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 class 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 Software 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 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.](image)

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 specific relation occurrences. 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 software design.

B. Relations Definitions For Model Transformation

Here follows the definitions of our relations:

1) Hyperonymy: A term \( t_1 \) is a hypernym of \( t_2 \) 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 \( \text{hyper} \):
   - transitive: if \( \text{hyper}(t_1, t_2) \) and \( \text{hyper}(t_2, t_3) \) then \( \text{hyper}(t_1, t_3) \) (example: vehicle, car, fiat 500)
   - strongly antisymmetric

We can thus consider \( \text{hyper} \) as a strict partial order on \( T \mid c \).

2) Hyponymy: A term \( t_1 \) is a hyponym of \( t_2 \) if it is more specific. For example, dog is an hyponym of animal. Mathematically, hypoo can be also modeled as a binary relation, then we have \( \text{hypo}(\text{dog}, \text{animal}) \). Here, follow some properties of the binary relation \( \text{hypo} \):
   - transitive: (example: labrador, dog, animal)
   - strongly antisymmetric

We can thus consider \( \text{hypo} \) as a strict partial order on \( T \mid c \).

The relations \( \text{hyper} \) and \( \text{hypo} \) are the inverse of each other, thus if \( \text{hyper}(t_1, t_2) \), then \( \text{hypo}(t_2, t_1) \).

3) Synonymy: A term \( t_1 \) is a synonym of \( t_2 \) under the context \( c \) if it is equivalent to \( t_2 \). For example, car is a synonym automobile under the context of transports. Here follow some properties of the binary relation \( \text{sym} \):
   - 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 $t_1$ is a direct hyperonym of $t_2$ 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 $d_{hyper}$:

- strongly antisymmetric

Moreover, we have $d_{hyper}(a, b) \rightarrow hyper(a, b)$.

5) Direct Hyponymy: A term $t_1$ is a direct hyponym of $t_2$ if it is directly more specific. For example, dog is a direct hyponym of animal. Mathematically, $d_{hypo}$ can be also modeled as a binary relation, then we have $d_{hypo}(dog, animal)$. Here follow some properties if the binary relation $d_{hypo}$:

- strongly antisymmetric

Moreover, we have $d_{hypo}(a, b) \rightarrow hypo(a, b)$. The relations $d_{hyper}$ and $d_{hypo}$ are the inverse of each other, thus if $d_{hyper}(t_1, t_2)$, then $d_{hypo}(t_2, t_1)$.

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-hyponym 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”. $d_{1cohypo}(a, b) \leftrightarrow d_{hypo}(a, c) \land d_{hypo}(b, c) \land a \neq b$. As $d_{hypo}$ and $d_{hyper}$ are inverse, we have: $d_{hypo}(a, b) \leftrightarrow d_{hyper}(b, a)$. Then, $d_{hypo}(a, c) \land d_{hypo}(b, c) \land a \neq b \leftrightarrow d_{hyper}(c, a) \land d_{hyper}(c, b) \land a \neq b$. Thus, $d_{cohypo}(a, b) \leftrightarrow d_{hyper}(c, a) \land d_{hyper}(c, b) \land a \neq b$. Here follow some properties of the relation 1cohypo:

- symmetric.

2) $\theta_n$co-hyponymy: The $\theta_n$co-hyponym is a formalization of a generalized co-hyponym.

Two terms $a$ and $b$ are $\theta_n$co-hyponyms if there is common hyperonem $h$ between $a$ and $b$ such as $d_{min}(a, h) \leq \theta$ and $d_{min}(b, h) \leq \theta$, $d_{min}(x, y)$ being the shortest path between two comparable terms for the relation $d_{hypo}$. We call $h$ a $\theta_n$minor of $a$ and $b$. Here follow some properties of the relation 2cohypo:

- symmetric.

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

Here follow an extended version of the 2co-hyponym for an n-tuple $x_1, \ldots, x_n$ argument.

