An XML Architecture for Shallow and Deep Processing

Kiril Simov, Alexander Simov, Petya Osenova
BulTreeBank Project
http://www.BulTreeBank.org
Linguistic Modelling Laboratory, Bulgarian Academy of Sciences
Acad. G. Bonchev St. 25A, 1113 Sofia, Bulgaria
kivs@bultreebank.org, alex@bultreebank.org, petya@bultreebank.org

Abstract

The paper presents a set of XML tools for natural language processing such as regular grammars, constraints, transformations, remove and insert operations. The architecture allows any combinations of the tools depending on the task and the concrete analysis. The main control mechanism is the backtracking which depends on achieving a particular subgoal in the analysis.

The main advantage of the architecture is better control over interleaving of ”sure” steps (the shallow processing) and the ”uncertain” steps (the deep processing). In this way the grammar developers can apply shallow processing not just as a first step, but at any level of processing.

We define shallow processing as a sequence of deterministic analyses and deep processing as a sequence of non-deterministic analyses. Thus the shallow and the deep components can be applied to each language level (morphology, syntax, semantics and pragmatics). The complexity of the processing depends on the complexity of the concrete task, not on the language level.

1 Introduction

The idea of robustness [Stede 2003] has trailed the NLP research into combining the shallow and deep techniques in the so called ‘mixed-depth’ processing [Hirst and Ryan 1992]. There is no strict boundary between the two approaches. After having discovered the advantages and weaknesses of either technique of language analysis, the researchers started to pursue hybrid strategies to solve the variety of tasks. Another priority in NLP is the preference of model-tasks to general and broad applications [Stede 2003]. Thus, different potential scenarios should be envisaged and made executable. Additionally, best practices in the creation of language resources rely on bootstrapping techniques and knowledge-assisting databases (lexicons). This means that platform flexibility, information complexity and non-determinism should be well supported. For the variety of tasks a core set of techniques is also required, such as: handling frequent cases as well as cases with great data impact, analogy mechanisms and incrementality.

In our opinion such a software architecture for supporting shallow and deep processing has to ensure at least the following functionalities: (1) application of the tools in a cascaded way (with possible repetition of the application); (2) context dependent application of the tools; (3) usage of temporary annotation (non-monotonicity); (4) change of the parsing strategy: mixing of top-down and bottom-up parsing; (5) dynamic reordering of the tool applications; (6) possibilities to redraw some of the previously generated analyses in cases of failure. A hybrid strategy for shallow processing covering points from 1 to 5 we have already outlined in [Simov and Osenova 2004]. In this paper we present an extension of the shallow processing architecture towards deep linguistic processing. The new architecture can be considered as an incremental partial processing interleaved with deep processing steps. The main goal is to maximize the deterministic processing and to apply more complex non-deterministic processing only when it is unavoidable. The main application of the architecture for the moment is the construction of a treebank for Bulgarian. This application requires a semi-automatic construction of full syntactic analysis of sentences. This is achieved by using limited (at least with respect to the coverage) language resources and processors. Thus, the main task is to maximize the utility of the available resources in order to meet the requirements of the deep processing. This aim is handled by experimenting with various customizing strategies.

Most of the existing architectures for combining shallow and deep processing offer a central representation of the potential linguistic analyses which are produced by different processors like taggers, partial parsers and deep parsers. Thus these architectures are a media for exchange and combining of the different analyses. One of the prominent hybrid architectures, which provides possibilities for access to XML standoff annotation is WHAT: [Schäfer 2003]. Our approach is compatible with this view because it provides an interface to other programs based on XML as an exchange protocol, but it also provides a complete set of tools for the creation of full analysis inside the system. This is important when there are not enough external processors for a language. In this case it is simple for the developers to use the same software environment for all the processing instead of learning several platforms.

The structure of the paper is as follows: in Section 2 our architecture for shallow and deep processing is outlined. Section 3 describes the XML implementation of this general architecture in the CLaRK system. In Section 4 some applications to Bulgarian are considered with respect to the proposed ideas for combining shallow and deep processing in a dynamic way.

