has thus suggested to NLP researchers to help them to find astute methods to automatically generate an alignment between two similar metamodels. Schema alignment already exists in domains such as semantic web, ontology integration, e-commerce and so forth. It takes as input two schemas and produces as output a set of relations (*e.g.* equivalence and subsumption) between the entities of the two input schemas. Concretely, a schema can be an XML Schema, an ontology, a database schema or an object-oriented class model. Despite the variation between these formats, the mechanisms involved to perform the match operation are highly similar. The NLP community has already contributed to facilitate Schema alignment [Rahm and Bernstein 2001]) whether for databases (*e.g.*, [Duchateau et al., 2007]), conceptual graphs (*e.g.*,[Montes y Gomez et al., 2007]), or domain specific ontologies (*e.g.*, [Fan et al. 2007] for medical ontologies). The meta-model alignment operation aims at finding a set of correspondences between elements (classes, attributes, references and enumerations) from a source meta-model and elements from a target metamodel. Those correspondences can be used later in several tasks:

- Automatic generation of a model transformation,
- Comparison of two models conforming to two different meta-models,
- Increasing efficiency of model merging or composition, as the last step after model transformation and model comparison.

This contribution focuses on the first step, as a necessary requirement for model merging. The goals our work is intended to achieve are the following:

- to discover possible relations between entity identifiers appearing in models: we would thus rely on the possible lexical or semantic relationships induced by names assigned to the model entities. For instance, if two class identifiers are synonyms in a thesaurus, this might suggest a possible redundancy between those two classes.
- if models elements are generated by meta-modeling tech-

niques without identifiers, to try to assign them names according to the semantics of the surrounding other elements (a topically driven name assignment).

Since both goals need an extensive description, this contribution sticks to the first of the preceding items. Lexical relations between identifiers are at the core of the added value of NLP to this task. In next section, the important lexical relations and their modeling are explained. Then the application is detailed in the following section: how compound identifiers are segmented, tagged with a POS tagger, and a dependency analysis assigns a function and induces ontological relations between terms. Experiments have been run on an existing UML modeling corpus, and their results show that NLP techniques have largely enhanced the automation of models transformation. Conclusion summarizes the benefits from such a cooperation and indicates the next tracks that are currently followed in order to succeed in this task.

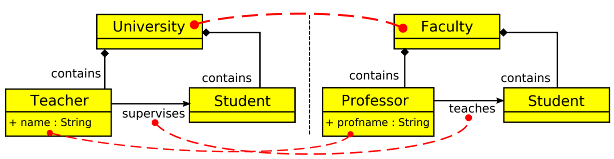

Fig. 1. Example of two models comparison under a general context. Correspondances are found, mainly synonymy, and most certainly those two models could be fusioned into one.

II. Modeling Lexical Relations Between Items Identifiers

Discovering lexical relations between identifiers (terms) appearing in models, seemed to be the first step for a semantic approach of models before transformation. We focused on possible ontological relations (synonymy, hyponymy, ...) between identifiers. Those relations are defined on the set of terms, but they are mostly context sensitive. For example, in a medical context *affection* is a synonym of *disease*, but in another context it may not be the case. Thus, to achieve a correct modeling, relations have to be context-dependent.

A. Basic Items

The **set of terms** is the set of correct identifiers in the models. A **context** is the formalization of a given domain where terms can hold specic relation occurences. Simply speaking, a context is a term (or more generally a set of terms) that specializes a given term. If the context is empty, then we assume we are in the most general domain of common knowledge. Let T be a set of terms. If c belongs to T then it can be a context. For example, *plane — aeronautics* is close to *aeroplane — aeronautics*. But, *plane — aeronautics* and *plane — mathematics* certainly do not refer to the same meaning. Terms are assumed to be available through a lexical network. In its most general definition, a lexical network is a set of words (with or without specific context) linked together with relations. Relations can be of various types [Budanitsky and Hirst 2006], ontological (in that case, we speak more often of ontology) and/or lexical (like synonyms, part-of-speech, lemma, etc.). Wordnet ([Miller 1994]) and EuroWordnet ([Vossen 1998]) are typical examples of lexical networks.

For our purpose relations have to be designed with a set of binary predicates describing their properties and rules, determining the nature of the relation. The useful properties are:

- Transitivity: does a relation propagate whenever true?
- Reflexivity: is the relation self-relevant?
- Symmetry: does the relation introduce an order or does it create a possible "similarity"?

The **basic relations** between terms on which we have most focused are:

- Synonymy, restricted to contextual synonymy, that is, when meanings are close according to a given context,
- Hyperonymy, also a contextual relationship, when a term seems to be the name of a "superclass" of a given class, in the model,
- Hyponymy, as the symmetric relation to hyperonymy,
- To a lesser extent, relations such as Meronymy / Holonymy (or part-whole relations), which might appear in some models and for which modeling has only awkward answers to provide.