3) $\theta_n$co-hyponymy: $n$ terms $x_1, \ldots, x_n$ are $\theta_n$co-hyponyms if there is a common hyperonem $h$ between $x_1, \ldots, x_n$ such as $\forall i \in [1, n], d_{min}(x_i, h) \leq \theta, d_{min}(x_i, y)$ being the shortest path between two comparable terms for the relation $d_{hypo}$. We call $h$ a $\theta_n$minor of $x_1, \ldots, x_n$. Here follow some properties of the relation $\theta_ncohypo$:

- symmetric.

We can notice that $\theta_ncohypo(x_1, \ldots, x_n) \rightarrow \varphi_{cohypo}(x_1, \ldots, x_n), \varphi \geq \theta$. 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) Meronomy: A term $t_1$ is a meronym of $t_2$ if $t_2$ is semantically part of $t_1$. 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) Holonomy: A term $t_1$ is a holonym of $t_2$ if $t_1$ contains semantically $t_2$. For example, body is a 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 $(t_1, t_2)$, then holo $(t_2, t_1)$.

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 $T|c$ ($\theta_ncohypo$),
- 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:
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: \( \text{(body)} \) and \( \text{(skull, head)} \). It is trivial that \( \text{(body)} \) is an equivalence class, and we do have \( \text{syn(skull, head)} \). As we have moreover \( \text{mero(body, head)} \), from the previous property, we deduce \( \text{mero(body, skull)} \).

E. Importing Lexical Relations

In order to make good use of the rules we have defined above, it is necessary to construct several \emph{initial relation occurrences} between the identifiers of a model. In particular, this is necessary for initial \emph{syn} and \emph{dhyper} occurrences. In fact, other relations occurrences (for \emph{dhyper}, \emph{hyper}, \emph{hypo}, \emph{cohypo}) can be deduced from the former ones. Thus, we consider the initial set of occurrences of \emph{syn} and \emph{dhyper} 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 \emph{relids} application:

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

The application \emph{relids} is defined as follows:

\[
\text{relids} : T|c \times T|c \rightarrow LEX
\] (1)

III. Extracting relations from identifiers: tools and rules

In order to compute the result of the application of \emph{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.

Fig. 3. Illustration of the overall process applied to identifier. At the end two identifiers could be compared for relation identification.
It should be noted that it is possible to find the following type of segmentation \textit{RCAProcess} (RCA, process), but this case is less common than the case \textit{RCAProcess}. With a dictionary, it is possible to address this issue. The dictionary is by itself (a part of) the lexical network.

4) \textit{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 \textit{studentaddress}. This segmentation needs a dictionary to be effective, the dictionary being in fact some of the terms contained in the lexical network (which has been added at bootstrap time with the \textit{aspell} dictionary [Aspell 2008]). Segmenting with a dictionary is based on a prefix and a suffix approaches.

\textbf{Segmentation with prefixes.} The identifier is segmented from left to right, reading the string character at a time. We extract the longest existing substring. Example of segmentation of \textit{studentaddress}:

- \textit{string: studentaddress},
  - \textit{s exist? yes},
  - \textit{st exist? yes},
  - \textit{stud exist? no},
  - \textit{stud exist? yes},
  - \textit{student... exist? no},
- \textit{First segment found student},
- \textit{string left address},
  - \textit{a exist? yes},
  - \textit{ad exist? yes},
  - \textit{addr exist? yes},
  - \textit{addre exist? no},
  - \textit{address exist? no},
  - \textit{address exist? yes},
- \textit{Second segment found address},
- \textit{string left: empty}
- \textit{Result: (student, address)}.

Some example of segmentations:

- \textit{localname: (local, name)},
- \textit{trytounderstandthat: (try, to, understand, that)},
- \textit{taxis: (tax,is)}.

In the last example, the proper segmentation is in fact \textit{taxis} because \textit{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.

\textbf{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:

- \textit{localname: (local, name)},
- \textit{trytounderstandthat: (try, to, understand, that)},
- \textit{taxis: (taxis)}.

