CLaRK System: Construction of Treebanksthanks*

Kiril Simov, Alexander Simov, Milen Kouylekov, Krasimira Ivanova
BulTreeBank Project
http://www.BulTreeBank.org
Linguistic Modelling Laboratory, Bulgarian Academy of Sciences
Acad. G. Bonchev St. 25A, 1113 Sofia, Bulgaria
kivs@bgcict.acad.bg, adis_78@dir.bg, mkouylekov@dir.bg, krassy_v@abv.bg

1 Introduction

The creation of a treebank comprises at least the following phases: *design phase*, *implementation phase*, *validation phase*, and *documentation phase*. During the *design phase* one has to determine the linguistic knowledge that will be presented within the treebank and its format. The design phase ends with a style book for the treebank, which is the guidelines for the annotators. During the *implementation phase* one has to determine the sources of information to be used during the annotation process, the architecture of the processing, the tools, the organization of the actual annotation. The sources of information could be text archives, partial grammars, lexicons, general linguistic analyses. The *validation phase* requires consistency checks over the whole treebank. These checks determine the conformance of the treebank with the guidelines of the style book and the agreement between annotators on their decisions in the cases where the style book is too general. The *documentation phase* completes the treebank and enables its usage. Needless to say, these phases are interconnected. To facilitate the whole process of creation of a Bulgarian treebank and for better integration of the above mentioned phases we have chosen the following methodology (described in details in [Simov, Popova and Osenova, 2001] and [Simov et al., 2002a]):

- **Fine-grained description.** The linguistic descriptions in the treebank are aimed to be rather detailed in order to demonstrate the multilayered information flow in the syntactic structure of the sentences. Thus not only constituent labels, but dependency relations, agreement features and grammatical functions have been added.
- **Theory dependency.** On more detailed levels of linguistic granularity it becomes necessary to exploit some theory-dependent components, which ensure the consistency of the representation. We have chosen Head-driven Phrase Structure Grammar (HPSG) (see [Pollard and Sag, 1987] and [Pollard and Sag, 1994]) as an underlying theory of our treebank.
- **Formal representation.** Robustness on the level of linguistic descriptions is achieved by the formal representation of the annotation scheme elements. In our opinion, the notion of formal representation refers to the fact that the annotation scheme has a logical interpretation, which allows for inference and check for inconsistency of the annotated data. Our formal mechanism is a special representation of HPSG grammars and sentence analyses as feature graphs (see [King and Simov, 1998], [Simov, 2001], [Simov et al., 2002b], and [Simov, 2002]) based on a logical formalism for HPSG (see [King, 1989]).

*The work reported here is done within the BulTreeBank project. The project is funded by the Volkswagen Stiftung, Federal Republic of Germany under the Programme *"Cooperation with Natural and Engineering Scientists in Central and Eastern Europe"* contract I/76 887.

- **Partial analyses and predictions.** The detailness of the descriptions, the size of the treebank and the desired high level of quality requires semi-automatic facilities for the annotators during the annotation process. Such a minimization of human labour is achieved mainly by exploiting all the possibilities for providing automatic partial analyses.

In order to implement this methodology and to support the phases of the creation of our treebank we have developed further the CLaRK System ([Simov et. al., 2001]) in order to support specific tasks such as: (1) textual archive processing and maintenance, (2) partial grammar implementation and testing, (3) representation of the annotation scheme, (4) manual annotation, writing of validation theories, (5) reclassification when the annotation scheme is changed. The background for achieving all these tasks is the formal representation of the annotation scheme and the actual annotated sentences. This formal representation is based on the feature graphs. Feature graphs are defined with respect to SRL — a logic for HPSG. Having a representation based on logic we could use the inference mechanisms of the logic in order to support some of the tasks during the creation of the treebank. Because we do not intend to implement a system supporting full inference mechanisms for this logic,we use an already existing system for grammar processing with respect to the logic. This is the TRALE system based on [Götz and Meurers, 1997].

In our work we try to implement in the CLaRK System as many constraints as possible that could be stated in a TRALE grammar. In fact we aim at using the CLaRK System as a front-end for TRALE system in such a way that the user could write, use and debug TRALE grammars within the CLaRK System. Functionality necessary for writing TRALE grammars within CLaRK are also useful during the construction of the treebank. We also use the inference mechanisms of TRALE system in order to support the annotation work.