Other derived relations, such as co-hyponymy or directy hyperonymy or hyponymy are also described hereafter, because of their usefulness to modeling in sofware design.

B. Relations Definitions For Model Transformation

Here follows the definitions of our relations:

1) Hyperonymy: A term $t1$ is a **hyperonym** of $t2$ if it is more general. For example, *vehicle* is a hyperonym of *car* in the context of transports. Here follow some properties if the binary relation$hyper$:

- *transitive*: if $hyper(t1, t2)$ and $hyper(t2, t3)$ then $hyper(t1, t3)$ (example: vehicle, car, fiat 500)
- *strongly antisymmetric*

We can thus consider $hyper$ as a strict partial order on $T|c$.

2) Hyponymy: A term $t1$ is a **hyponym** of $t2$ if it is more specific. For example, *dog* is an hyponym of *animal*. Mathematically, $hypo$ can be also modelized as a binary relation, then we have $hypo(dog, animal)$. Here, follow some properties of the binary relation $hypo$:

- *transitive*: (example: *labrador*, *dog*, *animal*)
- *strongly antisymmetric*

We can thus consider $hypo$ as a strict partial order on $T|c$.

The relations $hyper$ and $hypo$ are the inverse of each other, thus if $hyper(t1, t2)$, then $hypo(t2, t1)$.

3) Synonymy: A term $t1$ is a **synonym** of $t2$ under the context c if it is equivalent to $t2$. For example, *car* is a synonym *automobile* under the context of transports. Here follow some properties of the binary relation sym:

- *transitive*,
- *reflexive*,
- *symmetric*.

We can consider syn as a equivalence relation under $T|c$. One can notice that if, linguistically speaking, reflexivity is not relevant (a term being synonym of itself is not something to be considered as an interesting achievement), for software design purposes, this property introduces this equivalence relation that creates a *class of terms*, the use of which is quite obvious in model comparison.

4) Direct Hyperonymy: A term $t1$ is a **direct hyperonym** of $t2$ if it is directly more general. For example, *vehicle* is a direct hyperonym of *car* under the context of transports. Here follow some properties if the binary relation $dhyper$:

- *strongly antisymetric*

Moreover, we have $dhyper(a, b) \rightarrow hyper(a, b)$.

5) Direct Hyponymy: A term $t1$ is a **direct hyponym** of $t2$ if it is directly more specific. For example, *dog* is a direct hyponym of *animal*. Mathematicaly, $dhypo$ can be also modelized as a binary relation, then we have $dhypo(dog, animal)$. Here follow some properties if the binary relation $dhypo$:

- *strongly antisymmetric*

Morevover, we have $dhypo(a, b) \rightarrow hypo(a, b)$. The relations $dhyper$ and $dhypo$ are the inverse of each other, thus if $dhyper(t1, t2)$, then $dhypo(t2, t1)$.

C. Derived Relations and Less Frequent Relations

As explained before, modeling needs to provide horizontal relations between identifiers, and not only 'vertical' ones. Two terms are cohyponyms, if they are both hyponyms of a common term. Co-hyponymy frequently appears in models created by different teams, and is an issue in models comparison. However, as the notion of co-hyponymy depends strongly on the maximal distance of the common term we want to accept (all terms are cohyponyms of the most general one), we define several versions of this relation.

1) 1co-hyponymy: **1co-hyponymy** indicates that two terms are "children of the same direct parent'.
$1cohypo(a, b) \leftrightarrow dhypo(a, c) \wedge dhypo(b, c) \wedge a \neq b$. As $dhypo$ and $dhyper$ are inverse, we have: $dhypo(a, b) \leftrightarrow dhyper(b, a)$. Then, $dhypo(a, c) \wedge dhypo(b, c) \wedge a \neq b \leftrightarrow dhyper(c, a) \wedge dhyper(c, b) \wedge a \neq b$. Thus, $cohypo(a, b) \leftrightarrow hyper(c, a) \wedge dhyper(c, b) \wedge a \neq b$. Here follow some properties of the relation $1cohypo$:

- *symmetric*.

2) θco-hyponymy: The θco-hyponym is a formalization of a generalized co-hyponymy.
Two terms a and b are θco-hyponyms if a there is common hyperonym h between a and b such as $dmin(a, h) \leq \theta$ and $dmin(b, h) \leq \theta$, $dmin(x, y)$ being the shortest path between two comparable terms for the relation $dhypo$. We call h a θminor of a and b. Here follow some properties of the relation $\theta cohypo$:

- *symmetric*.

We can notice that $\theta cohypo(a, b) \rightarrow \varphi cohypo(a, b), \varphi \geq \theta$.

Here follow an extended version of the θco-hyponymy for a n-tuple $x_1, \ldots, x_n$ argument.

