Incremental and Adaptive Software Systems Development of Natural Language Applications

Elena Lloret[†], Santiago Escobar[‡], Manuel Palomar[†], and Isidro Ramos[‡]

Abstract Natural Language (NL) processing tools, such as tokenizers, part-of-speech taggers or syntactic processors obtain knowledge from a set of documents (e.g., tokens, syntactic patterns, etc.) and produce the different elements that will take part on the discourse universe in a NL text (e,g., noun phrases, verbs, sentences, etc). In this paper, we present how NL software systems development can be performed incrementally by using a high-performance specification language like Maude. A generic algebraic specification for NL is defined, including sorts and subsorts apart from equational properties, such as associativity and commutativity for built-in lists and sets. Then, the full discourse universe, available for NL processing, is described in terms of the algebraic specification by providing a non-deterministic but terminating set of transformation rules. Finally, and as a proof of concept, a set of documents for NL processing is given to Maude as an input term and successfully transformed into a proper document, exploring all the non-deterministic possibilities, as well as resolving the ambiguity in language. The main advantages of implementing NL in this manner are: generality, transparency, extensibility, reusability, and maintainability. To the best of our knowledge, this is the first attempt to represent and develop complex NL software systems with this formal notation, and based on the analysis conducted, this implementation constitute the basis for the design and development of more specific NL processing applications, such as text summarization.

[†]Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante, Apdo. de correos, 99, E-03080 Alicante, Spain
e-mail: elloret@dlsi.ua.es, mpalomar@dlsi.ua.es
[‡]Departamentos de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia Valencia, Spain
e-mail: sescobar@dsic.upv.es, iramos@dsic.upv.es

1 Introduction

Software Engineering (SE) is a systematic and disciplined approach to developing software [9]. It applies both computer science and engineering principles and practices to the creation, operation, and maintenance of software systems. Important principles in SE are abstraction, modularity, incremental development, or functional independence, among others [17]. A popular approach is Formal Methods (FM), where a specific notation with a formal semantics, along with automatic and effective tools for reasoning, is used to specify, design, analyze, verify, certify, and even implement software [10].

On the other hand, Natural Language Processing (NLP) is a research area within computer science that concerns with the design and implementation of artificially intelligent systems that are capable of using language as fluently and flexibly as humans do [7]. Intuitively, NLP, as a computer science discipline, should create computer software following a SE methodology, thus, taking into account SE principles. However, when designing and developing NLP systems, SE aspects are generally neglected [13], resulting in highly specific applications that work only for restricted domains that are really difficult to reuse, scale, or adapt. Moreover, such applications heavily depend on hardware and software local specifications, making hard to separate the logic of the system from the computing requirements.

NLP applications could significantly benefit when taking into account SE fundamentals in the design and development of the components. Currently, natural language is modelled differently, depending on several issues, such as the purpose of its application, or the available resources. No attention is paid, however, to the structures selected for its modelling, and the manner in which NLP systems are developed. For this, it should be necessary to study in a more detailed manner, methods and models that could be employed in the software development process that allow the NLP applications to work independently from the environment characteristics, and produce good quality programs in a systematic, cost-effective and efficient way.

An example of these SE techniques could be the Model-Driven Development philosophy [8], which considers models as the main assets in the software development process. Models collect the information that describes the information system at a high abstraction level, allowing the development of the application in an automated way following generative programming techniques [6]. For the Model-Driven Architecture, we choose the high-performance specification language Maude [4]. The FM approach for software specification and implementation is perfect here by providing high-level specification languages for rapid software development and easy-to-check semantics that are also easily extended by repeatedly adding more algebraic properties. This would allow us to process natural language in an incremental manner with: (i) the possibility to extent and adapt the developing systems gradually, (ii) guaranteeing SE principles, such as transparency, maintainability, and reusability of the developed components, and (iii) a high-performance prototype, (iv) providing a formal and easy-to-check semantics.