\textbf{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.

\textbf{B. Labelling}

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

\[
tag: \bigcup_{i=1}^{\infty} (W[c] \times POS)^i \rightarrow \bigcup_{i=1}^{\infty} (W[c] \times POS)^i\]  

For example: \(tag:\tag((\text{get, Next, Warning})) = (\text{get, Verb})(\text{Next, Adj})(\text{Warning, Noun})\).

\textbf{C. Dependency analysis}

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

- \(Noun = NN \cup NNS \cup NP \cup NPS\),
- \(Verb = VV \cup VVZ \cup VZ \cup VVZ \cup VZD \cup VV \cup VB \cup VBP \cup VBZ \cup VBG \cup VBD \cup VBN \cup VH \cup VHP \cup VH \cup VH \cup VHD \cup VHN\),
- \(Adj = JJ \cup JJR\),
- \(Prep = IN \cup TO\).

After the labelling, we have a list of pairs \((\text{segment, pos})\) for each identifier. For example, for \textit{getNextWarning} we obtain \([\{(\text{get, Verb})\} \cup \{(\text{next, Adj})\}] \cup \{(\text{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, \(\{(\text{get, Verb})\} \cup \{(\text{next, Adj})\}] \cup \{(\text{Warning, Noun})\}\} should be reorganized as \(\{(\text{get, Verb})\} \cup \{(\text{Warning, Noun})\} \cup \{(\text{next, Adj})\}\}]. The output is then a list of pairs \((\text{segment, pos})\) 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,

\[
\text{dep}: \bigcup_{i=1}^{\infty} (W[c] \times POS)^i \rightarrow \bigcup_{i=1}^{\infty} (W[c] \times POS)^i
\]  

This procedure is based on the POS given to the various segments of the identifier. For example, for \((\text{Verb, Noun})\) most probably the verb dominates the noun (example \textit{compute sum, add number, ...}). The dependency analysis is done through a rule-based 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 \((\text{segment, pos})\), 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 $(\text{segment}, \text{pos})$ 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 beginning 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 $\text{putPersonInNicePlace}$:

1) $I = (\text{put}, \text{Verb})(\text{person}, \text{Noun})(\text{in}, \text{Prep}) (\text{nice}, \text{Adj})(\text{place}, \text{Noun}), N = \emptyset$
   
2) Rule 4 activates (verb in initial position)

3) $I = (\text{person}, \text{Noun})(\text{in}, \text{Prep})(\text{nice}, \text{Adj}) (\text{place}, \text{Noun}), N = (\text{put}, \text{Verb})$

4) Rule 6 activates (there a preposition in the middle of the list)

5) $I$ is cut in 3 pieces: $(\text{person}, \text{Noun}); (\text{in}, \text{Prep})$ and $(\text{nice}, \text{Adj})(\text{place}, \text{Noun})$

6) The result of the applications of the rules on $(\text{person}, \text{Noun})$ is added at the end of $N$

7) Rule 2 activates on $(\text{person}, \text{Noun})$ (list of size 1)

8) $N = (\text{put}, \text{Verb})(\text{person}, \text{Noun})$

9) The preposition is added to $N$

10) $N = (\text{put}, \text{Verb})(\text{person}, \text{Noun})(\text{in}, \text{Prep})$

11) The result of the applications of the rules on $(\text{nic}, \text{Adj})(\text{place}, \text{Noun})$ is added at the end of $N$

12) Rule activates on $(\text{nic}, \text{Adj})(\text{place}, \text{Noun})$ (default rule), $(\text{place}, \text{Noun})$ is added at the end of $N$

13) $N = (\text{put}, \text{Verb})(\text{person}, \text{Noun})(\text{in}, \text{Prep}) (\text{place}, \text{Noun})$

14) There is $(\text{nic}, \text{Adj})$ left to place, rule 2 activates

15) $N = (\text{put}, \text{Verb})(\text{person}, \text{Noun})(\text{in}, \text{Prep}) (\text{place}, \text{Noun})(\text{nic}, \text{Adj})$

Let us take a second example with the identifier $\text{JavaBlock12}$:

1) $I = (\text{Java}, \text{Noun}),(\text{Block}, \text{Noun})(12, CD), N = \emptyset$

2) Rule 7 activates (number in final position). The number is moved to the front.