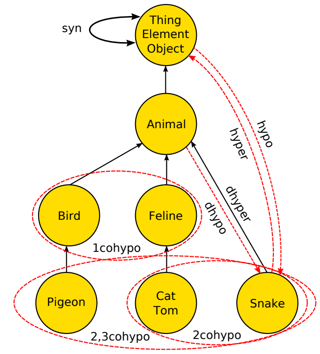

Fig. 2. Lexical relations between terms

3) $\theta_n$co-hyponymy: n terms $x_1, \ldots, x_n$ are $\theta_n$co-hyponymys if there is a common hyperonym h betwenn $x_1, \ldots, x_n$ such as $\forall i \in [1, n]$, $dmin(x_i, h) \leq \theta$, $dmin(x, y)$ being the shortest path between two comparable terms for the relation $dhypo$. We call h a $\theta_n$minor of $x_1, \ldots, x_n$. Here follow some properties of the relation $\theta_n cohypo$:

- *symmetric*.

We can notice that $\theta_n cohypo(x_1, \ldots, x_n) \rightarrow \varphi cohypo(x_1, \ldots, x_n), \varphi \geq \theta$. Moreover, if n terms are $\theta_n$co-hyponyms, then any set of cardinality larger than those n terms, contains also $\theta_n$co-hyponyms.

A graphical summary between the lexical relations are displayed in 2

4) Meronymy: A term $t1$ is a **meronym** of $t2$ if $t2$ is semantically of part of $t1$. For example, *wheel* is a meronym of *car* under the context of transports. Here follow some properties of the relation $mero$:

- *transitive* (car, wheel, rim),

5) Holonymy: A term $t1$ is a **holonym** of $t2$ if $t1$ contains semantically $t2$. For example, *body* is an holonym of *arm* under the context of anatomy. Here follow some properties of the relation $holo$:

- *transitive* (finger, hand, arm),

$mero$ and $holo$ are inverse relations, in effect if $mero(t1, t2)$, then $holo(t2, t1)$.

D. Composing relations

To summarize, we have now:

- One equivalence relation $T|c$ (syn),
- Two strict partial order relations $T|c$ ($hyper$, $hypo$),
- One symmetric relation $T|c$ ($\theta cohypo$),
- Two only transitive relations $T|c$ ($mero$ and $holo$).

Software designers have been interested in investigating if possible combinations of relations might occur, as a path to relate two items in their design. Therefore, we have worked on the properties of relation composition. More precisely, as syn is a equivalence relation on $T|c$ it is possible to define equivalence classes on $T|c$. We write $[x]$ the equivalence class of an element $x \in T|c$. A property of an equivalence class is as follows: $\forall y \in [x], \forall z \in [x], syn(y, z)$. Thus, we obtain the following property:

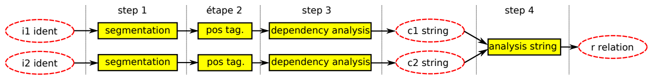

Fig. 3. Illustration of the overall process applied to identifier. At the end two identifiers could be compared for relation identification.

Relating Elements For Property Deduction
If there is a relation $r(x, y)$ (r can be any of: $hypo$, $hyper$, $dhyper$, $dhypo$, $cohypo$, $mero$ and $holo$) between x and an element y, then $\forall e \in [x], \forall f \in [y], r(e, f)$.

Here follow some examples to illustrate this property. We have two equivalence classes on the set of terms under the *anatomy* context: $(body)$ and $(skull, head)$. It is trival that $(body)$ is an equivalence class, and we do have $syn(skull, head)$. As we have moreover $mero(body, head)$, from the previous property, we deduce $mero(body, skull)$.

E. Importing Lexical Relations

In order to make good use of the rules we have defined above, it is necessary to construct several *initial relation occurences* between the identifiers of a model. In particular, this is necessary for initial syn and $dhypo$ occurences. In fact, other relations occurrences (for $dhyper$, $hyper$, $hypo$, $cohypo$) can be deduced from the former ones. Thus, we consider the initial set of occurrences of syn and $dhypo$ as a starting point under the set of identifiers. In practice, this set is available in any general use lexical network.

What follows is the description of some processing aiming at discovering lexical relations occurrences (synonymy, hyponymy, ...) between identifiers present in models. We define the $relids$ application.

- Let $T|c$ be the set of terms with a context c,
- Let $LEX = \{SYN, HYPO, HYPER, COHYPO\} \cup \emptyset$ be the set of the name of the lexical relations,

The application $relids$ is defined as follows:

$$relids : T|c \times T|c \rightarrow LEX \tag{1}$$

III. Extracting relations from identifiers: tools and rules

In order to compute the result of the application of $relids$ on an element $T|c \times T|c$, we setup the following process (figure 3) :

1) Segmentation: identifiers, which are generally a concatenation of terms, are cut into atomic terms (segments),
2) Labelling: a part of speech (POS) is given to each segment,
3) Dependency analysis: each segment is organized according to a dependency relation (which one is the head?),
4) Relation identification: the strings related to identifiers are analysed in order to determine the lexical relation between them.