The main goal of this paper is to study and analyze how natural language can be formally defined and modelled. Our approach follows an algebraic specification approach, via the Maude language [4]. As a result we obtain an implementation consisting of a set of transformation rules that will allow us to process natural language for each context and discourse. Achieving this objective, we will have an independent implementation for natural language that will be highly scalable and reusable, easy to adapt and extend and with the possibility to integrate more knowledge in an easy way through an incremental development. Moreover, this prototype may constitute the basis for developing wider NLP applications, such as text summarization. Specifically, this paper makes three contributions. First, it constitutes one of the first attempts to represent natural language in terms of a formal specification with a formal semantics, bringing NLP and SE/FM together. Second, it introduces the concept of incremental modelling for NLP, and shows how algebraic specification implemented through Maude can offer great advantages to the incremental development of NLP software applications. Third, it discusses how well-known problems of natural language, such as ambiguity, are resolved with this approach.

In the remainder of the paper, we first provide an analysis of previous work that bears some relation to ours (Section 2). Then, we focus on describing our formalization approach for natural language (Section 3) and its implementation (Section 4). Finally we summarize the main conclusions and outline future directions obtained from our research work (Section 5).

2 Related Work

Representation and modelling of natural language has always attracted the attention of NLP researchers. However, there is no consensus in the NLP research community about the form or the existence of a data structure that is able to describe natural language texts [5]. From a NLP perspective, natural language is modelled differently depending on: (i) the level of language analysis performed, and (ii) the specific task and/or application to be tackled.

Over the years, several mechanisms have been employed to represent natural language. Grammars were the ones which experimented a huge progress in the late 1990s, since they were a key element in the development of syntactic parsers. In those years, different types of grammars were developed, including probabilistic context-free grammars (lexicalized or unlexicalized) [12], definite clause grammars [16], or combinatory categorial grammars [19], among others. Most of them were derived from large corpora, such as Penn treebank. However, grammars had well-known limitations [18]. On the one hand, they were too big to be managed, whereas their coverage was somewhat limited. On the other hand, the degree of ambiguity was too great that exhaustive search was not feasible. Other existing approaches that also try to model natural language employ statistical techniques and probability, such as n-grams or bag-of-words [11], but their objective is to obtain knowledge, considering that a text is characterized by such elements, rather than proposing a reusable approach for defining language. Recent efforts have proposed to represent and model the semantics natural language using ontologies [1], as well to establish formal mechanisms (e.g., NLP Interchange Format) to facilitate the interoperability of NLP tools [2]. However, these are still in their early stages.

Platforms such as GATE[1] or UIMA[2] allow the integration of different NLP resources and tools, that were initially developed in a different way, to build complete NLP systems. Our aim greatly differs from the development of these tools. It focuses more on the definition of NLP tasks in an incremental and reusable way, so that complex NLP processes can be built on top of the basic ones with the addition of minimum knowledge.

To the best of knowledge there is not previous research work with the goal of modelling natural language from a SE perspective, and more concretely using FM, and in particular, algebraic specification. By describing natural language in an incremental manner and through a high-performance specification language like Maude, as we proposed in our approach, we have a structure flexible and adaptative enough to be used for developing NLP applications following a standard methodology.

3 Formalization of Natural Language

To set the basis for modelling natural language by means of a formal specification, we can follow one of these approaches: (i) a manual approach, where a linguistic expert defines all necessary knowledge and takes into account all its possible variations; (ii) a fully automatic approach, where existing NLP tools are employed for extracting and obtaining knowledge from a set of documents; and (iii) a hybrid approach, where (i) and (ii) are combined, in the sense that the output of NLP tools is revised by an expert human in order to correct the possible errors made by the automatic processing. Our proposed approach follows this latter approach, relying on the knowledge obtained from automatic NLP tools, and correcting the output of these tools, when necessary, in order to avoid the time-consuming process of (i), and the errors derived from automatic tools in (ii). One important difference is that we explore all possibilities, so returning all possible meanings to a human user for further inspection.

