model[NL]generation: Natural Language Model Extraction

Lars Ackermann
University of Bayreuth, Germany
Universitaetsstrasse 30
95440 Bayreuth, Germany
+49-921-55-7624
lars.ackermann@uni-bayreuth.de

Bernhard Volz
University of Bayreuth, Germany
Universitaetsstrasse 30
95440 Bayreuth, Germany
+49-921-55-7629
bernhard.volz@uni-bayreuth.de

ABSTRACT
In this paper, we describe a novel approach of extracting models from natural language text sources. This requires linguistic analysis as well as techniques for interpreting and using the analysis results. Our linguistic analysis engine provides feature analysis for a rule-based model element detection. Furthermore, the presented approach enables users to generate domain- and application-specific model element detection rules based on natural language sample sentences. Detection rules also have to be connected to instantiation rules for the respective type of model element. This is done through a highly system-supported mapping step where users are able to choose elements from arbitrary meta models and to connect their properties with functions over natural language sentence parts. As both, the definition and application of detection rules is always a sensitive balancing act between precision and recall, these steps are highly interactive. That is why our current prototype also supports detection rule adaption and iterative rule set completion – always to the level of current need.

General Terms
Algorithms, Documentation, Human Factors, Languages.

Keywords
Modeling Tools, Natural Language Modeling, Information Extraction, Natural Language System Modification.

1. INTRODUCTION
Transferring information into a machine-recognizable form is one of the most time-consuming tasks in IT projects. No matter if one wants to formalize a production process, if a data scheme has to be created or even both, the person in charge usually has to read process instructions, functional specifications, product requirement documents and other human readable documentation.

Domain-specific modeling increases the number of possible target forms since each general purpose language is replaced by multiple domain specific languages (DSLs). The huge amount of different target forms requires an even greater amount of knowledge regarding formalisms, domain knowledge and tool usage.

Combining the previously mentioned issues we can shape the core problem for which the approach below provides possible solutions: How to automatically transform various and already existing natural language sources into highly diverse and domain-specific, interlinked target models using a system which, in respect of domain-dependent knowledge parts, can be extended intuitively. Solving this task would save time, provided the possibility of objectively transforming unstructured information into structured forms or provided at least a possibility for checking models which has been created manually reading descriptions in natural language documents.

Whereas there is rich tool support regarding the modeling task itself, the process of (semi-)automatically transforming informal information, meaning natural language texts, into formal models is most widely more a research topic than an market-ready feature of modeling tools [3].

Since the first steps in Natural Language Processing (NLP) during the 1940s [7] the knowledge about automatically interpreting unstructured natural language texts has grown enormously. Techniques of Information Extraction, Sentiment Analysis, Information Retrieval and others are used to "mine" structured information. These techniques base on more general analysis principles providing syntactic and semantic insights into natural language texts.

Consequently this rich knowledge can also be used to support the modeling process. This paper presents a novel approach for semi-automatically transforming natural language descriptions into formal models using syntactically and semantically enriched grammar rules for model element detection in combination with mapping functions for model element extraction.

The core benefits of this approach particularly relevant to DSM are its meta-model variability, the intuitive extendibility and a training method that enables the system to handle exactly the problems which are relevant but does not require input for any probably occurring issue. Flexibility in terms of "training on demand" is essential since our prototypical system is intended to be trained by its users. The meta-model variability enables the user to use a meta model that fits the current domain. Without learning effort for users and with the possibility for partial training the time exposure for extending the system is less compared to an approach which requires expert knowledge in shaping linguistic detection rules.

2. RELATED WORK
There is a wide variety of approaches for transforming natural language texts into formal models. This is evident from both the large number of target languages and the differences in linguistic knowledge and information extraction steps used. Below we give summaries of the most important work.