2 General Architecture for Combined Processing

The key ideas behind our XML architecture are in accordance with the annotation schema of the HPSG-based treebank for Bulgarian (BulTreeBank). This annotation schema can be viewed as a combination of shallow and deep processing where the shallow component is used to minimize the possible analyses resulting from the deep processing step. The actual selection of the true analyses is performed on the basis of information supplied by the annotator. Hence, the annotation schema comprises the following steps:

Partial parsing step. This step includes all the processing steps before the application of the HPSG grammar: (1) Sentence extraction from the corpus. (2) Automatic pre-processing. Each sentence needs first to be pre-processed at all the levels, that precede deeper syntactic annotation. These include: Morphosyntactic tagging; Named entity recognition; Part-of-speech disambiguation; Partial parsing. We aim at a result of a 100 % accurate partial parsed sentence. Thus at this level we do not apply the rules that are likely to produce errors in some contexts.

HPSG step. The result from the previous step is encoded into an HPSG compatible representation with respect to the HPSG sort hierarchy. The result is sent to the TRALE System [Meurers et al. 2002]. It takes the partial sentence analyses as an input and evaluates all the attachment possibilities for them. The output is encoded as feature graphs. In these feature graphs the initial status of the phrases is restored, i.e. their representation before the application of the HPSG grammar is still available.

Resolution step. The output feature graphs from the previous step are further processed in the following way: (1) their intersection is calculated; (2) then, on the basis of the differences, a set of constraints over the intersection is introduced; (3) during the actual annotation step, the annotator's task is to extend the intersection to full analysis by adding the missing information. The constraints determine the appropriate extensions and also propagate the information, added by the annotator, in order to minimize the number of the incoming possibilities.

This annotation schema can be considered as an example of the *generate and test* approach to the exploration of the search space in order to find the right analyses. The role of shallow processing is mainly to reduce the initial size of the search space. The resolution step can be considered as a consultation with linguistic knowledge sources. The three steps design was guided by practical solutions but in general we think it can be a sequence of *generate and test* steps where the generate steps are of two kinds. These are called *shallow* and *deep* steps. Thus our architecture comprises two main types of processing mechanisms:

1. **Deterministic.** A processing step which produces a unique analysis within a given context.
2. **Non-deterministic.** A processing step which produces several different analyses within a given context, but only some of them are correct.

We consider the first kind of processing steps as shallow processing because their application is efficient and straightforward. The second kind of processing steps requires some additional source of linguistic knowledge to select the correct analysis from the potential analyses that have been generated. Thus we consider these steps as deep processing.

The main tools of the CLaRK System that we use to implement the two kinds of processing steps are: cascaded regular grammars and constraints. As an approximation of the constituent grammar we use the cascaded regular grammars. For the propagation of the information along the trees we use the constraints in insertion mode. For the selection of the appropriate analysis we use the constraints in validation mode. Note that we use these tools for both processing steps, but in different modes. The deterministic mode applies the tool and stores the result in the current analysis. The non-deterministic mode stores temporarily one of the possible analyses and checks for its validity. If the check is satisfied the system proceeds with the next processing. If the check failed then the next analysis would be selected.

The architecture allows a dynamic interleaving of the processing steps. Thus, after some non-deterministic steps deterministic ones can be applied. The general idea is for the non-deterministic steps to be applied as rarely as possible. This is done in accordance with the idea of dynamic networks of grammars presented in [Simov and Osenova 2004]: a dynamic network of grammars is a set of grammars (or other grammar networks) which apply to different contexts. The set is ordered (including cycles). The application of each grammar in the network depends on the place in the order and the context to which it is applicable. The crucial novelty in the current architecture is that in the previous work we applied the grammars only deterministically and now they are extended to handle some non-deterministic tasks. Thus the same idea about the dynamic nature of application is relevant in the new architecture.

3 Implementation in the CLaRK System

In this section we first describe the basic technologies of the CLaRK System[1] — [Simov et. al. 2001]. Then we describe the definition of macro language comprising tool queries and control operators which is the basis of the implementation of the architecture described above. The backtracking facility is an extension of the definition of the queries.

[1]For the latest version of the system see `http://www.bultreebank.org/clark/index.html`.

CLaRK is an XML-based software system for corpora development. It incorporates several technologies: *XML technology*; *Unicode*; *Regular Grammars*; and *Constraints over XML Documents*.