The process of knowledge extraction follows a top-down methodology. Using existing NLP tools that take a document as input and obtain information about their components, we employ such knowledge for describing natural language. Two levels of language analysis are involved in this process: (i) lexical-morphological analysis through a part-of-speech tagger, and (ii) syntactic analysis through a syntactic parser. The former is useful for identifying the types of tokens (i.e. words) included in a document, whereas the second determines the types of syntactic structures.

Then, our framework for describing natural language is as follows. First of all, a generic algebraic specification for NLP is defined, including sorts and subsorts apart of equational properties, such as associativity and commutativity for built-in

[1] http://gate.ac.uk/

[2] uima.apache.org/

lists and sets. This establishes a cornerstone for the assets on generality and extensibility of our incremental natural representation. Then, the full discourse universe, obtained from existing NLP tools and which must be available for processing natural language, is described in terms of the algebraic specification. For each lexical element, a transformation rule is defined. If a lexical element has some ambiguity, several transformation rules would be defined, providing a non-deterministic set of transformation rules. Also, for each syntactic association, a transformation rule is defined but in terms of the lexical elements rather the original text elements. This part is clearly non-deterministic, since multiple combinations of associations may be possible. The addition of extra transformation rules for lexical elements or syntactic associations provides the main assets on transparency, reusability, and maintainability of our incremental natural language processing. Finally, a set of documents for natural language processing is given to Maude as an input term and successively transformed into a proper document, exploring all the non-deterministic possibilities. Since, all the transformation rules are terminating, we know the search space is finite and, thus, manageable by existing formal methods tools, such as a reachability command within the specification language Maude.

4 The Rewriting Logic Semantics of Natural Language

This section contains the technical implementation details for representing natural language in an incremental and adaptive manner, using the Maude language. Maude [3] is a high-performance reflective language and system supporting both equational and rewriting logic specification and programming. Functional modules describe data types and operations on them by means of equational theories. Mathematically, such a theory can be described as a pair $(\Sigma, E \cup A)$, where: Σ is the signature that specifies the type structure (sorts, subsorts, kinds, and overloaded operators); E is the collection of equations declared in the functional module, and A is the collection of equational attributes (associativity, commutativity, and so on) that are declared for the different operators. System modules allow the description of the dynamic behaviour of a system beyond data types and operations on them by allowing system transition rules that constitute a search space graph. Given an initial state and a search pattern, the `search` command performs the search through the rules and equations defined. These equations are, in fact, rules associated to deterministic actions and their application do not generate new states in the search space.

4.1 Incremental Implementation

The cornerstone of our framework is the built-in Maude sort for *Quoted Identifiers* (QID). A QID is denoted by symbol `'` followed by letters and numbers, e.g. `'account123`, and is a wrapper for strings in order to provide a Maude representation for tokens of Maude syntax, which is essential for the meta-representation and meta-reasoning capabilities of Maude. We could have used the built-in Maude sort *String* for representing words but its treatment is much more inefficient than that of QID. The module QID-LIST defines a built-in data type of lists of quoted identifiers, and it is our most important tool because it provides an associative operator `__` (this represents the symbol with empty syntax in Maude) with an identity operator `nil` for an empty list, allowing yuxtaposition of QID's. This is crucial for allowing us to define our types based on sequences of QIDs without having to look for the individual elements in a different data structure. That is, if we need to search for the word `'a` followed by the word `'tree` with zero or more words between them, for example to be replaced by `'a 'dog`, we just have to write the transition rule

```
 rl X:QidList 'a Y:QidList 'tree Z:QidList
=> X:QidList 'a Y:QidList 'dog Z:QidList .
```

in Maude where `X:QidList`, `Y:QidList`, and `Z:QidList` can be substituted by the empty list or any sequence of QID's. This avoids writing code for searching for the word `'a` and then searching for the word `'tree` or restarting the search after finding `'a` but not finding `'tree`, thus easing the way for the specification of NLP.