In [3] ***Friedrich et al.*** introduce a system for automatically generating process models in Business Process Model and Notation (BPMN) out of natural language process descriptions. The underlying multi-level analysis uses information of sentence level as well as of text level. During both levels a *word model* is synchronized which means holding all currently extracted model elements connected with their text passages.

On sentence level a combination of lexico-syntactic rule set and techniques of text simplification [7], e.g. Anaphora Resolution, is used to identify and extract activity, actor and object candidates. On text level more complex model elements like sequences and gateways are detected using lexical markers and (inter-)sentence dependencies. In a final step on this level descriptions of different aspects of the same model element are merged. In general this approach uses fixed rule sets which can be parameterized by individual word lists.

The ***IBM Research Division*** developed a prototypical system, called Online Analysis Environment (OAE), for transforming industrial use case descriptions into appropriate formal representations. "An Analysis Engine for Dependable Elicitation of Natural Language Use Case Description" [14] describes this approach as a three-phases process. First the text passes pre-processing steps, e.g. tokenization, lemmatization and basic syntactic structure parsing, i.e. Part-Of-Speech-Tagging [7] and identifying syntactic word groups (phrases) as well as identifying grammatical functions of particular sentence parts. This is followed by a dictionary annotation phase where verbs are tagged using semantic word categories, e.g. INPUT, OUTPUT, READ, etc.

The final phase, process building, uses all previously extracted information to perform a pattern matching "over tokens, phrases and already identified patterns" [14]. A pattern can be understood as follows:

- Pattern: `Noun VerbalPhrase` the system.
- Matching sentence: Users contact the system.

Ordering of extracted elements, merging identical elements with different regarded aspects and presenting potential processing error causes are the last steps of the whole process.

There are several other approaches, e.g. [4] and [13] which come up with different grades of convenience, user friendliness and flexibility as well as different target models. In the following section we discuss which requirements a respective system should meet in order to provide a *maximum* of convenience, user friendliness and flexibility.

3. REQUIREMENTS

This section summarizes requirements which shall be fulfilled by a system that is designed to provide support for automated modeling tasks which start with natural language descriptions as source.

Natural Language Interface (NLI). The most intuitive way for users to describe processes, use cases and other objects of interests is to specify all aspects using their respective native language [2][5]. Thus, the NLI should particularly consider the diversity of natural language depending on the individual person and domain. However, according to Li, Dewar and Pooley most of the comparable systems do not take this into account:

"[…] they are limited by the simplicity of input NL expressions and the size of those descriptions (typically <200 words)." [13]

Domain and language independence. In the context of domain specific modeling it is also important to enable the system to capture domain specific terms and expressions whereas the system itself has to be domain independent. This means that the detection algorithm as well as the used linguistic background knowledge must be language independent or at least easily exchangeable in such a way that detection and extraction knowledge specifically tailored to the domain of interest may be created.

At the end this means that there should be *one* system for *all* languages and target models instead of *one* system for *each*.

Transparency of linguistic expert knowledge. Customizable linguistic rule sets or parameterizable machine learning algorithms may perform well but the user usually needs linguistic expert knowledge for adjusting the system's behavior. This would cause an overhead which could annihilate the advantage of the automatic transformation. Therefore, the system should keep the used linguistic knowledge transparent.

Adaptability and Feedback. Many NLP applications require a training phase which often is based on supervised learning techniques. The aim is to learn a *real* function which provides the desired results when it is applied on unseen data. In statistical approaches learning this function is time consuming since it mostly requires a manual task where lots of raw data is annotated with the desired function value. The more complex a function is the more training data is necessary to approximate the real function as close as possible [11]. Mostly the mentioned statistical approaches aim to build an approximation of the full function. Instead, it should be possible to successively train *fragments* according to the *current, domain- and application-specific needs.* This aspect mainly refers to the *completeness* of the detection engine. Another aspect is the possibility to *adjust* the detection engine in the case of error occurrence. In this case, an appropriate solution is to request the user to provide structured feedback and to adjust the configuration of the detection engine considering this feedback.

