A System for A Semi-Automatic Ontology Annotation

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

Abstract

Reliable automatic semantic annotation systems do not exist for many languages. Their creation depends in many respects on construction of gold standard corpora. In this paper we present a system for supporting the semi-automatic construction of such corpora. The general annotation architecture comprises two steps: chunk annotation - identification of the text segment which represents a given concept or a relation in the text; concept selection - a chunk might represent more than one concept depending on the context. Implementation of the annotation architecture requires as a prerequisite the following resources - ontology as a source of concepts and relations for the annotation, terminological lexicons, regular grammars linked to the ontology in such a way that each rule incorporates the potential concepts or relations it could recognize in the text. In most cases the creation of a gold standard corpus annotated with semantic information will require the ontology, the lexicons and the regular grammars to be continually extended on the basis of the actual annotation in the text.

The system is implemented as an extension of CLaRK system which already supports similar functionalities on the basis of regular grammar engine and the constraint engine. In the new implementation these two engines are integrated with each other. The new extension is the access to the ontology engine and the possibilities for writing of new grammar rules and/or context rules (implemented as constraints) in the process of annotation.

1 Introduction

Reliable automatic semantic annotation systems do not exist for many languages. Their creation depends in many respects on construction of gold standard corpora. The prerequisite for the creation of such corpora is the definition of comprehensive annotation guidelines, appropriate stock of semantic information and appropriate tool for supporting the semi-automatic semantic annotation. In this paper we assume that the semantic information is represented in the form of ontology equipped with a lexicon in the given language and an annotation grammar. Then the annotation process follows the two steps: *chunk annotation* - identification of the text segment which represents a given concept or a relation in the text; *concept selection* - a chunk could represent more than one concept or relation depending on the context. In our work we follow the ideas of (Erdmann et al. 2000) that the manual (or semi-automatic) semantic annotation is a cyclic process mixing the actual annotation and the evolution of the ontology. In our case we also include the lexicon and the annotation grammar in the process of the concurrent development.

The process of concurrent development of the semantic annotation, the ontology and the annotation grammar (which encodes the lexicon) requires the following functionalities: *search for a text segment*: this step helps the annotator to determine the exact segment of text which is the carrier of the concept or relation from the ontology[1]; *concept selection*: this step determines which concept/relation to be added to the annotation of the corresponding text segment. In case of non-ambiguity or reliable disambiguation rules, the concept/relation could be added automatically; *ontology evolution*: this step updates the ontology. The reasons for this change might be: (1) new concept/relation is necessary for the annotation of a text segment; (2) an existing concept needs to be changed in order to be more precise; *lexicon/grammar evolution*: update of the lexicon and the grammar is necessary when: (1) there are changes in ontology; (2) there are new expressions for already existing concepts/relations; *annotation evolution*: after changes in the ontology and/or the lexicon/grammar it is necessary to update the previously done annotations. In the imple-

[1] In this paper we will not speculate on the question which kinds of textual segments what kind of ontology information carry.

mentation of these functionalities we follow the requirements for a semantic annotation system as they are stated in (Uren et al. 2006).

The structure of the paper is as follows: in the next section the main parameters of the semantic annotation are described and the related problematic issues are discussed, section 3 presents the basic technologies of the CLaRK System, section 4 discusses the extensions of CLaRK with the new functionalities which to support the semantic annotation. The last section concludes the paper.

2 Parameters of Semantic Annotation

The semantic annotation has become a key ingredients of Semantic Web. There is already a vast quantity of literature and initiatives, which approach this topic from various perspectives. For example, there was SAAW 2006 - the First Semantic Authoring and Annotation Workshop devoted on tools, standards and practise of semantic annotation. We can also point to existing annotation systems for Semantic Web (at: www.ncb.ernet.in/groups/dake/annotate/index.shtml). As it was mentioned above, many systems rely on the availability of gold standard data with semantic annotation (see (Collier and Takeuchi 2002)). For discussion on manual, semi-automatic and automatic annotation systems see in (Reeve and Han 2005). Also in (Reeve and Han 2005) it is said that 'annotation systems require the initial definition of an ontology as well as a knowledge base'. Their knowledge base is the wikipedia. In this requirement list only one thing is not mentioned explicitly, namely the tool, which annotates the text.