XML Technology

The XML technology is at the heart of the CLaRK System. It is implemented as a set of utilities for data structuring, manipulation and management. We have chosen the XML technology because of its popularity, its ease of understanding and its already wide use in description of linguistic information. In addition to the XML language [XML 2000] processor itself, we have implemented an XPath language [XPath 1999] engine for navigation in documents and an XSLT engine [XSLT 1999] for transformation of XML documents. We started with basic facilities for creation, editing, storing and querying XML documents and developed further this inventory towards a powerful system for processing not only single XML documents but an integrated set of documents. The main goal of this development is to allow the user to add the desirable semantics to the XML documents. The XPath language is used extensively to direct the processing of the document pointing where to apply a certain tool. It is also used to check whether some conditions are present in a set of documents.

Tokenization

The CLaRK System supports a user-defined hierarchy of tokenizers. At the very basic level the user can define a tokenizer in terms of a set of token types. In this basic tokenizer each token type is defined by a set of UNICODE symbols. Above this basic level tokenizers the user can define other tokenizers for which the token types are defined as regular expressions over the tokens of some other tokenizer, the so called parent tokenizer. For each tokenizer an alphabetical order over the token types is defined. This order is used for operations like the comparison between two tokens, sorting and similar.

Regular Grammars

The regular grammars in CLaRK System [Simov, Kouylekov and Simov 2002] work over token and element values generated from the content of an XML document and they incorporate their results back in the document as XML mark-up. The tokens are determined by the corresponding tokenizer. The element values are defined with the help of XPath expressions, which determine the important information for each element. In the grammars, the token and element values are described by token and element descriptions. These descriptions could contain wildcard symbols and variables. The variables are shared among the token descriptions within a regular expression and can be used for the treatment of phenomena like agreement. The grammars are applied in cascaded manner. The evaluation of the regular expressions, which define the rules, can be guided by the user. We allow the following strategies for evaluation: ‘longest match’, ‘shortest match’ and several backtracking strategies.

Constraints over XML Documents

The constraints that we have implemented in the CLaRK System are generally based on the XPath language (see [Simov, Simov and Kouylekov 2003]). We use XPath expressions to determine some data within one or several XML documents and thus we evaluate some predicates over the data. Generally, there are two modes of using a constraint. In the first mode **validation**, the constraint is used for a validity check, similar to the validity check, which is based on a DTD or an XML schema. In the second mode **insertion**, the constraint is used to support the change of the document to satisfy the constraint. The constraints in the CLaRK System are defined in the following way: `(Selector, Condition, Event, Action)`, where the selector defines to which node(s) in the document the constraint is applicable; the condition defines the state of the document when the constraint is applied. The condition is stated as an XPath expression, which is evaluated with respect to each node, selected by the selector. If the XPath expression is evaluated as true, then the constraint is applied; the event defines when this constraint is checked for application. Such events can be: selection of a menu item, pressing of a key shortcut, an editing command; the action defines the way of the actual constraint application.

Cascaded Processing

The central idea behind the CLaRK System is that every XML document can be seen as a “blackboard” on which different tools write some information, reorder it or delete it. The user can arrange the applications of the different tools (not just regular grammars) to achieve the required processing. This possibility is called **cascaded processing**.

In the CLaRK System most of the tools support a mechanism for describing their settings. On the basis of these descriptions (called *queries*) a tool can be applied only by pointing to a certain description record. Each query contains the states of all settings and options which the corresponding tool has. In other words, each query has all the necessary information for applying the tool without any additional information or user interaction.

For user convenience and debugging purposes the queries themselves are represented in XML format. Within the system they can be treated like ordinary XML documents having their names and DTD assignments. For each kind of queries there is a special DTD included in the distribution package of the system. There the user can see the required structure for an XML document to serve as a query.

Once having this kind of queries there is a special tool for combining and applying them in groups (macros called **multiqueries**). During application the queries are executed successively and the result from an application is an input for the next one. The final result is given by the last query application.