A. Segmentation

In design (conceptual) model as well as in implementation models, the various elements (classes, attributes, operations, parameters, application programming interface, ...) are usually named with identifiers. As those identifiers are chosen by designers and/or programmers, they constitute (quite often) some clear information regarding the understanding of the model. It is largely accepted that identifiers of a given model are supposed to be easily associated (for a human being at least) to concepts or terms of the real world. Considering the naming restriction imposed on identifiers by programming languages or model syntax (for example, C or Java languages or UML), identifiers are often concatenations of various words, acronyms or word shortenings and abbreviations (like *getnextwarning*, *getimplementation* or *getimpl*). Those words are most of the time put into emphasis with a variation of case (upper and dowcase like in *getNextWarning*) or separators (*get_next_warning*).

The procedure of segmentation of an identifier into elements is thus the natural first step in order to retrieve possible lexical chains and match them with terms.

- Let $T|c$ be the set of valid terms under the context c,
- Let $W|c$ be the set of valid terms ($W|c \subseteq T|c$),

$$seg : T|c \rightarrow \bigcup_{i=1}^{\infty} (W|c)^i \tag{2}$$

Segmentation relies on clues. Here follows the different types of clues that can specify a segment end inside a given identifier:

- A case change (uppercase dans downcase) (*getNextWarning*),
- A separator (*get_next_warning*),
- Some numercial characters (*Block12*).

It occurs frequently that clues of several types appear for the same identifier (*stdlib_GetWarning123*). To take into account this fact, we specified several segmentation strategies, which are therafter combined into a global strategy. What follows is the description of the various strategies.

1) Separator based strategy: A separator list is given (term tagged as separator in the lexical network). An identifier is read from left to right character by character. As soon a separator is encountered, a segment is created (*date_label* becomes *(date,label)*).

2) Numerical character based strategy: We separate numerical characters from alphabetical ones (*Block122* becomes *(Block,122)*).

3) Case strategy: Here, the variation between uppercase and downcase in a given identifier is used as a clue for segmentation. For example, we have:

- *nextThing*: next, Thing.
- *ClassLoader*: Class, Loader.
- *RCAProcess*: RCA, Process.

It should be noted that it is possible to find the following type of segmentation *RCAprocess* (RCA, process), but this case is less common than the case *RCAProcess*. With a dictionary, it is possible to address this issue. The dictionary is by itself (a part of) the lexical network.

4) Dictionary based strategy: The dictionary based strategy is used in case there is no clue usable for segmentation with the previous strategies, like for example with `studentaddress`. This segmentation needs a dictionary to be effective, the dictionary being in fact some the terms contained in the lexical network (which has been added at bootstrap time with the *aspell* dictionary [Aspell 2008]). Segmenting with a dictionary is based on a prefix and a suffix approches.

Segmentation with prefixes. The identifier is segmented from left to right, reading the string character at a time. We extract the longuest existing substring. Example of segmentation of `studentadress`:

- string: `studentaddress`,
  - `s` exist? yes,
  - `st` exist? yes,
  - `stu` exist? no,
  - `stud` exist? yes,
  - `stude...` exist? no,
- First segment found `student`,
- string left `address`,
  - `a` exist? yes,
  - `ad` exist? yes,
  - `add` exist? yes,
  - `addr` exist? yes,
  - `addre` exist? no,
  - `addres` exist? no,
  - `address` exist? yes,
- Second segment found `address`,
- string left: empty
- Result: (student, address).

Some example of segmentations:

- `localname`: (local, name),
- `trytounderstandthat`: (try, to, understand, that),
- `taxis`: (tax,is).

In the last example, the proper segmentation is in fact (t,axis) because *t* is in this just a prefix of the identifier name. This strategy is not suitable when strings to be segmented are prefixed. To address this issue, we propose a suffix segmentation.

Segmentation with suffixes. This time, we scan the identifier from right to left finding longest suffixes. With the previous example, we obtain with the suffix segmentation the following results:

- `localname`: (local, name),
- `trytounderstandthat`: (try, to, understand, that),
- `taxis`: (t,axis).

Double segmentation. We combine both segmentations by choosing the result with the fewer number of segments. This is not an exact procedure, but in practice for identifiers, it is quite reliable.

B. Labelling

We aim at attaching a POS to each segment, for example for *get, next, warning* we have *verb, adj, noun*. We use *Tree Tagger* by H. Schmid ([Schmid 1994]) for this purpose. Here follows the definition of the tag function:

- Let $T|c$ be the set of valid identifiers,
- Let $W|c$ be the set of valid segments ($W|c \subseteq T|c$),
- Let POS be the set of POS,

$$tag : \bigcup_{i=1}^{\infty}(W|c)^i \rightarrow \bigcup_{i=1}^{\infty}(W|c \times POS)^i \qquad (3)$$

For example: tag: $tag((get, Next, Warning)) = (get, Verb)(Next, Adj)(Warning, Noun))$.