It is possible to state more requirements but we postpone them as later refinements since we want to focus our key innovations.

4. APPROACH

The following passages describe concepts and algorithms of our prototypical implementation of an interactive and dynamic rule based model extraction system. A main principle is to define formal model element detection rules in natural language. Linguistic analysis tools are used to extract particular aspects of model element descriptions and to enrich plain syntactic structure detection rules with semantic and additional syntactic information. Passage 4.1 describes structure and content of these rules.

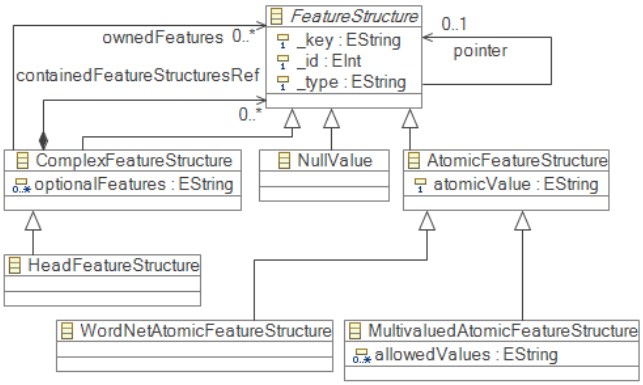

Figure 1. Feature Structure Meta Model

In order to correctly map detected information from sentences to formal model element properties the system includes an interactive mapping phase for newly created detection rules (see subsection 4.4). The algorithm used for applying the rules to arbitrary natural language model descriptions is called Unification [7]. We use a modified concept and implementation – Adaptive Fuzzy Unification.

Another core principle is to avoid defining all rules before being able to use the system. Instead the rules and mappings are defined iteratively according to the current need. This also includes a supervised learning step for correcting falsely failing rules based on user's error feedback, which is considered within subsections 4.3 and 4.5, respectively.

4.1 Templates

Talking about rules hereafter means talking about linguistic templates for sentence structures, contents and dependencies. Syntactic parse trees [7] form the template basis. Linguistic analysis tools are used to detect aspects of model element descriptions and are able to enrich syntactic sentence structure analyses with additional information. Considering the natural language sentence "The manager notifies the customer" the syntactic structure is shown in Figure 2a.

The format the figure shows is a so called Feature Structure (FS) [7] representation. A Feature Structure is a hierarchical composition or, more precisely, a tree of features. Figure 1 shows the meta model for Feature Structures used in our current prototype. Primarily there are two different specializations for Feature Structures: *Complex* FS and *Atomic* FS. These two classes form the hierarchical structure, which can also be seen in Figure 1 since the meta model element *FeatureStructure* shows recursive characteristics. The other specializations are discussed during the following subsections.

(**a**) (**b**)

$$\begin{bmatrix} S & \begin{bmatrix} NP & \begin{bmatrix} DT & The \\ NN & manager \end{bmatrix} \\ VP & \begin{bmatrix} VBZ & notifies \\ NP & \begin{bmatrix} DT & the \\ NN & customer \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \quad \begin{bmatrix} S & \begin{bmatrix} NP & \\ VP & [NP] \end{bmatrix} \end{bmatrix}$$

Figure 2. Left (a): Syntactic Structure of example sentence, Right (b): Reduced Feature Structure

In the case of Figure 2a all keys are syntactic categories, e.g. phrases (S, NP, VP) or Part-Of-Speech (POS) tags[1] (DT, NN, VBZ). Each value is a complex Feature Structure except those below the POS tags. These are atomic values, concretely the plain words. Combining both produces a hierarchical structure which has to be interpreted from left to right where each nesting level refines the previous. This is a valid alternative representation for a linguistic context-free phrase-structure-grammar tree [7].

The rules for detecting model elements are based on such syntactic Feature Structures.