Control Operators

For a better control on the process of applying several queries in one we introduce several conditional operators. These operators can determine the next query for application depending on certain conditions. When a condition for such an operator is satisfied, the execution continues from a location defined in the operator. The mechanism for addressing queries is based on user defined labels. When a condition is not satisfied the operator is ignored and the process continues from the position following the operator. In this way constructions like `IF-THEN-ELSE` and `WHILE-DO` easily can be expressed.

The system supports five types of control operators: `IF (XPath)`: the condition is an XPath expression which is evaluated on the current working document. If the result is a non-empty node-set, non-empty string, positive number or true boolean value the condition is satisfied; `IF NOT (XPath)`: the same kind of condition as the previous one but the result from the evaluation of the XPath expression is negated; `IF CHANGED`: the condition is satisfied if the preceding operation has changed the current working document or has produced a non-empty result document (depending on the operation); `IF NOT CHANGED`: the condition is satisfied if either the previous operation did not change the working document or did not produce a non-empty result; `GOTO`: unconditional changing the execution position.

Each macro defined in the system can have its own query and can be incorporated in another macro. In this way some limited form of subroutine can be implemented.

Backtracking

Two basic tools can be a subject of backtracking: the regular grammars and the constraints in insertion mode. A regular grammar can backtrack when over a given input it can succeed more than once, i.e. producing more than one different analysis. A constraint can backtrack when in a context of application it selects more than one value for insertion. In the case of a grammar the backtracking mechanism works in left to right scanning mechanism enumerating all possible analyses. In the case of a constraint an order over the selected values is imposed and they are inserted in this order.

The backtracking over these basic queries is generalized over composite queries in the following way: (1) for a group of grammars the backtracking mechanism works for each grammar depending on the order of the grammars in the group. Thus, for a given input the first withdrawn analysis is the last one of the last grammar from the group which succeeded. So, if, for example, the first grammar does not produce any acceptable analysis, the second one is tried and so on. Of course, in some cases the result could be a result of analyses of several grammars in the group; (2) for a query containing several constraints the values of the first one are checked before the values of the second one and so on; (3) for a multiquery the backtracking mechanism works over all tools: the grammars, constraints, grammar group and constraints group queries in the order in which they appear in the multiquery.

A backtracking occurs when some of the `IF-THEN` operators or the `GOTO` operators are used with a special label `backtrack` instead of ordinary labels. Then the system withdraws all changes of the document to the point where the last check point for backtracking was saved and an attempt for a new analysis is done. For a better control over the backtracking mechanism backtracking cancellation operator (`CUT`) is implemented in the macro language. It causes deletion of all choice points.

4 Applications to Bulgarian

Here we present a few examples which demonstrate a possible application of the proposed architecture. Assuming the modularity of the linguistic knowledge available to the system we show how different levels of deeper linguistic analyses can be achieved. The language data normally provides conditions for both: deterministic and non-deterministic analyses. Some of them depend on universal criteria, but other are language-dependent. Deterministic ones are true: *always* or *in certain circumstances*. Non-deterministic ones resist unary interpretation due to insufficient knowledge. So, the idea is: knowing the selections of the deterministic steps, to reduce the triggers of non-determinism as much as possible and to reach the best parse/parses. Thus, besides the general principles which restrict all grammatical sentences, we also rely on storing more specific instances of these principles in order to manipulate deterministically concrete cases.

In our application to Bulgarian we consider as deterministic the following pre-processing steps:

1. Always applicable:
   (a) Unary or one-option analyses: morphosyntactic, chunks, one to one mappings from the other lexicons;
   (b) Lexicalized units or multiword expressions;
2. Applicable in certain contexts
   (a) Morphosyntactic disambiguation;
   (b) Chunking of phrases with certain right or left context;
   (c) Assigning dependency relations within chunks. See in [Osenova 2002].

We consider as non-deterministic the non-resolved ambiguities or in other words, overgeneration, on each level: grammatical, lexical, structural, constituent, dependency, semantic. The linguistically rich lexicon is of a crucial importance for resolving the ambiguities. See [Bouma 2001] and [Grover and Lascarides 2001] among others. The main problem remains the parsing failure due to the unavailability or the insufficient coverage of such lexicons. This problem can be handled well with a provisional hierarchy of lexicons, which to supply the information, necessary for the parsing. In BulTreeBank we have several lexicons: morpho-syntactic, valency, named-entity, semantic, lexicon of multi-word and parenthetical expressions.