C. Dependency analysis

In practice, we used a simplified set of POS compared to those definied in Tree Tagger[Schmid 1994] for English:

- $Noun = NN \cup NNS \cup NP \cup NPS$,
- $Verb = VV \cup VVP \cup VVZ \cup VVG \cup VVD \cup VVN \cup VB \cup VBP \cup VBZ \cup VBG \cup VBD \cup VBN \cup VH \cup VHP \cup VHZ \cup VHG \cup VHD \cup VHN$,
- $Adj = JJ \cup JJR$,
- $Prep = IN \cup TO$,

After the labelling, we have a list of pairs $(segment, pos)$ for each indentifier. For example, for *getNextWarning* we obtain *[(get,Verb)(next,Adj)(Warning,Noun)]*. The goal of the dependency analysis is to reorganize the segment in function of the dominating order (i.e. finding heads). For example, *[(get,Verb)(next,Adj)(Warning,Noun)]* should be reorganized as *[(get,Verb)(Warning,Noun)(next,Adj)]*. The output is then a list of pairs $(segment, pos)$ ordered by dominating order.

- Let $T|c$ be the set of valid identifiers,
- Let $W|c$ be the set of valid segments ($W|c \subseteq T|c$),
- Let POS be the set of POS,

$$dep : \bigcup_{i=1}^{\infty}(W|c \times POS)^i \rightarrow \bigcup_{i=1}^{\infty}(W|c \times POS)^i \qquad (4)$$

This procedure is based on the POS given to the various segments of the identifier. For example, for *(Verb,Noun)* most probably the verb dominates the noun (example *compute sum*, *add number*, ...). The dependency analysis is done through a rulebased expert system. Rules are ordered by priority (for example, two nouns follow each other, an adjective follows a noun,...). For each rule, an action is defined and applied if the rule activates. Rules are applied iteratively on the pair list $(segment, pos)$, until all pairs are consumed. Generally speaking, such an algorithm becomes complicated to understand as the number of rules grows, making conflicts difficult to resolve, but in the case of identifiers, a small set of rules (between 5 and 10) is enough to compute properly dependency analysis.

The description of the set of rules follows.

1) English Rules Set: Let I the initial list of pairs $(segment, pos)$ and N the new reordered list, initialized to the empty list. Rules are the following (ordered from highest to lowest priority):

1) if I has size 0, then the procedure stops.
2) if I has size 1, then the element of I is added at the end of N and deleted from I.
3) if I has size 2 and the first element is a noun and the second is not a noun, then the first element is added at the end of N and deleted from I.
4) if the first element of l is a verb, it is added at the end of N and deleted from I.
5) if the first element of l is a preposition, it is added at the end of N and deleted from I.
6) if l is composed of elements that are not prepositions, followed by a preposition, followed by anything, l is divided into 3 segments (non prepositions, the preposition, the rest); the result of the application of the rule on the first part is added at the end of N, the preposition is added in N as well as the application of the rules on the rest.
7) if the last element of l is a number, then it is moved to the begining of l.
8) (default rule) the last element of l is inserted at the end of N and deleted form l.

Here follows an example of rule application for the identifier *putPersonInNicePlace*:

1) $I = (put, Verb)(person, Noun)(in, Prep)$ $(nice, Adj)(place, Noun)$, $N = \emptyset$
2) Rule 4 activates (verb in initial position)
3) $I = (person, Noun)(in, Prep)(nice, Adj)$ $(place, Noun)$, $N = (put, Verb)$
4) Rule 6 activates (there a preposition in the middle of the list)
5) I is cut in 3 pieces: $(person, Noun)$; $(in, Prep)$ and $(nice, Adj)(place, Noun)$
6) The result of the applications of the rules on $(person, Noun)$ is added at the end of N
7) Rule 2 activates on $(person, Noun)$ (list of size 1)
8) $N = (put, Verb)(person, Noun)$
9) The preposition is added to N
10) $N = (put, Verb)(person, Noun)(in, Prep)$
11) The result of the applications of the rules on $(nice, Adj)(place, Noun)$ is added at the end of N
12) Rule activates on $(nice, Adj)(place, Noun)$ (default rule), $(place, Noun)$ is added at the end of N
13) $N = (put, Verb)(person, Noun)(in, Prep)$ $(place, Noun)$
14) There is $(nice, Adj)$ left to place, rule 2 activates
15) $N = (put, Verb)(person, Noun)(in, Prep)$ $(place, Noun)(nice, Adj)$

Let us take a second example with the identifier *JavaBlock12*:

1) $I = (Java, Noun), (Block, Noun)(12, CD)$, $N = \emptyset$
2) Rule 7 activates (number in final position). The number is moved to the front.
3) $I = (12, CD)(Java, Noun)(Block, Noun)$, $N = \emptyset$
4) Rule 8 activate (default rule). The rightmost word is moved to the end of N.
5) $I = (12, CD)(Java, Noun)$, $N = (Block, Noun)$
6) Rule 68 activates again.
7) $I = (12, CD)$, $N = (Block, Noun)(Java, Noun)$
8) Rule 2 activates
9) $N = (Block, Noun)(Java, Noun)(12, CD)$

D. Identifying lexical relations

Now, given the strings computed at the previous stage, we try to identify if there is a proper lexical relation between two strings.

- Let $T|c$ be the set of valied identifiers,
- Let $W|c$ be the set of valid segments ($W|c \subseteq T|c$),
- Let POS be the set of POS,
- Let $LEX = \{SYN, HYPO, HYPER, COHYPO, MERO, HOLO\} \cup \emptyset$ the set of lexical relation types,

.

$$rel : (\bigcup_{i=1}^{\infty}(W|c \times POS)^i) \times (\bigcup_{i=1}^{\infty}(W|c \times POS)^i) \to LEX \quad (5)$$

This procedure looks for correspondences between two strings c_1 and c_2. If a correspondance is detected, the name of the lexical relation is returned, otherwise $\emptyset$ is returned. Results of the procedure depend on the set $W|c$. We consider here that this set is defined, and that some occurences of relations do exists (on *syn*, *hypo*, *hyper*, *mero* and *holo*). We have:

$$c_1 = [(w_1^1, pos_1^1), (w_2^1, pos_2^1), \ldots, (w_{n_1}^1, pos_{n_1}^1)] \quad (6)$$

$$c_2 = [(w_1^2, pos_1^2), (w_2^2, pos_2^2), \ldots, (w_{n_2}^2, pos_{n_2}^2)] \quad (7)$$

Let be len the function asosciating to a string its length (for example, $len(c_1) = n_1$ and $len(c_2) = n_2$):

$$len : (\bigcup_{i=1}^{\infty}(W|c \times POS)^i) \times (\bigcup_{i=1}^{\infty}(W|c \times POS)^i) \to N \quad (8)$$

We should remind here that strings are composed of the various segments composing a given identifier, segments being ordered by importance. The discovery of a relation is done in two steps: first, is the lookup of the longest prefix pc_1c_2 between c_1 and c_2.

$$pc_1c_2 = [(w_1^{pcc}, pos_1^{pcc}), (w_2^{scc}, pos_2^{pcc}), \ldots, (w_s^{pcc}, pos_s^{pcc})] \quad (9)$$

such that

$$\forall i \in [1, s], syn(w_i^1, w_i^2) \quad (10)$$

We have then $len(pc_1c_2) = pcc$. Now, for the second step, 4 tests are done to identify the proper lexical relation. The relation type returned corresponds to the first test that passes.

1) if $len(c_1) = len(c_2) = len(sc_1c_2)$, then SYN is returned.
2) if $len(c_1) = len(pc_1c_2)$ and $len(c_1) > 0$, then $HYPER$ is returned.
3) if $len(c_2) = len(pc_1c_2)$ and $len(c_2) > 0$, then $HYPO$ is returned.
4) if $len(c_1) \neq len(pc_1c_2)$ and $len(c_2) \neq len(pc_1c_2)$ and $len(pc_1c_2) > 0$, then $COHYP$ is returned.
5) $\emptyset$ is returned.

Here follow some examples of lexical relation identification:

- Let $c1 = [(car, Noun)]$ and $c2 = [(auto, Noun)]$. Morevover, $syn(car, auto)$ is defined in $W|c$. in that case, $pc_1c_2 = [(car, Noun)]$, because of $syn(car, auto)$ (otherwise $pc_1c_2 = \emptyset$). This fullfills condition 1 and SYN is returned.
- Now, suppose we have $c1 = [(car, Noun)(big, Adj)]$ and $c2 = [(auto, Noun)]$ with $W|c$ as previously. We still have $pc_1c_2 = [(car, Noun)]$. But this time, condition 3 fullfills, thus $HYPO$ is returned.
- Finaly, let be $c1 = [(car, Noun)(big, Adj)]$ and $c2 = [(auto, Noun)(little, Adj)]$ with $W|c$ as previously. We still have $pc_1c_2 = [(car, Noun)]$. But this time, condition 4 fullfills, thus $COHYP$ is returned.

IV. Experiment and Results

We ran our system on a set with thousands of identifiers from various models and software packages (see table I). Those are real models and code (as open software) freely available on the web. We evaluated over 400 identifiers taken randomly by manually executing the chain of processes (segmentation, labelling, dependency analysis, and relation identification).