From the knowledge obtained through the lexical-morphological and syntactic analysis, we begin with the definition of the most basic types, so complex types could be then generated from the basic ones. For the implementation of our prototype, a corpus of 567 English newswire documents provided by DUC[3] was employed. This corpus will constitute the discourse universe for our implementation.

From the part-of-speech tagger output (lexical-morphological analysis), the different categories were grouped into 9 general ones (*determiner*; *pronoun*; *noun*; *preposition*; *adjective*; *adverb*; *verb*; *stopword*; *punctuation mark*). The set of tags from the part-of-speech tagger[4] was automatically mapped to the previously mentioned categories. For instance, the *verb* category would consider tags, such as *VVN*, *VBD*, *VBG*, *VBZ*, or *VBP*. Each of these categories will be a different module in Maude.

For each module, we first define a sort for the category and we specify whether it is also a subsort of another one. All these basic categories are subsort from the QID sort. The next step is to define the signature and semantics for each of the modules. The signature consists of the operations for obtaining the categories. The semantics of each module will contain the equations and rules by means of which this category is built. For this, we use a symbol representing each category, i.e., `pronoun`, `noun`, `adverb`, etc., and the original word is kept as an argument. We could have defined a symbol for each word in each category, e.g. `pronoun-he` or `noun-attack`, but it is easier to implement by keeping the original word as an argument. However, this forces us to make all the category symbols to be frozen for further rewriting (denoted by the labels `[frozen strat (0)]` in Maude), i.e., to avoid nested applications of the transformation rules in the form `'you` $\rightarrow$ `pronoun('you)` $\rightarrow$ `pronoun(pronoun('you))`, leading to an infinite loop.

[3] http://www-nlpir.nist.gov/projects/duc/

[4] http://www.sketchengine.co.uk/tagsets/penn.html

We next provide an example of the signature and fragments of the semantics for the category *noun*.

```
mod NOUN is
 protecting QID .

 sort Noun .
 subsort Qid < Noun .

 op noun : Qid -> Noun [frozen strat (0)] .

 eq ‘advantages = noun(‘advantages) .
 eq ‘adventurers = noun(‘adventurers) .
 eq ‘adventures = noun(‘adventures) .
 eq ‘advice = noun(‘advice) .
 [...]
 rl ‘account => noun(‘account) .
 rl ‘act => noun(‘act) .
 rl ‘aid => noun(‘aid) .
 rl ‘bear => noun(‘bear) .
 [...]
 endm
```

As it can be seen, for this module, as well as for the remaining types (determiner, verb, adverb, adjective, stopword, and puntuation mark), we define that each type can take the form of a QID. The attribute *frozen* is also present in all of them and it is crucial in the definition, in order to ensure that the program terminates; otherwise, we could end up with an infinite loop. In the semantics of each type, the elements of our discourse universe that fall into the corresponding categories are included. Non-ambiguous terms (e.g., `eq 'advice = noun(‘advice) .`) are represented through equations (eq), whereas rules (rl) are used for the ambiguous ones (e.g., `rl 'account => noun(‘account) .`).

Having implemented the basic categories, we exploit the knowledge obtained from the syntactic parser (syntactic analysis) for defining types of language structures that are needed for creating sentences, and documents. In particular, the ones that we intially use for our research are: *noun phrase*, *verb phrase*, which will lead to the definition of the modules *SN*, *SV* in Maude. Below, the implementation of *SN* module is shown.

```
mod SN is
 protecting DETERMINER .
 protecting PRONOUN .
 protecting STOPWORD .
 protecting PREPOSITION .
 protecting ADJECTIVE .
 protecting NOUN .
 protecting QID-LIST .

 sort Sn .
 subsort QidList < Sn .
 op sn : QidList -> Sn [frozen strat (0)] .
```

```
  rl noun(A:Qid) => sn(A:Qid) .
  rl pronoun(A:Qid) => sn(A:Qid) .
  rl determiner(A:Qid) sn(B:QidList) => sn(A:Qid B:QidList) .
  rl preposition(A:Qid) sn(B:QidList) => sn(A:Qid B:QidList) .
  rl stopword(A:Qid) sn(B:QidList) => sn(A:Qid B:QidList) .
  rl adjective(A:Qid) sn(B:QidList) => sn(A:Qid B:QidList) .
  rl sn(A:QidList) sn(B:QidList) => sn(A:QidList B:QidList) .
endm
```