The idea is to order the lexicons according to the coverage and granularity, i.e. first would come the lexicon with the greatest coverage irrespectively of granularity. Then we should try to derive as much information as possible from the ones with best coverage while the others would be 'helpers' in that task. In our case the morpho-syntactic lexicon has the biggest coverage. Then come the lexicons of named-entities (16 000 items), which provide not only semantic, but also grammatical information. However, the more detailed lexicons suffer from insufficient coverage: the valency lexicon covers only 1000 most frequent verbs and the semantic one - 3000 nouns. The lists of multi-word expressions, parenthetical expressions, introductory expressions have also been compiled from scratch on the corpus language data. Note that (1) the semantic lexicon is incorporated as semantic constraints over the verb's arguments in the valency lexicon and (2) the introductory expressions when being verbs are considered extensions of the valency lexicon. It is clear that parsing would collapse given the availability of the presented above lexicons only. For that reason we are trying to customize the encoded knowledge as much as possible.

As it was mentioned above, the morpho-syntactic lexicon is the most elaborate and thus the most reliable one. Elsewhere we have already discussed the transfer of representation structures from this lexicon to attribute:value encodings [Osenova and Simov 2003]. But here we are concerned with the task how to derive as much linguistic knowledge as possible at this very level. We view the morpho-syntactic lexicon as a proto-valency lexicon. From the grammatical features we have derived some simple subcategorization rules, such as: if the verb is intransitive, then there can occur only one NP, which is the subject; if there is only one masculine noun with a full article inflection, it is the subject of the sentence etc. Also, agreement patterns are explored. The named-entity lexicons being provided with grammatical information according to their headwords, do not disturb the flow of the linguistic information at this level. Thus when there is no support from the valency or semantic dictionary, at least the rough proto-valency level survives. Note that in longer sentences more rules interfere and their management becomes more complex. The processing follows the basic algorithm below:

1. First, all the strings are tokenized; tokens are assigned classes according to the token classification — common words, names, or abbreviations; then they are morphologically tagged and disambiguated (including named-entities). It is done by the morphological tagger and two disambiguators. All multi-words and parenthetical expressions are analyzed as well;

2. If there exists an appropriate frame in the valency dictionary, it is mapped to the corpus;

3. If such a frame does not exist, then the proto-valency level is activated from the morphological lexicon via rules and propagation principles. As an additional source of information the accusative and dative clitics are used because they correspond to the arguments of the verb;

4. If it cannot ensure enough support, then some guessing mechanisms are activated on the base of the context information and the general token classification information is consulted.

Let us consider some example sentences with respect to the possibilities of the proposed architecture.

(1) Dyzhdyt prodylzhavashe da shumi navyn.
rain-the continued to babble outside
The rain continued babbling outside.

The morpho-syntactic annotation and disambiguation is trivial in the example and can be achieved with rules. The only composite chunk is the string ‘da shumi’. The analysis after the first shallow processing steps is:

```
<N>Dyzhdyt</N><V>prodylzhavashe</V><V-da>da shumi</V-da><Adv>navyn</Adv>
```

At this level the valency lexicon is consulted and the information is copied to the chunk level. In this example, this is done for the chunk ‘da shumi’ which receives practically the same valency frame as the verb form ‘shumi’[2]. The chunk ‘da shumi’ receives an intransitive frame specifying only the subject position. The frame for the matrix verb ‘prodalzhavashe’ determines an NP subject and a ‘da-clause’ complement additionally with the information for a control relation between the subject of the matrix verb and the subject of the ’da-clause’.