A. Results for segmentation and labelling

394 identifiers out of 400 were well segmented (0.985 ratio) – some examples have been given previously. 356 identifiers out of 400 were well labelled (0.89 ratio). 48 identifiers well segmented got at least one wrong label. For example, the identifier *ParseResult* got a verb label for *Parse* which is linguistically correct, but the clearly intended meaning was the noun and should have been *ParsingResult*. Such, linguistically inconsistent formation of identifier is in fact quite common.

B. Results for dependency analysis

356 identifiers out of 400 got a proper dependency analysis (0.89 ratio). All well labelled identifiers were correctly analysed.

C. Results for relation identification

We run the relation identification on an *a priori general* context, until we extracted around 200 relations. As picking up randomly two identifers is too time consuming and ineficient, the process was to select one identifier randomly and check for relations all other identifiers having at least one element (from segmentation) in common. in order to get around 200 relation it took a bit more than 4000 tries, which means that less than 5 percent of identifiers having one element in commun may have an insghtfull relation between them. We got an 0.84 ratio for proper relations (168 out of 200). Failure cases are typical of wrong semantic relations, valid in the general cases, but invalid for *computing* context. For example, *getThreadId* and *getStringId* were found synonyms because in the general context *string* and *thread* can be synonyms. In a more specific context, both identifier wouldn't have been identified as synonyms. Interesting and typical results follow (other actual results have been given as examples previsouly):

- LevelImpl syn LevelImplementation because Impl syn Implementation
- (OneArgumentOptionHandler, ShortOptionHandler, MapOptionHandler) $hypo$ OptionHandler $hypo$ Handler
- OneArgumentOptionHandler $cophyp$ ShortOptionHandler $cophyp$ MapOptionHandler
- (ColorStringParser, ShortStringParser) $hypo$ StringParser
- ColorStringParser $cophyp$ ShortStringParser
- getMeaning syn getSense
- getValueList $hyper$ getNumberList as value $hyper$ number
- getBigInteger syn getLargeInt
- ExpandCharArr syn ExpandCharArray syn ExpandCharacterArray
- isErrorLogged syn isErrorConnected

V. Conclusion

Automating the discovery of mappings between schemas, ontologies, documents or models has been thoroughly investigated [Rahm and Bernstein 2001], [Shvaiko 2005]. In the context of Model-Driven Engineering, several approaches for semi-automatic generation of transformations based on mapping have recently been proposed. Mixing NLP techniques and model specification has also been a track followed by some works([Liu et al. 2004] [Ilieva and Ormandjeva 2005]). As for model transformation generation, in [Roser and Bauer 2006], model transformations are generated based on ontological information, but less frequently on lexical ones. The two models are supposed to have their semantics provided by a mapping onto a known ontology. Reasoning on the ontology then allows to generate a model transformation, adapting a bootstrap transformation that is whether automatically generated or existing. When dealing with a lexical network as we do in this paper, the same features apply to both models, and we take advantage to do it jointly.

Working on names, or more generally on identifiers, is an issue for MDE [Caprile et Tonella 1999] [Lawrie et al., 2006] In this paper, we have first modelled possible and useful relations between identifiers in models, inspired from lexical semantics in NLP, with a contextual orientation, and an opportunity to compose relations for model transformation generation. We have thus presented some approaches that may be combine to extract relations form identifiers, by using POS tagging (in English, using Tree Tagger) to retrieve words functions in compound identifiers, and then obvious dependency rules to assign a government role to a given item. The role of each item is crucial in order to assert its position in the hierarchy of identifiers, and to detect relations between words. Naturally, dependency rules are shaped for English since most programming names are english based denominations for attributes, classes or variables.

Experiments conducted so far are very promising and clearly show the benefit that can be leveraged from introducing NLP techniques in the domaine of UML modelling. Possible ongoing tracks in NLP for this research could be the use of mutual information approach (like LSA) applied on models and programs in order to access terms sharing the same context and possibly revealing some not so trival relations between them; this approach has been successfully applied to texts.

ACKNOWLEDGMENT

The authors would like to thank France Telecom R&D (Orange Labs) for their support. (CPRE 5326).

Prog	Classes	Itf	Att	Methods
JSON	172	69	1092	3748
SableCC	358	78	1502	5602
JavaCC	280	74	1943	5364
OpenCloud	187	94	1182	4102
Salome TMF	483	252	4715	11058
JUnit	251	89	1138	4124
NgramJ	244	113	1453	4518
JWNL	269	130	1744	5761
SimMetrics	261	78	1171	4076
Commons CLI	184	69	1153	3877
Args4J	187	72	1143	3812
JSAP	452	183	4236	8949
Choco	546	170	2301	8324
Colt	784	280	5524	13826
JGA	721	266	5638	13554
JScience	342	170	2419	6780
JSci	301	95	1471	5620
Commons Math	340	122	1830	5570
Lucene	473	108	2785	6996
JCommon	679	267	5567	12685
XOM	421	189	1784	6312
Julia	491	211	2364	6513

TABLE I
THE CORPUS OF MODELS AND PACKAGES USED IN OUR EXPERIMENT

REFERENCES

[Aspell 2008] Aspell (2008). Aspell. http ://aspell.net.