4.2 Reduce and Enrich Feature Structures

Reduce. Using Feature Structures for representing hierarchical syntactic sentence structures provides flexibility. To form a model element detection rule we discard two hierarchy levels from the Feature Structures: the words themselves and their POS tags. The reason for discarding the words is that this level overspecializes the rules to match only the source sentence itself. POS tags are likewise too specific since they limit the detection rule to a very *strict* syntax. Consider the following examples (including POS tags):

- The$_{(DT)}$ manager$_{(NN)}$ contacts$_{(VBZ)}$ the$_{(DT)}$ employees$_{(NNS)}$.$_{(.)}$
- Managers$_{(NNS)}$ contact$_{(VB)}$ the$_{(DT)}$ employees$_{(NNS)}$.$_{(.)}$

In the case of process modeling it is quite likely to instantiate model elements of the same type for both sentences. But if the detection rule contains POS tags the detection would fail because the first sentence contains an article as first word (the second not), a singular noun (plural in second sentence) and a third-person verb form (base-form word in second case). This means that two rules for detecting the same type of model element are required because of, in terms of process modeling, slight modifications. This would significantly increase the number of necessary rules.

Instead only the phrase structure (cf. Figure 2b) is used – which in turn is too general for detecting model elements. For instance it is possible to distinguish between "The manager *notifies* the customer" and "The manager *is* the customer", since they have the same phrase structure.

Ambiguities are the logical consequence. The reduction to phrase structure templates is also problematic because there may be use cases where you *want* to distinguish e.g. between plural and singular nouns, meaning that in some cases the word and/or POS tag level would be useful. Due to Feature Structure's flexibility one has the possibility to solve this problem "organically" by enriching them with additional features.

Enrich. Syntactic structures are, referring to natural language, ambiguous themselves [3]. After reducing the Feature Structures we encounter a lot of *additional* unwanted ambiguities. This is where we make use of semantic and additional syntactic information. From each phrase we select the syntactically most important word, which is called head word or briefly head [12], and perform some feature analyses. The set of extracted features currently consists of:

- LexAbs: Taxonomic abstraction [10]
- SynDep: Syntactic clause dependencies, e.g. `nsubj` or `dobj` [9]

After this analyses each complex Feature Structure, which means each phrase, contains a complex Feature Structure with key *HEAD*. It is *enriched* by inserting all extracted features for the corresponding head word (cf. Figure 3).

$$\begin{bmatrix} S & \begin{bmatrix} NP & \begin{bmatrix} HEAD & \begin{bmatrix} LexAbs & person \\ nsubj & [1] \end{bmatrix} \end{bmatrix} \\ VP & \begin{bmatrix} HEAD[1] & [LexAbs \quad interact] \\ NP & \begin{bmatrix} HEAD & \begin{bmatrix} LexAbs & person \\ dobj & [1] \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 3. Reduced and Enriched Feature Structure

We experienced that these rule modifications achieve disambiguation in many cases. Regarding the two example sentences given above, it is now possible to distinguish between an interaction ("notifies") and some kind of entity specialization ("is"). How to extract these features and how they are used for disambiguation is shown within the next two subsections.

4.3 Training: Feature Structure Composition considering Transparency of Linguistic Knowledge

Figure 3 indicates that detection rules can become very complex, especially for more complex sentences. Defining these rules in their native language, which means writing Feature Structures, implied additional linguistic knowledge and a significant overhead. That is why a principle is used which enables the user to create such rules using natural language.

For each sentence, which shall be detected by some rule, it is possible to find more sentences which describe the *same type* of model element in the *same way*. Considering some of these

[1] See http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html (called up 8/5/2013) for list of specifiers

samples, one can identify common linguistic features as the following examples show:

- The manager notifies the customer.
- An employee meets his colleague.

In both cases there is some kind of interaction between two persons. This knowledge can be reconstructed using the following principles.