For us, the ideal situation for an adequate ontological annotation would be the interaction between the domain ontology, the related terminological lexicons as knowledge resources and grammars as the tool for annotation. The ontology is connected to the lexicons. The lexicon, however, is sparse with regard to the wording. It consists of lexicalized as well as non-lexicalized elements. The non-lexicalized elements can vary syntactically. So, very often we cannot be sure that all possibilities are captured. All the listed elements are mapped to the concepts within the ontology. The grammars reflect the degree of coverage and accuracy of both - the terminological lexicon and the ontological concepts. The three interrelated components need a facility for dynamic changes (additions, deletions, corrections)[2].

Let us consider each of them:

2.1 Domain ontology

The domain ontology consists of both - specific and more general concepts. The specific concepts reflects the domain, the more general concepts relate the domain very specific concepts with the concepts in an upper ontology. As the ontology is language-independent, it relies on the quality of the definitions of concepts and relations rather than the concrete concept/relation naming. Needless to say, some name is needed as a working label to the concept. Usually it is in English as lingua franca. Our approach is as follows: either we select the most representative term naming, or we construct a name of meaningful words that altogether do not form a good expression in the language in question. For the first case let us have the following example: the name 'ASCII' is chosen for a concept which has the definition *Standard 8 bit coding system used in data communications.* Otherwise, in the text there can be more complex expressions, which come under the same concept, such as: 'ASCII code table', 'code table ASCII', etc. For the illustration of the second case we can cite the concept 'BarWithButtons', which is the non-lexicalized variant in English in comparison to 'toolbar'.

Some problems arise from the fact that the definitions, in general, reflect various aspects of the concept, but very often they are either too general, or too specific. The reason for this is that definitions are usually created for human use. In such a case the human users are completing the missing parts on the basis of their own knowledge. Sometimes there are not enough definitions per ambiguous concepts, which leads to availability of beyond-domain interpretations only.

2.2 Lexicons

The main issues that have to be considered when constructing terminological lexicons for a language inde-

[2]Due to the fact that we need to provide some illustrations of our ideas, we will use the domain of Computer Science for non-specialists

pendent ontology are as follows: (1) for some concepts there is no lexicalized term in the language, and (2) some important term in the language has no appropriate concept in the ontology which to represent its meaning. Thus, the entries in the lexicon should be viewed as lists of various wordings of one and the same concept. This approach becomes highly relevant in real search scenarios. Also, it is important for the ontology annotation process. The more terminological expressions mapped to the concepts, the better the annotation coverage. Of course, it is impossible to predict all the wordings which correspond to a concept. For that reason, we first concentrated on the most frequent ones. The generalized structure of the lexicons is as follows: (1) a representative term which constitutes the meaning for all the term wordings within the entry. This representative term usually ensures the mapping to the relevant concept; (2) explanation of the concept meaning in lingua franca (usually it is English, but in fact it might be any natural language); and (3) a set of terms in a given language that have the meaning expressed by the representative term.

2.3 Grammars

As it was mentioned above, the grammars reflect the coverage and precision of the ontology and the related terminological lexicons. We call such grammars *annotation grammars* because they are using for recognition of text chunks that are carriers of the concepts in the ontology. In most cases the grammars are implemented as regular grammars. Since the ontology and the lexicons are not perfect (they under- or overgenerate), the grammars are designed in such a way that they assign all the possible mappings of the found terms to the ontology. Then constraints are used on top of the grammars. The constraints play a twofold role: 1. they help in manual disambiguation of the polysemous concepts within a given context, and 2. they help in manual handling the missing or incorrect concepts. For more details on how the grammar and constraints work see Section 3, and for annotation-oriented application see Section 4.

2.4 Problematic issues in Annotation process

The actual process of annotation showed us several problems. Below we will give a brief overview of them. Recall that examples will be in the domain of Computer Science for non-specialists.

First of all, disambiguation among concepts is needed depending on the context. For example, between 'Connection' as 'A link between two communicating computers' and 'Hyperlink' as 'A link in a document to information within that document or another document'. In English both of them might be called 'link'. In Bulgarian these two concepts also share the same term. Other examples are: 'Display' and 'Screen'; 'Image' and 'Picture'.