The *SN* module inherits the properties of some of the basic categories, as well as the built-in QID-LIST Maude module. The rules reflected in the definition of this type shows how a noun phrase could be produced. These are recursive rules, that will allow the construction of noun phrases following a recursive and transformational process starting from the basic types of words included in the text. Following an analogously process, but taking into account the elements that would constitute a verb phrase, we define the *SV* type.

With the *SN* and *SV* data types, we can define the signature and semantics for a sentence and a document. We define a sentence as a structure that contains a noun phrase, followed by a verb phrase, and ended with a punctuation mark. In addition, the syntactic analysis carried out also detected as sentences, grammatical structures involving either a noun or a verb phrase followed by a punctuation mark. Finally, we create a module for representing natural language documents. We assume that a document can be formed by a single sentence or a concatenation of them.

It should be immediate to understand for the reader that all the modules are easily extensible and reusable, since all the lexical and syntactical information is defined by transformation rules/equations and all such rules/equations are local to the specific word or token being used, so there is no necessary global data or context information. Furthermore, these modules provide a lot of generality and transparency, since the set of new defined sorts is minimal.

4.2 Analysis and Discussion

Once all the necessary data types via Maude modules were implemented, we conduct an analysis of its performance. In this analysis, we want to check if natural language can be represented through our proposed types. Moreover, we want to check whether our approach would be computationally efficient, even though optimization is out of the scope of this paper.

We conduct a battery of tests in order to evaluate our implementation in a qualitative manner. Next, we provide several representative examples for the different types and discuss their results.

Example 1: Basic categories.

The following examples show cases where individual words are tested in order to check if they belong to the correct category.

```
search ‘happy =>* adjective(Q:Qid) .
```

```
Solution 1 (state 0)
Q:Qid --> ‘happy
No more solutions.

search ‘happy =>* verb(Q:Qid) .
No solution.

search ‘account  =>* verb(Q:QidList) .
Solution 1 (state 2)
Q:QidList --> ‘account
No more solutions.

search ‘account  =>* noun(Q:QidList) .
Solution 1 (state 1)
Q:QidList --> ‘account
No more solutions.
```

As it can be seen, the token “happy” is correctly recognized as an adjective, and not as a verb. It is worth mentioning how ambiguity is tackled at the execution stage. Let’s consider the term *“account”*. This word is ambiguous since it can act as a noun as well as a verb. At the execution time, we perform a search within a particular category, so for all the possible states in the specific category, the program will check whether the word *account* exists. In our examples, when we search this word either in the space of a verb or a noun, the word will be recognized.

Another issue that is also important to note is the number of states visited and the time spent for processing the terms. Maude is really fast, consuming less than 1 second of CPU time. The predominant use of rules and equations makes the program to be faster than if we had used only rules for implementing the semantics for all the types.