Corresponding Heads. The statement that a person interacts with another person has been built by comparing the two example sentences. Concretely some of the *corresponding* words have been compared, which are manager/employee, notifies/meets and customer/colleague. These head word pairs or corresponding heads are used to find commonalities between the sample sentences through feature analysis. Head words are inherited from bottom to the root of the parse tree. In Figure 2a this can be seen considering the entry *HEAD[1]*.

LexAbs. *Person* is a taxonomic abstraction of manager/employee and customer/colleague respectively whereas *interacts* is the generalization of notifies/meets. Using taxonomies, a word classification hierarchy, it is possible to identify these abstractions automatically by searching for the Least Common Subsumer (LCS) [1]. The LCS is the taxonomy term with the greatest distance to the root term and must subsume all given words. If and only if a LCS can be found for some particular set of corresponding heads this feature and its manifestation is declared as *commonality*.

SynDep. Rule definitions like in Figure 3 also include syntactic clause dependencies. If a corresponding head of some particular example sentences participates in a clause dependency and the dependency partner is also included in the complete Feature structure a new Feature Structure is inserted. In this case the value is a *reference* to the dependency partner's Feature Structure. If the dependency partner is not one of the head words the plain word is included. Clause dependencies provide information like "*manager* is the subject referring the predicate *interact*". If and only if these relations exist between the same syntactical partners, i.e. the head words with the same index or lexical equal words, this feature is included as additional commonality describing typed relations between sentence parts.

Resulting Training Process. If the system encounters a sentence it is not able to interpret, this is handled as a so called *seed sentence* and the training process starts. Since the main task of the feature analysis is resolving ambiguities the user has to provide at least one additional sample sentence with the same syntax as the seed sentence and the same type of target model element. Using these sample sentences the system creates a new template with the phrase structure of the samples and performs the mentioned feature analyses. The condition for *including* a feature in the new detection rule is that it has been identified as commonality over all given samples as stated above.

In this way the user only is asked to provide additional natural language sentences for which a template shall be extracted. This keeps the linguistic background knowledge hidden.

The generation of new templates is necessary for detecting sentences with the same features. Now the system is able to *detect* those sentences but it is not able to instantiate the respective model elements since the *mapping* of sentence parts to model element properties and the type to instantiate are unknown.

4.4 Mapping

For instantiating the formal model element with the correct properties, it is necessary to define mapping expressions. Each expression is a key path through a sentence Feature Structure to some contained target Feature Structure. For instance, considering the example in Figure 3 the path *S-NP-HEAD* points to the complex Feature Structure surrounding "person". Depending on the level which is reached by one particular expression the Feature Structure spans one to *n* words where *n* is the number of words in the sentence. Our implementation does not only cover simple "copy and paste" mappings, but also more complex ones such as interpreting natural language strings as values of a certain data type ("not" may be mapped to a Boolean value, e.g. "false") or functions on strings such as concatenation. Each mapping expression is firmly connected with a detection rule and therefore ensures expression evaluation validity. This generic approach enables the system to deal with arbitrary model types if a respective meta model is provided.

4.5 Adaptive Fuzzy Unification

Unification [7] is an algorithm which takes at least two Feature Structures as input, compares them successively and provides a merged Feature Structure as output. The latter is the consensus of all input Feature Structures. Only features which can be reached by the same key path are compared. If their values are equal the feature is included in the output Feature Structure, called *Unifier*.

The Unification algorithm can be used to compare a defined detection rule and a natural language sentence which describes some model element. In order to be able to perform the comparison it is necessary to represent the sentence as Feature Structure, too. This means that the same algorithms and tools are used to syntactically and semantically analyze the respective sentence that have been used to form the detection rules. Since the sentence to analyze is not compared to any other sentences the resulting Feature Structure contains *all* extractable features.