In the paper we first present the main technologies and tools of the CLaRK System. Then we discuss the connection between the CLaRK and TRALE systems. In the next section we describe an interface between the CLaRK System and TRALE. We demonstrate how this interface could use during syntactic annotation. The last section concludes the paper.

2 Technologies behind the CLaRK System

In this section we describe the basic technologies on which the CLaRK System is built. The material here is taken from the documentation of the system and some of the previous publications on the CLaRK System. CLaRK is an XML-based software system for corpora development. It incorporates several technologies:

- XML technology;
- Unicode;
- Regular Grammars;
- Constraints over XML Documents.

2.1 XML Technology

The XML technology is at the heart of the CLaRK System. It is implemented as a set of utilities for structuring, manipulation and management of data. We have chosen the XML technology because of its popularity, its ease of understanding and its already wide use in description of linguistic information. Besides the XML language (see [XML, 2000]) processor itself, we have implemented an XPath language (see [XPath, 1999]) engine for navigation in documents and an XSLT engine (see [XSLT, 1999]) for transformation of XML documents. The documents in the system are represented as DOM Level1 trees (see [DOM, 1998]). We started with basic facilities for creation, editing, storing and querying of XML

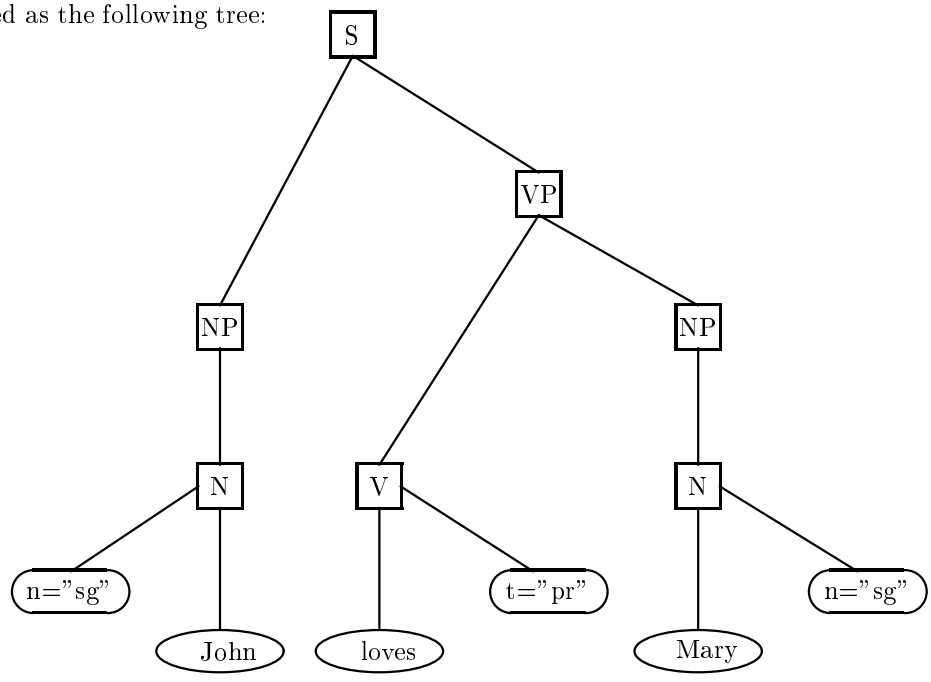
documents and developed further this inventory towards a powerful system for processing not only single XML documents but an integrated set of documents and constraints over them. The main goal of this development is to allow the user to add the desirable semantics to the XML documents.

In the implementation of cascaded regular grammars within the CLaRK System, a crucial role plays the XPath language. XPath is a powerful language for selecting elements from an XML document. The XPath engine considers each XML document as a tree where the nodes of the tree represent the elements of the document, the document's most outer tag is the root of the tree and the children of a node represent the content of the corresponding element. The content nodes can be *element* nodes or *text* nodes. Attributes and their values of each element are represented additionally to the tree.