After the propagation of the information to the chunk top level a set of competing grammars is applied. The order of the grammars corresponds to the order of the realization of the head dependants in our grammar which is as follows: *complements, subjects, adjuncts*. Thus, first the grammar for verbal head-complement phrases is run, but it fails because: (1) the ‘da-clause’ is still not analyzed and there is no way to satisfy the complement requirements of the verb ‘prodalzhavashe’ and (2) the verb form ‘da shumi’ does not take a complement. Then the grammar for verbal head-subject phrases is applied, but it also fails because the complement requirements of the matrix verb are not satisfied. The N ’dyzhdyt’ is not taken as a subject of the ‘da shumi’ because our grammar presumes that in subject to subject control relation the subject of the complement clause cannot be extracted unless the matrix verb is impersonal. So, the subject is indicated by a coreferential *pro* element. However, the next grammar for verbal head-adjunct phrases succeeds and attaches the adverb to the verb form ‘da shumi’. Also a choice point is created, because the adjunct could be attached higher in the tree. The last grammar is applied once more because in general there can be more than one adjunct. For our example it fails and then the grammar for clauses is applied completing the ‘da-clause’ analysis. Then again the same grammar group is applied and in this case the analysis for the matrix verb is completed. It the end, a group of constraints for the control relation is applied. The resulting analysis is as follows:

```
<S>
  <VPS>
   <N id="1">Dyzhdyt</N>
   <VPC>
    <V>prodylzhavashe</V>
     <CLDA>
       <VPA><V-da pro-ss="1">da shumi</V-da><Adv>navyn</Adv></VPA>
     </CLDA>
   </VPC>
  </VPS>
</S>
```

Thus the analysis is complete. In case we want to enumerate all analyses we have to cause backtracking. In this case all the data added after the choice point is removed and an attempt for a different analysis is done. In this case we receive an analysis in which the adverb is attached at the sentential level. We will not present this analysis here.

In case that there is no information from the valency dictionary, the proto-valency information from the grammatical features is triggered. This, of course, raises the number of possible analyses, because

[2]Note that if it was a verb with accusative and/or dative clitics presented, then the valency frame would be changed accordingly.

the provided linguistic information is less constrained. Thus we have the information that both verbs are intransitive and by the guessing rules we come up with the following possibilities: the verb form ‘da shumi’ receives an intransitive valency frame, because it is the second verb and therefore - the governed one. However, the first verb, i.e. the matrix one ‘prodylzhavashe’ has two possibilities: either it receives also an intransitive valency frame, or it receives a valency frame of an intransitive subject-subject control verb as above. In the latter case there are two possible analyses which are identical to the ones mentioned above. In the case when the verb ‘prodylzhavashe’ receives an intransitive frame, the ‘da-clause’ can be only an adjunct to the matrix verb. Because there is no rule which requires the matrix clause and the adjunct clause to share their subjects there are again two possibilities. Thus in this case there are four possible analyses. Some of these analysis can be ruled out if additional preference rules are executed.

(2) Losha shega izigra na DPS zhurnalisticheskata nablyudatelnost.
Bad joke played to DPS journalistic-the watchfulness
Journalisic watchfulness played a practical joke on the Movement of Rights and Freedom

According to the token classification there is one abbreviation and common words. The abbreviation is matched against the appropriate lexicon and receives its extension and grammatical characteristics according to its head word. The result after the morphological and chunk processors is as follows:

```
<NPA>Losha shega</NPA> <V>izigra</V> <PP> na <NPA>DPS</NPA></PP>
<NPA>zhurnalisticheskata nablyudatelnost</NPA>
```

The valency lexicon provides the frame of the verb: izigraya(Subj-NP, DirObj-NP, IndirObj-PP). But here we have to deal also with the word order and to map the correct grammatical roles. Thus, the two NP(Adjunct)s, which are output of the chunking stage, undergo head-dependency transformation. Then, via a special principle of definiteness, the information from the non-head daughters is propagated to the mother nodes. Thus, the first NPA is indefinite while the second (we do not consider here the NPA inside the PP) is definite. Both competing NPAs are singular, so the agreement with the verb is not of a help to us. Thus without additional information we have two possible ananlyses. In order to rule the unlikely analysis we can apply a preference rule which says that in all other circumstances being equal the subject is more likely to be definite and the dependants are more likely to be indefinite.

5 Conclusion

In this paper we presented an architecture for a shallow and deep processing. The processing tools are the same in both kinds of analysis, but with different modes of application. The shallow processing is mainly deterministic and the deep processing includes also possibilities for non-deterministic steps. The architecture allows for a dynamic interleaving of the two kinds of the steps as well as different degree of depth of the used linguistic knowledge. The main processing tools are: (1) cascaded regular grammars arranged in networks, and (2) constraints used for information propagation and validation of the analyses. The architecture is implemented within an XML-based system for corpora development: CLaRK System.