One change which comes into play with the modifier *Fuzzy* is the feature-type sensitive comparison. This means that the algorithm uses feature-type based Unificators. An *AtomicFeatureUnificator* compares atomic features, e.g. clause dependencies between a determiner and a noun. If multiple possible feature manifestations are allowed, an *AtomicMultivaluedFeatureUnificator* can be used to determine if a single feature manifestation of one particular Feature Structure can be found within the possible manifestations of the multivalued Atomic Feature Structure. Finally, the *LexAbsUnificator* compares the lexical abstraction feature by searching for the Least Common Subsumer of all input Feature Structures. If this LCS is a heritage of the rule's LexAbs feature or it is equal to, the Unification succeeds.

If a feature holds a reference as value this reference is resolved and the responsible Unificator evaluates the referenced Feature Structure. Considering Figure 1 this is implemented using the *EReference* that is called *pointer*.

The Unification of two Feature Structures can fail[2] for several reasons – even in cases where it should be successful. By traversing a Feature Structure the algorithm also compares the visited feature key paths of all input Feature Structures. Each divergence is tracked and if no detection rule syntactically fits perfectly the user receives feedback. Otherwise the corresponding mapping expressions are applied. The former case is where the attribute *adaptive* of this

[2] We treat a Unification as unsuccessful if it does not produce a Unifier.

particular Unification algorithm can be justified. Adaptive in this case means that for each input sentence falsely non-fitting rules are modified automatically according to the following principles:

- LexAbs:
  If the rule's lexical abstraction is not general enough the input sentence is used to find a more general LCS.
- Other non-matching atomic features:
  In this case the rule seems to have overspecialized the detection scope. This problem is solved by discarding the interfering feature.
- Additional phrases:
  In some cases the rule was created from natural language sentences with "noise", e.g. an uninformative parenthesis like "of course" or "logically". In cases of additional phrases the rule marks these as optional. Whereas if the rule lacks a particular phrase it is included and marked likewise as optional.

Without claiming for completeness these examples show the general adaptability of the enriched templates.

No matter if adaption has been necessary or if one template directly has matched, a model element has been detected which starts the model element extraction step.

4.6 Model Element Extraction

First of all a model element instance is created using the type information from the associated mapping rule.

Afterwards the model element's properties are set. In Section 4.4 key paths have been introduced. Following all key paths – not through the detection rule but through the new sentence's Feature Structure which already has been created for the Unification step – leads to information which are used as the property's data base. After applying the mapping functions previously mentioned the properties are set. This completes the model element extraction.

5. EVALUATION

The evaluation of the presented approach can be divided into two aspects: qualitative considering all requirements from section 3 and quantitative using manually composed test samples.

5.1 Qualitative: Requirements Compliance

Ref. "Natural Language Interface (NLI)". In general the system is able to transform natural language texts into formal model elements through its ability to extract templates which generalize concrete natural language sentences and its mapping model for model element property extraction.

Ref. "Transparency of linguistic expert knowledge". The presented approach uses user-generated linguistic templates which include syntactic and semantic information for detecting particular model elements. To keep this knowledge hidden the user only is requested to provide sample sentences which describe the same type of model element. Feature analyzers automatically compose the linguistic detection template. Rules for model element instantiation and property setting are described by simply assigning sentence parts to properties.

Ref. "Domain and language independence". Currently the existing prototype is able to automatically analyze English language texts. But each language-dependent part of the system is integrated using an abstraction level which is why these parts can be replaced. Though the Stanford Parser supports other languages, e.g. Chinese and German, it is exchangeable. WordNet is for English only but there are other semantic word nets[3], e.g. Chinese Wordnet, plWordNet, etc. Feature Structure representation and our unification algorithm are completely language-independent.

Since it is possible to provide individual meta models specified in EMF, domain independence is also given.

Ref. "Adaptability and Feedback". In the current state the prototype is able to detect causes for falsely failing template Feature Structures. Since multiple templates could fail for similar minor reasons it is difficult to decide which one is the most appropriate one. That is why the system presents the most probable fitting templates out of which the user may choose the correct one or even create a new template. In the former case the system adapts the existing template by re-analyzing the sample sentences used for extracting (and possibly previously modifying) the current template in combination with the sentence the template falsely missed. This always leads to a more general template which at least is able to detect the new sentence it falsely missed.