Example 2: Language structures for creating sentences

These examples show how natural language structures can be also identified.

```
search ‘the ‘account =>* sn(Q:QidList) .
Solution 1 (state 5)
Q:QidList --> ‘the ‘account
No more solutions.


search
’the ‘dog ‘and ‘the ‘bear =>* sn(Q:QidList) .
Solution 1 (state 22)
Q:QidList --> ‘the ‘dog ‘and ‘the ‘bear
No more solutions.
```

Concerning verb phrases (*SV*), our implementation deals with verb phrase that include prepositional phrases form with a preposition, a determiner and a noun.

```
search
’play ‘with ‘the ‘cat =>* sv(Q:QidList) .
Solution 1 (state 20)
Q:QidList --> ‘play ‘with ‘the ‘cat
No more solutions.
```

Finally, we provide an example in order to ensure that noun phrases are not recognized as verb phrases.

```
search
’the ‘dog ‘and ‘the ‘bear =>* sv(Q:QidList) .
No solution.
```

From all these illustrative examples, we can highlight the fact that although the number of states and rewriting steps have increased with respect to the search performed for the basic categories, the time spent for obtaining the solution is marginal.

Example 3: Sentence and document recognition

Provided that we have the vocabulary employed in our discourse universe, our model for representing natural language will be able to recognize sentences and documents. In these cases, the number of rewriting steps is higher than in the previous examples. The longer and more complex the sentence or the document is, the longer it takes for the system to analyze it. Moreover, for their definition, we only have used rules and not equations, leading to the fact that the search space is higher.

```
search ‘the ‘dog ‘plays ‘with ‘the ‘cat ‘;
   =>* sentence(Q:QidList) .
Solution 1 (state 81)
states: 82
rewrites: 158 in 0ms cpu
Q:QidList --> ‘the ‘dog ‘plays
              ‘with ‘the ‘cat ‘;
No more solutions.
states: 95
rewrites: 191 in 0ms cpu

search ‘nero ‘specializes ‘in ‘sniffing
   ‘out ‘bombs ‘and ‘narcotics ‘;
    =>* sentence(Q:QidList) .
Solution 1 (state 602)
states: 603
rewrites: 1805 in 12ms cpu
Q:QidList --> ‘nero ‘specializes ‘in ‘sniffing
              ‘out ‘bombs ‘and ‘narcotics ‘;
No more solutions.
states: 616
rewrites: 1848 in 12ms cpu

search ‘the ‘dog ‘plays ‘with ‘the ‘cat ‘;
     ‘nero ‘specializes ‘in ‘sniffing
     ‘out ‘bombs ‘and ‘narcotics ‘;
     =>* document(Q:QidList) .
Solution 1 (state 58515)
states: 58516
rewrites: 288541 in 1928ms cpu
Q:QidList --> ‘the ‘dog ‘plays ‘with ‘the
              ‘cat ‘; ‘nero ‘specializes ‘in
              ‘sniffing ‘out ‘bombs ‘and
              ‘narcotics ‘;
No more solutions.
```

```
states: 58549
rewrites: 288737 in 1932ms cpu
```

In this paper, we have focused on the presentation of the framework and further optimizations on the search space have to be considered, which is a typical topic of research in formal methods applied to verification. The key idea is that the transformation rules are terminating and, thus, the search space would always be finite. Although our approach has not been evaluated in a quantitative manner, we have verify the adequacy of our method through a set of representative examples, focusing also on the time spent for processing the different modules. More evaluation examples and their performance analysis can be found in [14].

5 Conclusion and Future Directions

In this paper we modelled natural language following an incremental and adaptive algebraic specification approach via a high-performance language (Maude). As far as we know, this is the first attempt to represent natural language in this manner. NLP tools were employed for obtaining and extracting knowledge from documents, and such knowledge was used to define sorts, subsorts and equational properties. Not only was a novel method to represent natural language proposed, but also our implementation was designed in a way that is highly scalable and reusable, thus taking into consideration SE principles, which are not often paid attention by the NLP research community.

An important advantage of this representation is that it would be very easy to add further knowledge, as well as to extend it to other languages. Furthermore, the development of more complex NLP applications, such as information retrieval or text summarization, could be quite straightforward, given that the necessary language analysis levels are performed.