From implementation point of view our future work is connected with the refinement of backtracking strategies and with more efficient ways for storing the linguistic information.

From linguistic point of view we plan to make more reliable integration between the language resources themselves, and between language resources and implementation possibilities.

References

[Bouma 2001] Gosse Bouma. 2001. *Extracting Dependency Frames from Existing Lexical Resources.* In: Proc. of the NAACL Workshop on WordNet and Other Lexical Resources: Applications, Extensions and Customizations, Somerset, NJ, USA.

[Grover and Lascarides 2001] 2001. Claire Grover and Alex Lascarides. *XML-Based Data Preparation for Robust Deep Parsing.* In: Proc. of the Joint EACL-ACL Meeting (ACL-EACL 2001), Toulouse, Frace.

[Hirst and Ryan 1992] Graeme Hirst and Mark Ryan. *Mixed-depth representations for natural language text.* In: Jacobs, Paul S. (editor). Text-basedintelligent systems, Hillsdale, NJ: Lawrence Erlbaum Associates.

[Meurers et al. 2002] Detmar Meurers, Gerald Penn, and Frank Richter. 2002. *A Web-based Instructional Platform for Constraint-Based Grammar Formalisms and Parsing.* In *Proc. of the Effective Tools and Methodologies for Teaching NLP and CL.* ACL. Philadelphia, PA, USA.

[Osenova 2002] Petya Osenova. 2002. *Bulgarian Nominal Chunks and Mapping Strategies for Deeper Syntactic Analyses.* In: *Proc. of The Workshop on Treebanks and Linguistic Theories.* Sozopol, Bulgaria.

[Osenova and Simov 2003] Petya Osenova and Kiril Simov. 2003. *Between Chunk Ideology and Full Parsing Needs.* In: *Proc. of the Shallow Processing of Large Corpora (SProLaC 2003) Workshop.* Lancaster, UK.

[Schäfer 2003] Ulrich Schäfer. 2003. *WHAT: An XSLT-based Infrastructure for the Integration of Natural Language Processing Components.* In: Proc. of HLT-NAACL 2003 Workshop: Software Engineering and Architecture of Language Technology Systems. Edmonton, Alberta, Canada.

[Simov and Osenova 2004] Kiril Simov and Petya Osenova. 2004. *A Hybrid Strategy for Regular Grammar Parsing.* In: Proc. of LREC 2004. Lisbon, Portugal.

[Simov et. al. 2001] Kiril Simov, Zdravko Peev, Milen Kouylekov, Alexander Simov, Marin Dimitrov, Atanas Kiryakov. 2001. *CLaRK - an XML-based System for Corpora Development.* In: Proc. of the Corpus Linguistics 2001 Conference. Lancaster, UK.

[Simov, Kouylekov and Simov 2002] Kiril Simov, Milen Kouylekov, Alexander Simov. *Cascaded Regular Grammars over XML Documents.* In: Proc. of the 2nd Workshop on NLP and XML (NLPXML-2002), Taipei, Taiwan.

[Simov, Simov and Kouylekov 2003] Kiril Simov, Alexander Simov, Milen Kouylekov. *Constraints for Corpora Development and Validation.* In: Proc. of the Corpus Linguistics 2003 Conference. Lancaster, UK.

[Stede 2003] Manfred Stede. 2003. *Shallow - Deep - Robust.* In: G. Willee, B. Schroder, H.-C. Schmitz (eds.): Computerlinguistik -Was geht, was kommt? Computational Lingusitics - Achievements and Perspectives. Sankt Augustin.

[XML 2000] XML. 2000. *Extensible Markup Language (XML) 1.0 (Second Edition).* W3C Recommendation. http://www.w3.org/TR/REC-xml

[XPath 1999] XPath. 1999. *XML Path Language (XPath) version 1.0.* W3C Recommendation. http://www.w3.org/TR/xpath

[XSLT 1999] XSLT. 1999. *XSL Transformations (XSLT). version 1.0.* W3C Recommendation. http://www.w3.org/TR/xslt