3) $I = (12, CD)(\text{Java}, \text{Noun})(\text{Block}, \text{Noun}), N = \emptyset$

4) Rule 8 activate (default rule). The rightmost word is moved to the end of $N$.

5) $I = (12, CD)(\text{Java}, \text{Noun}),(\text{Block}, \text{Noun})$

6) Rule 68 activates again.

7) $I = (12, CD), N = (\text{Block}, \text{Noun})(\text{Java}, \text{Noun})$

8) Rule 2 activates

9) $N = (\text{Block}, \text{Noun})(\text{Java}, \text{Noun})(12, CD)$

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

\[ \text{rel} : \left( \bigcup_{i=1}^{\infty} (W[c] \times \text{POS})^i \right) \times \left( \bigcup_{i=1}^{\infty} (W[c] \times \text{POS})^i \right) \rightarrow \text{LEX} \] (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 $0$ 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 $\text{syn}$, $\text{hypo}$, $\text{hyper}$, $\text{mero}$ and $\text{holo}$). We have:

\[ c_1 = [(w_1^1, \text{pos}_1^1), (w_2^1, \text{pos}_2^1), \ldots, (w_n^1, \text{pos}_n^1)] \] (6)

\[ c_2 = [(w_1^2, \text{pos}_1^2), (w_2^2, \text{pos}_2^2), \ldots, (w_n^2, \text{pos}_n^2)] \] (7)

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

\[ \text{len} : \left( \bigcup_{i=1}^{\infty} (W[c] \times \text{POS})^i \right) \rightarrow \mathbb{N} \] (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 $c_2$.

\[ pc_1c_2 = [(w_1^{pcc}, \text{pos}_1^{pcc}), (w_2^{pcc}, \text{pos}_2^{pcc}), \ldots, (w_s^{pcc}, \text{pos}_s^{pcc})] \] (9)

such that

\[ \forall i \in [1, s], \text{sym}(w_i^1, w_i^2) \] (10)

We have then $\text{len}(pc_1c_2) = pcc$. 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 \( \text{len}(c_1) = \text{len}(c_2) = \text{len}(s_{c_1}c_2) \), then SYN is returned.
2) if \( \text{len}(c_1) = \text{len}(pc_1c_2) \) and \( \text{len}(c_1) > 0 \), then HYPER is returned.
3) if \( \text{len}(c_2) = \text{len}(pc_1c_2) \) and \( \text{len}(c_2) > 0 \), then HYPO is returned.
4) if \( \text{len}(c_1) \neq \text{len}(pc_1c_2) \) and \( \text{len}(c_2) \neq \text{len}(pc_1c_2) \) and \( \text{len}(pc_2c_2) > 0 \), then COHYP is returned.
5) \( \emptyset \) is returned.

Here follow some examples of lexical relation identification:

- Let \( c_1 = [(\text{car}, \text{Noun})] \) and \( c_2 = [(\text{auto}, \text{Noun})] \). Moreover, \( \text{syn}(\text{car}, \text{auto}) \) is defined in \( W[c \text{ in that case, } pc_1c_2 = [(\text{car}, \text{Noun})] \), because of \( \text{syn}(\text{car}, \text{auto}) \) (otherwise \( pc_1c_2 = \emptyset \)). This fullfills condition 1 and SYN is returned.
- Now, suppose we have \( c_1 = [(\text{car}, \text{Noun})(\text{big}, \text{Adj})] \) and \( c_2 = [(\text{auto}, \text{Noun})] \) with \( W[c \text{ as previously. We still have } pc_1c_2 = [(\text{car}, \text{Noun})] \). But this time, condition 3 fullfills, thus HYPO is returned.
- Finally, let be \( c_1 = [(\text{car}, \text{Noun})(\text{big}, \text{Adj})] \) and \( c_2 = [(\text{auto}, \text{Noun})(\text{little}, \text{Adj})] \) with \( W[c \text{ as previously. We still have } pc_1c_2 = [(\text{car}, \text{Noun})] \). 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 identifiers is too time consuming and inefficient, 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 common may have an insightful 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 previosly):

- 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 domain 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 trivial 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).