The XPath language uses the tree-based terminology to point to some direction within the tree. One of the basic notions of the XPath language is the so called *context node*, i.e. a chosen node in the tree. Each expression in the XPath language is evaluated with respect to some context node. The nodes in the tree are categorized with respect to the context node as follows: the nodes immediately under the context node are called *children* of the context node; all nodes under the context node are called *descendant* nodes of the context node; the node immediately above the context node is called *parent* of the context node; all nodes that are above the context node are called *ancestor* nodes of the context node; the nodes that have the same parent as the context node are called *sibling* nodes of the context node; siblings of the context node are divided into two types: *preceding siblings* and *following siblings*, depending on their order with respect to the context node in the content of their parent - if the sibling is before the context node, then it is a preceding sibling, otherwise it is a following sibling. Attribute nodes are added with the context node as *attribute* nodes and they are not children, descendant, parent or ancestor nodes of the context node. The context node with respect to itself is called *self* node. For instance, the following XML document:

```
<S>
   <NP><N n="sg"> John </N></NP>
   <VP>
         <V t="pr"> loves </V>
         <NP><N n="sg"> Mary </N></NP>
   </VP>
</S>
```

is represented as the following tree:



In this picture the rectangles correspond to elements in the XML document and they are called *element nodes*, the ovals represent the text elements in the document and they are called *text nodes*, and the rounded rectangles contain the attribute-value pairs and they are called *attribute nodes*. Each rectangle contains the tag of the corresponding element, each text node contains the text from the content of the corresponding element in the document. Each attribute node is attached to the element that contains it. The root of the tree corresponds to the element containing the whole document. The immediate descendant nodes of a given node N represent the content of the node N. The attribute nodes are considered in a different way and they are connected with the corresponding nodes in a different dimension. Thus the attribute nodes are not descendant nodes of any node in the tree.

The other language from XML world implemented in the CLaRK System is XSL(Transformations). At the moment it is partially implemented, but the full language is under implementation. Each XSLT transformation is an XML document in which some of the elements have a predetermined procedural meaning. Each XSL Transformation can be applied to an XML document which is transformed in another XML document (within the CLaRK System). One extension within the CLaRK System is that an XSL Transformation could be applied locally to some elements and their content. The elements can be selected by an XPath expression. In this way the user can specify on which elements a given transformation to be applied.

2.2 Tokenization

XML considers the content of each text element a whole string that is usually unacceptable for corpus processing. For this reason it is required for the wordforms, punctuation and other tokens in the text to be distinguished. In order to solve this problem, 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.

2.3 Regular Grammars

The basic CLaRK mechanism for linguistic processing of text corpora is the regular grammar processor. The application of the regular grammars to XML documents is connected with the following problems:

- how to treat the XML document as an input word for a regular grammar;
- how the return-up grammar category to be incorporated into the XML document; and
- what kind of 'letters' to be used in the regular expressions so that they correspond to the 'letters' in the XML document.

The solutions to these problems are described in the next paragraphs.

First of all, we accept that each grammar works on the content of an element in an XML document. The content of each XML element (excluding the EMPTY elements on which regular grammar cannot be applied) is either a sequence of XML elements, or text, or both. The input word for a grammar is calculated in the following way: each textual element of the content is tokenized by an appropriate tokenizer and each XML element is substituted by a list of values (called element value) returned by XPath based keys. Each key is defined by an XPath expression. The expression defining the key is evaluated over the corresponding element. The returned value for the key is transformed into an appropriate way and stored in the element value. In order to recognize whether a value in the input word is from a text or from an element, the CLaRK System requires that the values of an XML element were enclosed in angle brackets: <. . . >. For instance, the content of the following element:

```
<s>John <v>loves</v> Mary</s>
```

forms the following input word under appropriate tokenizers and XPath values:

```
"John" " " < "v" > " " "Mary"
```

In order to map the values in the input word, we use token descriptions in the regular expressions. The simplest token descriptions are the tokens themselves. Additionally, in the token description we can use wildcard symbols `#` for zero or more symbols, `@` for zero or one symbol, and token categories. Each token description in a regular expression is matched exactly to one token in the input word. It is important to see that the value of an XML element is not just the tag of the element. For example, if we have the following element:

```
<s>
   <w g="N">John</w>
   <w g="V">loves</w>
   <w g="N">Mary</w>
</s>
```

then the sequence of tags is simply `<"w">` ... `<"w">`, which is not intended to be an input word for a grammar. In order to get the values of the attributes of the elements, we have to use the following XPath expression: `attribute::g`. And then the input word will be: `<"N"> <"V"> <"N">`. Writing complex XPath expressions, the user can determine very complicated values for the XML elements depending of the context of their use.