5.2 Quantitative: First tests on hand-labeled data

It is difficult to find an appropriate metric for quantitative evaluation of the current prototype, since the system improves iteratively and interactively. One of the core principles is to avoid requesting a complete training set in order to just raise the system to the ability level currently needed. So two stages of a first quantitative evaluation have been applied:

1. Test for disambiguation ability of detection rules in the case of syntactically identical sentences but different model element types,
2. Test of processing experience (complete pipeline).

The first stage considers example process model descriptions from various sources [3]. In order to detect syntactic ambiguities 400 sentences have been syntactically analyzed and grouped by equal parse trees. This identified about 300 different syntax patterns. Three different patterns with a total of 35 sentences have been annotated manually using tags for different model elements. Then a share of sentences with same syntax and same tags have been used to automatically produce a detection rule, respectively. This rule has been applied to all other sentences with the same syntax. Considering precision (about 0.82) and recall (about 0.9) regarding the detected model elements a final F1 score of 0.86 has been calculated. Compared to plain phrase-structure based detection rules this is an improvement of 0.17. Because only two additional feature types have been used the result is encouraging.

Secondly, the complete pipeline has been tested. Since this heavily interactive system hardly can be evaluated by calculating another F1 score we take the percentage of automatically processed sentences at the end of each working cycle. A working cycle means the complete processing of about 200 sentences belonging to two process model descriptions [3]. Afterwards a third description, including about 100 words, has been processed to measure the proportion of automatically processed sentences. The test includes two working cycles where, at the end of the first, 100% of the sentences could be processed automatically. The second, independent cycle showed a proportion of 80% automatically processable sentences. This high accuracy could be achieved since during the processing of the first 200 sentences the set of adaptable detection rules has been updated continuously.

[3] See: http://casta-net.jp/~kuribayashi/multi/ (called up 8/5/2013)

One source of error are incorrect parse trees provided by the Stanford Parser, e.g. it sometimes interprets the third person form of "to contact", means "contacts", as plural noun. A more accurate parser could solve this problem.

Sometimes the system failed to determine a lexical abstraction because of words that are not contained in the lexical knowledge base. E.g. if the corresponding sample head words for rule creation are "Tim" and "manager" there is no entry for "Tim" in WordNet. This could be solved by using a named entity tagger [8].

In summary, the first test results are encouraging. Together with the lack of tools of this type this is a reason for further research.

6. CONCLUSION AND FUTURE WORK

With the presented approach we provide tool support for rule-based transformation of natural language texts to models. Rules can be defined using natural language as well. For that purpose our approach performs automatic linguistic analyses on sample sentences which shall be detected by some particular rule. The resulting rules are represented as Feature Structures. Using Feature Structures we are able to uniquely map features of a particular sentence to an instantiated model element. At least this enables the system to detect and instantiate model elements from an individual meta model as well as setting its properties.

Without performing any training steps the system is not able to extract any model element. The knowledge base is composed "learning by doing". It would be useful if the user is also supported in early states of the knowledge base. This could be achieved using statistical back-off models which provide suggestions for model element extractions. As data basis the text currently to transform could be used. Sentences with similar syntactic and semantic features could be generalized to a template Feature Structure which the user just has to accept or discard.

For several reasons it is recommendable to integrate the solution for transforming natural language texts to models into a modeling platform with particular properties. Since it is possible to choose any meta model as transformation target, it would be helpful if one could define this meta model within the same environment where it is used. This means that the respective platform should provide support for defining modeling languages, models and in the ideal case also model instances. Sometimes it is necessary to be able to simultaneously use different modeling languages, e.g. in business process modeling with special impact of producing and consuming data. Defining an additional combined process and data description language would cause redundancy. If there already are modeling languages for describing processes and data respectively it would avoid this overhead if mutual linking is possible.