Taking this preliminary study as a starting point, there are several interesting issues that will guide our research work in the future. In the short-term we plan to define more complex data types, as well as optimizing the performance of our initial prototype. The addition of new knowledge as well as new data types will constitute the basis for the development of complex NLP applications, e.g., text summarization, that we plan to tackle in the long-term. Moreover, we also plan to adopt the ideas for natural language deconstruction proposed in [15] that could benefit our approach by providing a flexible approach and determining which language structures should be modelled depending on the aim of the NLP application.

Acknowledgements This research has been partially funded by the Spanish Government through the project TEXT-MESS 2.0 (TIN2009-13391-C04) and Técnicas de Deconstrucción en la Tecnologías del Lenguaje Humano (TIN2012-31224) and by the Generalitat Valenciana through project PROMETEO (PROMETEO/2009/199). Moreover, S. Escobar has been partially supported by the EU (FEDER) and the Spanish MEC/MICINN under grant TIN 2010-21062-C02-02, and by Generalitat Valenciana PROMETEO2011/052.

References

1. Bateman, J.A., Hois, J., Ross, R., Tenbrink, T.: A linguistic ontology of space for natural language processing. Artificial Intelligence **174**(14), 1027 – 1071 (2010)
2. Chiarcos, C.: A generic formalism to represent linguistic corpora in rdf and owl/dl. In: Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12) (2012)
3. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: The Maude 2.0 system. In: Rewriting Techniques and Applications (RTA 2003), 2706, pp. 76–87 (2003)
4. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L. (eds.): All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic, *Lecture Notes in Computer Science*, vol. 4350 (2007)
5. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. Journal of Machine Learning Research **12**, 2493–2537 (2011)
6. Czarnecki, K., Eisenecker, U.W.: Generative programming: methods, tools, and applications. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (2000)
7. Dale, R., Somers, H.L., Moisl, H. (eds.): Handbook of Natural Language Processing. Marcel Dekker, Inc., New York, NY, USA (2000)
8. Frankel, D.: Model Driven Architecture: Applying MDA to Enterprise Computing. John Wiley & Sons, Inc., New York, NY, USA (2002)
9. Ghezzi, C., Jazayeri, M., Mandrioli, D.: Fundamentals of Software Engineering, 2nd edn. Prentice Hall PTR, Upper Saddle River, NJ, USA (2002)
10. Hinchey, M., Jackson, M., Cousot, P., Cook, B., Bowen, J.P., Margaria, T.: Software engineering and formal methods. Commun. ACM **51**(9), 54–59 (2008)
11. Huang, F., Yates, A., Ahuja, A., Downey, D.: Language models as representations for weakly-supervised nlp tasks. In: Proceedings of the Fifteenth Conference on Computational Natural Language Learning, pp. 125–134 (2011)
12. Klein, D., Manning, C.D.: Accurate unlexicalized parsing. In: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1, pp. 423–430. Association for Computational Linguistics, Stroudsburg, PA, USA (2003)
13. Leidner, J.: Current issues in software engineering for natural language processing. In: Proceedings of the Workshop on Software Engineering and Architecture of Language Technology Systems, pp. 45–50 (2003)
14. Lloret, E., Escobar, S., Palomar, M., Ramos, I.: Natural language modelling using maude. Tech. rep., University of Alicante (2013)
15. Martínez-Barco, P., Ferrández-Rodríguez, A., Tomás, D., Lloret, E., Saquete, E., Llopis, F., Peral, J., Palomar, M., Gmez-Soriano, J.M., Romá, M.T.: Legolang: Técnicas de deconstrucción en la tecnologías del lenguaje humano. Procesamiento de Lenguaje Natural (51) (2013)
16. Pereira, F., Warren, D.: Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks. Artificial Intelligence **13**, 231–278 (1980)
17. Pressman, R.S.: Software Engineering: A Practitioner's Approach, 5th edn. McGraw-Hill Higher Education (2001)
18. Steedman, M.: Some important problems in natural language processing. Tech. rep., University of Edinburgh (2010)
19. Steedman, M., Baldridge, J.: Combinatory Categorial Grammar, pp. 181–224. Wiley-Blackwell (2011)