The last problem when applying grammars over XML documents is how to incorporate the category assigned to a given rule. In general we accept that the category has to be encoded as an XML mark-up of arbitrary complexity. For instance, the following simple tagger (example is based on [Abney, 1996]):

```
"the"|"a" -> Det
"telescope"|"garden"|"boy" -> N
"slow"|"quick"|"lazy" -> Adj
"walks"|"see"|"sees"|"saw" -> V
"above"|"with"|"in" -> Prep
```

can be encoded in the CLaRK System as:

```
"the"|"a" -> <Det>\w</Det>
"telescope"|"garden"|"boy" -> <N>\w</N>
"slow"|"quick"|"lazy" -> <Adj>\w</Adj>
"walks"|"see"|"sees"|"saw" -> <V>\w</V>
"above"|"with"|"in" -> <Prep>\w</Prep>
```

where `\w` is a variable for the recognized word. The mark-up defining the category can be as complicated as necessary. The variable `\w` can be repeated as many times as necessary (it can also be omitted).

2.4 Constraints over XML documents

Several mechanisms for imposing constraints over XML documents are available. The constraints cannot be stated by the standard XML technology. The following types of constraints are implemented in CLaRK: 1) finite-state constraints - additional constraints over the content of given elements based on a document context; 2) number restriction constraints - cardinality constraints over the content of a document; 3) value constraints - restriction of the possible content or parent of an element in a document

based on a context. The constraints are used in two modes: checking the validity of a document regarding a set of constraints; supporting the linguist in his/her work during the building of a corpus. The first mode allows the creation of constraints for the validation of a corpus according to given requirements. The second mode helps the underlying strategy of minimisation of the human labour.

General syntax of the constraints in the CLaRK System is the following:

```
(Selector, Condition, Event, Action)
```

where the selector defines in 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 evaluation of the condition is a non-empty list of nodes then the constraints are applied; the event defines some conditions of the system when this constraint is checked for application. Such events can be: the selection of a menu item, the pressing of key shortcut, some editing command as enter a child or a parent and similar; the action defines the way of the actual application of the constraint. Here we present constraints of type "Some Children". This kind of constraints deal with the content of some elements. They determine the existence of certain values within the content of these elements. A value can be a token or an XML mark-up and the actual value for an element can be determined by the context. Thus a constraint of this kind works in the following way: first it determines to which elements in the document it is applicable, then for each such element in turn it determines which values are allowed and checks whether in the content of the element some of these values are presented as a token or an XML mark-up. If there is such a value, then the constraint chooses next element. If there is no such a value, then the constraint offers to the user a possibility to choose one of the allowed values for this element and the selected value is added to the content as a first child. Additionally, there is a mechanism for filtering of the appropriate values on the basis of the context of the element.

2.5 Cascaded Processing

The central view of the use of the CLaRK System is that an XML document under processing can be seen as a "blackboard" on which different tools can write some information, reorder it or delete it. The user can order the applications of the different tools so as to achieve the necessary processing. We call this possibility **cascaded processing** after the cascaded regular grammars. In this case we can order not just different grammars but also other tools like constraints, removal operations, transformations and sorting.

3 HPSG annotation in the CLaRK System

In this section we demonstrate how we process HPSG annotations in the CLaRK System. First, we briefly define feature graphs that are used. Afterwards, we present the main elements of a grammar in TRALE System and how it is related to feature graphs. Then we demonstrate how these feature graphs are presented in the CLaRK System. Then we discuss the mechanism for implementation of the interface between the CLaRK and TRALE system. Then we show an example of the annotation process based on the integration of the two systems. In future we envisage extension of this mechanism to support connection with other external systems.