[Budanitsky and Hirst 2006] A. Budanitsky and G. Hirst, Evaluating WordNet-based Measures of Lexical Semantic Relatedness, *Computational Linguistics*, vol. 32, no. 1, pp. 1347, 2006.

[Caprile et Tonella 1999] Caprile, B. et Tonella, P. (1999). Nomen est omen : Analyzing the language of function identifiers. In *WCRE*, pages 112-122.

[Duchateau et al., 2007] Duchateau F., Bellahsene Z., Roantree M., Roche M. An Indexing Structure for Automatic Schema Matching *SMDB-ICDE'07: International Workshop on Self-Managing Database Systems*, (2007)

[Falleri et al. 2007] Falleri, J.-R., Arevalo, G., Huchard, M. et Nebut, C. (2007a). *Rapport de tâche 1 du projet ftidm*. http ://www.lirmm.fr/falleri/ftidm/-data/rapports/rapportTache1.pdf.

[Falleri et al. 2007] Falleri, J.-R., Arevalo, G., Huchard, M. et Nebut, C. (2007b). *Rapport de tâche 2 du projet ftidm.* http ://www.lirmm.fr/ falleri/ftidm/- data/rapports/rapportTache2.pdf.

[Falleri et al. 2008] Falleri, J.-R., Lafourcade, M., Prince, V.,Huchard, M. et Nebut, C. (2008a). *Rapport de tâche 3 du projet ftidm.* http ://www.lirmm.fr/ falleri/ ftidm/data/rapports/rapportTache3.pdf.

[Falleri et al., 2008b] 008]Falleri08b Falleri, J.-R., Lafourcade, M., Prince, V., Huchard, M. et Nebut, C. (2008b). *Rapport de tâche 4 du projet ftidm*. http ://www.lirmm.fr/ falleri/ftidm/data/rapports/rapportTache4.pdf.

[Fan et al. 2007] Jung-Wei Fan, Hua Xu and Carol Friedman Using contextual and lexical features to restructure and validate the classification of biomedical concepts *BMC Bioinformatic*s 2007, 8:264

[Ilieva and Ormandjeva 2005] M. Ilieva and O. Ormandjieva, Automatic Transition of Natural Language Software Requirements Specication into Formal Presentation, in Natural Language Processing and Information Systems, ser. Lecture Notes in Computer Science, vol. 3513. Springer Berlin / Heidelberg, 2005, pp. 392397.

[Lawrie et al., 2006] Lawrie, D., Morrell, C., Feild, H. et Binkley, D. (2006). Whats in a name ? a study of identifiers. In *ICPC*, pages 3-12. IEEE Computer Society.

[Liu et al. 2004] D. Liu, K. Subramaniam, A. Eberlein, and B. H. Far, Natural Language Requirements Analysis and Class Model Generation Using UCDA, in *Innovations in Applied Articial Intelligence*, ser. Lecture Notes in Computer Science, vol. 3029. Springer Berlin / Heidelberg, 2004, pp. 295304. [Online]. Available: http://www.springerlink.com/ content/peeghrjy5kmkfrpc

[Miller 1994] Miller, G. A. (1994). Wordnet : A lexical database for english. In HLT. Morgan Kaufmann.

[Montes y Gomez et al., 2007] M. Montes y Gomez, A. Gelbukh, A. Lopez Lopez, R. Baeza-Yates. Flexible Comparison of Conceptual Graphs. Lecture Notes in Computer Science, N 2113, Springer, 2001, pp. 102-111;

[Rahm and Bernstein 2001] Rahm, E., and P. A. Bernstein: A Survey of Approaches to Automatic Schema Matching. VLDB Journal 10(4), 2001.

[Roser and Bauer 2006] B. Roser and S.,Bauer. "An Approach to Automatically Generated Model Transformation Using Ontology Engineering Space *Proceedings of SWESE(2nd International Workshop on Semantic Web Enabled Software Engineering*) 2006.

[Schmid 1994] Schmid, H. (1994). Probabilistic part-of-speech tagging using decision trees. In *Proceedings of International Conference on New Methods in Language Processing,* volume 12. Manchester, UK.

[Shvaiko 2005] Shvaiko, P., Euzenat, J.: A survey of schema-based matching approaches. In: J. Data Semantics IV, Volume 3730 of LNCS. (2005) 146-171

[Vossen 1998] Vossen, P. (1998). Eurowordnet a multilingual database with lexical semantic networks. *Computational Linguistics,* 25(4).