Because the domain part of the ontology is often incomplete, some other operations during the annotation are needed: addition, extension, deletion of concepts or their correction. The deletion operation is chosen by the annotator when the concept is assigned to an accidental word, which is not a term in the domain. For example, the word 'sector' in the expression 'cultural sector' does not corresponds to the concept 'disc sector' in the computer science domain. The extension operation is preferred when the suggested concept is too specific. It is typical for multiword expressions. For example, 'systems for personalization' is a complex term, which could not be recognized and hence, was not mapped to the concept. Thus, the mapped term 'system' has to be extended to cover the whole textual segment. There is one more repairing possibility, namely the option of introducing a new candidate concept. This option is activated when a more refined distinction is needed. For example, in the ontology there is the concept 'TableOfContents' with the meaning: the list of contents at the beginning of a book. However, another concept is needed, namely the content of an information object.

Also, it happens that there are spurious concept lists (false ambiguity). For example, the concept 'FormField' and 'Field' are semantically identical. The same goes for 'ComputerLanguage' and 'ComputerProgrammingLanguage'; 'Search' and 'Searching' etc. We should mention here that sometimes this spuriousness on the surface might in fact encode some more detailed relation, which we just decided to ignore. For example, 'Search' might be the functionality, while 'Searching' - the actual process.

Additionally, sometimes the context is not suffice for

a good ambiguity resolution. Then either the ambiguity is preserved, or one of the options is selected by chance.

From the discussion in this section it follows (as a rule) that the steps of creating: semantically annotated corpora, domain ontologies, terminological lexicons and annotation grammars are interconnected, and in many cases the work on one of these resources requires changes in the others. This is not surprising having in mind that these resources reflect the various aspects of the description of the domain. The corpus explicates the occurrences of domain concepts in text and serves as a training material for different machine learning approaches for semantic annotation; the ontology structures formally the domain conceptualization; the grammars mediate between the ontology and the text structure; the lexicon provides the connection to the human user. In order to support the development of a semantically annotated corpus, a system has to provide an integrated support towards the creation of each of these resources and a flexible mechanism for switching between the various tasks. Here we present such a system which extends the functionalities of the CLaRK System to support all the listed above features.

3 The CLaRK System

The implementation of the necessary functionalities discussed above is done by an extension of the CLaRK System[3] — (Simov et. al. 2001). In this section we first describe the basic technologies of the CLaRK System. Then we describe the implementation of the new functionalities. CLaRK is an XML-based software system for corpora development. It incorporates several technologies: *XML technology*; *Unicode*; *Regular Grammars*; and *Constraints over XML Documents*.

3.1 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.

3.2 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.

3.3 Regular Grammars in CLaRK System

The regular grammars in CLaRK System 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 markup (called return markup) (Simov, Kouylekov and Simov 2002). 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 syntactic agreement. The grammars are applied in a cascaded manner. The general idea

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

underlying the cascaded application is that there is a set of regular grammars. The grammars in the set are in a particular order. The input of a given grammar in the set is either the input string, if the grammar is first in the order, or the output string of the previous grammar. The evaluation of the regular expressions that define the rules, can be guided by the user. We allow the following strategies for evaluation: 'longest match', 'shortest match' and several backtracking strategies.

Here is an example, which demonstrates the cascaded application of two grammars. The first grammar consists of the following rule:

```
<np aa="NPns">\w</np> ->
     <("An#"|"Pd@@@sn")>,
     <("Pneo-sn"|"Pfeo-sn")>
```

Here the token description[4] `"An#"` matches all morphosyntactic tags for adjectives of neuter gender, the token description `"Pd@@@sn"` matches all morphosyntactic tags for demonstrative pronouns of neuter gender, singular, the description `"Pneo-sn"` is a morphosyntactic tag for the negative pronoun, neuter gender, singular, and the description `"Pfeo-sn"` is a morphosyntactic tag for the indefinite pronoun, neuter gender, singular. The brackets < and > delimit the element descriptions within the rule. This rule recognizes as a noun phrase each sequence of two elements where the first element has an element value corresponding to an adjective or demonstrative pronoun with appropriate grammatical features, followed by an element with element value corresponding to a negative or an indefinite pronoun. Notice the attribute `aa` of the rule's category. It represents the information that the resulting noun phrase is singular, neuter gender. Let us now suppose that the next grammar aims at the determination of prepositional phrases and it is defined as follows:

```
<pp>\w</pp> -> <"R"><"N#">
```

where `"R"` is the morphosyntactic tag for prepositions. Let us trace the application of the two grammars one after another on the following XML element:

```
<text>
  <w aa="R">s</w>
  <w aa="Ansd">golyamoto</w>
  <w aa="Pneo-sn">nisto</w>
</text>
```

[4]Here # and @ are wildcard symbols.