In standard cases, e.g. UML-compliant target models or data schemes, the complete meta model may be known. In domain specific cases there may be many individual modeling languages which sometimes are built up while modeling a concrete example. This requires continuous extension of the used modeling language. Usually this entails time-consuming regeneration of parser, linker and other language engine components. Instant usage after language modification without regeneration would save much time in evolutionary meta modeling cases.

Since the prototype presented within this paper is based on Eclipse it is easy to use it as plugin for a modeling platform with all properties described above and which also is an Eclipse application. The Open Meta Modeling Environment (OMME)[4] supports *all* of the desired features mentioned above. Additionally it supports advanced modeling patterns, e.g. Powertypes, Deep Instantiation and multiple meta levels [6]. It provides an automatically language-derived textual concrete syntax as well as it supports defining own graphical notations. Since OMME is built upon Eclipse an integration of the natural language transformation technique discussed in the previous sections is straightforward.

7. References

[1] Baader, F. and Küsters, R. 1998. Computing the least common subsumer and the most specific concept in the presence of cyclic ALN-concept descriptions. In *KI-98: Advances in Artificial Intelligence*, O. Herzog and A. Günter, Eds. LNCS. Springer Berlin Heidelberg, 129–140.

[2] Borman, L. 1985. *Human factors in computing systems. CHI '85 conference proceedings April 14 - 18 San Francisco*. Computer and Human Interaction Conference / Literaturangaben. - Einzelaufnahme eines Zs.-Heftes. Assoc. for Computing Machinery, New York, NY.

[3] Friedrich, F., Mendling, J., and Puhlmann, F. 2011. Process Model Generation from Natural Language Text. In *Advanced Information Systems Engineering*, Eds. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg, 482–496.

[4] Ghose, A., Koliadis, G., and Chueng, A. Process Discovery from Model and Text Artefacts. In *2007 IEEE Congress on Services (Services 2007)*, 167–174.

[5] Hendrix, G. G. 1982. Natural-language interface. *Comput. Linguist.* 8, 2, 56–61.

[6] Jablonski, S., Volz, B., and Dornstauder, S. A Meta Modeling Framework for Domain Specific Process Management. In *2008 32nd Annual IEEE International COMPSAC*, 1011–1016.

[7] Jurafsky, D. and Martin, J. H. 2009. *Speech and language processing. An introduction to natural language processing, computational linguistics, and speech recognition*. Prentice-Hall-series in artificial intelligence. Pearson Education International Prentice Hall

[8] L. Ratinov and D. Roth. 2009. Design Challenges and Misconceptions in Named Entity Recognition. In *CoNLL*.

[9] Marneffe, M.-C. de and Manning, C. D. 2008. The Stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*. CrossParser '08. ACL, Stroudsburg, PA, USA, 1–8.

[10] Marta Fernandez and Caroline Eastman. 1990. Basic Taxonomic Structures and Levels of Abstraction. *Advances in Classification Research Online* 1, 1.

[11] Mohri, M., Rostamizadeh, A., and Talwalkar, A. 2012. *Foundations of machine learning*. Adaptive computation and machine learning. MIT Press, Cambridge, Mass.

[12] Pollard, C. and Sag, I. A. 1994. *Head-driven phrase structure grammar*. Univ. of Chicago Press, Chicago, Ill.

[13] Pooley, L. D. *Object-Oriented Analysis Using Natural Language Processing*. CiteSeerX, DOI=10.1.1.60.1836.

[14] Sinha, A., Paradkar, A., Kumanan, P., and Boguraev, B. A linguistic analysis engine for natural language use case description and its application to dependability analysis in industrial use cases. In *Networks (DSN)*, 327–336.

[4] Project Page: http://www.ai4.uni-bayreuth.de/omme (called up 8/5/2013)