REFERENCES


[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 Bioinformatics 2007, 8:264


<table>
<thead>
<tr>
<th>Prog</th>
<th>Classes</th>
<th>Ifl</th>
<th>Att</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSON</td>
<td>172</td>
<td>69</td>
<td>1092</td>
<td>3748</td>
</tr>
<tr>
<td>SableCC</td>
<td>358</td>
<td>78</td>
<td>1502</td>
<td>5602</td>
</tr>
<tr>
<td>JavaCC</td>
<td>280</td>
<td>74</td>
<td>1943</td>
<td>5364</td>
</tr>
<tr>
<td>Openload</td>
<td>187</td>
<td>94</td>
<td>1182</td>
<td>4102</td>
</tr>
<tr>
<td>Salome1MF</td>
<td>483</td>
<td>252</td>
<td>4715</td>
<td>11058</td>
</tr>
<tr>
<td>JUnit</td>
<td>251</td>
<td>89</td>
<td>1138</td>
<td>4124</td>
</tr>
<tr>
<td>Ngranj</td>
<td>244</td>
<td>113</td>
<td>1453</td>
<td>4518</td>
</tr>
<tr>
<td>JWNLI</td>
<td>269</td>
<td>130</td>
<td>1744</td>
<td>5761</td>
</tr>
<tr>
<td>SimMetrics</td>
<td>261</td>
<td>78</td>
<td>1171</td>
<td>4076</td>
</tr>
<tr>
<td>CommonsCL</td>
<td>184</td>
<td>69</td>
<td>1153</td>
<td>3877</td>
</tr>
<tr>
<td>Args+J</td>
<td>187</td>
<td>72</td>
<td>1143</td>
<td>3812</td>
</tr>
<tr>
<td>JSAF</td>
<td>452</td>
<td>183</td>
<td>4236</td>
<td>8949</td>
</tr>
<tr>
<td>Choco</td>
<td>546</td>
<td>170</td>
<td>2301</td>
<td>8324</td>
</tr>
<tr>
<td>Colt</td>
<td>784</td>
<td>280</td>
<td>5524</td>
<td>13826</td>
</tr>
<tr>
<td>JGA</td>
<td>721</td>
<td>266</td>
<td>5638</td>
<td>13554</td>
</tr>
<tr>
<td>JScience</td>
<td>342</td>
<td>179</td>
<td>2419</td>
<td>6780</td>
</tr>
<tr>
<td>JSci</td>
<td>301</td>
<td>95</td>
<td>1471</td>
<td>5620</td>
</tr>
<tr>
<td>CommonsMath</td>
<td>340</td>
<td>122</td>
<td>1830</td>
<td>5570</td>
</tr>
<tr>
<td>Lucene</td>
<td>473</td>
<td>108</td>
<td>2785</td>
<td>6996</td>
</tr>
<tr>
<td>JCommon</td>
<td>679</td>
<td>267</td>
<td>5567</td>
<td>12085</td>
</tr>
<tr>
<td>XOM</td>
<td>421</td>
<td>189</td>
<td>1784</td>
<td>6312</td>
</tr>
<tr>
<td>Julia</td>
<td>491</td>
<td>211</td>
<td>2364</td>
<td>6513</td>
</tr>
</tbody>
</table>

TABLE I

THE CORPUS OF MODELS AND PACKAGES USED IN OUR EXPERIMENT