First, we define the element value for the elements with tag `w` with the XPath expression: "**attribute::aa**". Then the cascaded regular grammar processor calculates the input word for the first grammar: `"<" "R" ">" "<" "Ansd" ">" "<" "Pneo-sn" ">"`. Then the first grammar is applied on this input words and it recognizes the last two elements as a noun phrase. This results in two actions: first, the markup of the rule is incorporated into the original XML document:

```
<text>
  <w aa="R">s</w>
  <np aa="NPns">
      <w aa="Ansd">golyamoto</w>
      <w aa="Pneo-sn">nisto</w>
  </np>
</text>
```

Second, the element value for the new element `<np>` is calculated and it is substituted in the input word of the first grammar. In this way the input word for the second grammar is constructed: `"<" "R" ">" "<" "NPns" ">"`. Then the second grammar is applied on this word and the result is incorporated in the XML document:

```
<text>
  <pp>
     <w aa="R">s</w>
     <np aa="NPns">
         <w aa="Ansd">golyamoto</w>
         <w aa="Pneo-sn">nisto</w>
     </np>
  </pp>
</text>
```

The following rule demonstrates the usage of variables in a rule:

```
<np aa="NP&G&N">\w</np> ->
  (<"A&G&Nd">,<"A&G&Ni">*)?,<"N@&G&Ni">
```

Here `&G` and `&N` are variables whose use will ensure the agreement in gender and number. The variables can take as values arbitrary non-empty strings within a token. Additionally, the user can define a domain for a certain variable (a set of permissible values) and a negative domain (a set of values which are not allowable). In the example above, the domain for variable `&G` can be: `f`, `m` or `n` (standing for feminine, masculine and neuter gender). If no (positive) domain is defined then the variable can have any string, which is not presented in the negative domain, as a value. The rule itself says that an `np` is a sequence of a definite adjective followed by any

number of indefinite adjectives and an indefinite noun. The variable ensures the agreement in gender and number and their values are copied to the resulting annotation for the `np`.

The target of a grammar application is determined by an XPath expression, we call this expression a *target description*. The grammar is applied over the context of the elements described by the target description (selected by the XPath expression). This possibility gives us a flexible way to determine where to apply the grammar depending on the context of the elements. Another mechanism offered by the system is the filtering of the input for the grammar. We are able to hide some elements from the content of the element which the grammar is applied to. In this way, for example, we can hide some parenthetical expressions when they are inside some chunk.

3.4 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.

3.5 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.

3.6 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 ex-

pressed.

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.

4 Combining Grammar and Constraint Tools for Semi-Automatic Semantic Annotation

In order to achieve the necessary functionalities discussed in the previous sections we combine the grammar and constraint tools in a joint tool. The automatic part is ensured by the grammars. When there are no competing concepts per text segment, then we can assume that the appropriate concept was assigned. In case of ambiguities, the constraint module is executed, and the annotator is supposed to take decisions at each choice point.

However, the above described situation is far from ideal. In order to capture the inconsistencies within the ontology and lexicons in the beginning, we also provide some 'slower' annotation possibility. This is when each annotated text segment becomes a choice point, at which the annotator can modify, delete, add. We describe the procedures below more extensively.

The application scenario comprises two main steps as it was discussed above:

Search for a text segment.

An appropriate set of grammars is run in a cascaded manner. The rules in the grammars recognize textual segments related to ontological information. The return markup of the rules reflects the necessary information for the ontological annotation.

Additionally, we extend the return markup with calls to queries of other tools in the system. When such a rule succeeds, then the corresponding query is run for the annotation added to the document by the rule.

Concept selection.

The concept selection is done by the query, which is run when the textual segment is annotated. The main type of queries for concept selection is the constraint queries. The constraints are applied to the new annotation done by the grammars. On the basis of the context they could automatically select a concept/relation for the new textual segment or consult the annotator to select the right choice.

When the annotator is consulted there are some extensions of the possible actions of the user. First, the annotator has access to the possible concepts/relations for the annotation of the textual segment as they are stored in the return markup of the corresponding rule. For each of these possibilities the annotator could receive additional information on the basis of the definitions of the concepts/relations and their position within the ontology. If necessary, the user could select a different concept/relation from the ontology as an annotation for the segment. For example, when a more specific concept/relation is denoted by a segment for a more general concept. For instance, in the text segment 'computer' is used to denote the concept 'notebook'.

Second, the annotator realizes that there is a missing concept/relation. For example, when the ontology contains the concept 'server' for a kind of hardware, but not as a software. Then the user accesses an ontology editor with the help of which the ontology is updated. Such a change causes additional changes in the lexicon aligned to the ontology by adding terms for the new concept/relation. The annotation grammars are changed by construction of new rules or by modification of the existing ones. The modified grammars are compiled before the process of semantic annotation proceeds.

Third, the annotator recognizes that the selected segment is a part of a bigger segment which denotes a different concept/relation. For example, the segment 'computer' is selected, but it is a part of the segment 'portable

computer'. In this case, the annotator needs to change the annotation grammar. The change in the grammar might require also a change in the ontology if the concept/relation for the bigger segment does not exist in the ontology.

After each of these three actions the system proceeds further with the processing of the document by searching for next textual segmentation. In case of grammar change the new version of the grammars is used.

In addition to the above annotation scenario which allows for mutual semantic annotation, ontology development and lexicon/grammar development, we need also to re-annotate the documents that already were annotated with the older versions of the ontology and the annotation grammars. In order to do this, we run the new grammars over these documents in such a way that we keep the old annotation for the places where there were no changes from the previous version of the ontology and the grammars. Thus, the annotator needs to reconsider only the new added changes. The comparison between the previous versions of the ontology and the grammars, and the new versions is done on the basis of the recognized textual segments and the return markup for them.

5 Conclusion

In this paper we presented an architecture for the semantic annotation of XML documents in some domain. The idea was described from both sides of view — linguistic adequacy and implementation. We consider the semantic annotation as based on an ontology. The ontology is the source for the semantic information as well as the relations between relevant terms within text. The ontology information and the terminological lexicons are mediated by cascaded regular grammars. The process of semantic annotation interleaves with ontology/lexicon/grammar evolution. This way of combining the three tasks allows the annotation process also to develop from almost completely manual work towards an effective semi-automatic support module.

6 Acknowledgements

This work has been partially supported by two European projects: LT4eL (Language Technology for eLearning) project of the European Community under the Information Society and Media Directorate, Learning and Cultural Heritage Unit (FP6-027391) and AsIsKnown (A Semantic-Based Knowledge Flow System for the European Home Textiles Industry) project of the European Community under IST-2004-2.4.7 - Semantic-based Knowledge and Content Systems (FP6-028044).

References

N. Collier and K. Takeuchi. *PIA-Core: Semantic Annotation through Example-based Learning*. In: 3rd LREC, Las Palmas, Spain, 29-31 May 2002, pp. 1611-1614.

M. Erdmann and A. Maedche and H. Schnurr and S. Staab. 2000. *From manual to semi-automatic semantic annotation: About ontology-based text annotation tools.* In P. Buitelaar and K. Hasida (eds). Proceedings of the COLING 2000 Workshop on Semantic Annotation and Intelligent Content. Luxembourg.

L. Reeve and H. Han. *Semantic Annotation for Semantic Social Networks Using Community Resources.* In: AIS GIGSEMIS Bulletin, vol. 2 (issue 3 and 4), 2005, pp. 52-55.

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.

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.

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

Victoria S. Uren, Philipp Cimiano, Jos Iria, Siegfried Handschuh, Maria Vargas-Vera, Enrico Motta, Fabio Ciravegna. 2006. *Semantic annotation for knowledge management: Requirements and a survey of the state of the art.* in J. Web Sem. 4(1): 14-28.

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

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

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