3.1 Feature Graphs

In this section defined the feature graphs that we are using. First, we present a logical formalism for HPSG. Then a normal form for a finite theory is defined as a set of feature graphs. [Simov, 2002] shows that this normal form is suitable for the representation of an HPSG corpus and an HPSG grammar. We

consider these graphs as a representation of an HPSG annotation scheme.. Here we shortly present the syntax of the logic (SRL). For full description see [King, 1989].

$\Sigma = \langle \mathcal{S}, \mathcal{F}, \mathcal{A} \rangle$ is a finite **SRL signature** iff $\mathcal{S}$ is a finite set of *species*, $\mathcal{F}$ is a set of *features*, and $\mathcal{A} : \mathcal{S} \times \mathcal{F} \rightarrow Pow(\mathcal{S})$ is an *appropriateness function*. τ is a **term** iff τ is a member of the smallest set $\mathcal{T}$ such that (1) $: \in \mathcal{T}$, and (2) for each $\phi \in \mathcal{F}$ and each $\tau \in \mathcal{T}$, $\tau\phi \in \mathcal{T}$. δ is a **description** iff δ is a member of the smallest set $\mathcal{D}$ such that (1) for each $\sigma \in \mathcal{S}$ and for each $\tau \in \mathcal{T}$, $\tau \sim \sigma \in \mathcal{D}$, (2) for each $\tau_1 \in \mathcal{T}$ and $\tau_2 \in \mathcal{T}$, $\tau_1 \approx \tau_2 \in \mathcal{D}$ and $\tau_1 \not\approx \tau_2 \in \mathcal{D}$, (3) for each $\delta \in \mathcal{D}$, $\neg\delta \in \mathcal{D}$, (4) for each $\delta_1 \in \mathcal{D}$ and $\delta_2 \in \mathcal{D}$, $[\delta_1 \wedge \delta_2] \in \mathcal{D}$, $[\delta_1 \vee \delta_2] \in \mathcal{D}$, and $[\delta_1 \rightarrow \delta_2] \in \mathcal{D}$. Each subset $\theta \subseteq \mathcal{D}$ is an **SRL theory**. An HPSG grammar $\Gamma = \langle \Sigma, \theta \rangle$ in SRL consists of: (1) a signature Σ, which gives the ontology of entities that exist in the universe and the appropriateness conditions on them, and (2) a theory θ, which gives the restrictions upon these entities.

Let $\Sigma = \langle \mathcal{S}, \mathcal{F}, \mathcal{A} \rangle$ be a finite signature. A **feature graph** with respect to Σ is a directed, connected and rooted graph $\mathcal{G} = \langle \mathcal{N}, \mathcal{V}, \rho, \mathcal{S} \rangle$ such that: (1) $\mathcal{N}$ is a set of **nodes**, (2) $\mathcal{V} : \mathcal{N} \times \mathcal{F} \rightarrow \mathcal{N}$ is a partial **arc function**, (3) ρ is a **root node**, (4) $\mathcal{S} : \mathcal{N} \rightarrow \mathcal{S}$ is a total **species assignment function**, and (5) for each $\nu_1, \nu_2 \in \mathcal{N}$ and each $\phi \in \mathcal{F}$ such that $\mathcal{V}\langle \nu_1, \phi \rangle \downarrow$ and $\mathcal{V}\langle \nu_1, \phi \rangle = \nu_2$, then $\mathcal{S}\langle \nu_2 \rangle \in \mathcal{A}\langle \mathcal{S}\langle \nu_1 \rangle, \phi \rangle$. We say that the feature graph $\mathcal{G}$ is **finite** if and only if the set of nodes is finite. A feature graph $\mathcal{G} = \langle \mathcal{N}, \mathcal{V}, \rho, \mathcal{S} \rangle$ such that for each node $\nu \in \mathcal{N}$ and each feature $\phi \in \mathcal{F}$ if $\mathcal{A}\langle \mathcal{S}\langle \nu \rangle, \phi \rangle \downarrow$ then $\mathcal{V}\langle \nu, \phi \rangle \downarrow$ is called a **complete feature graph**. For each two graphs $\mathcal{G}_1 = \langle \mathcal{N}_1, \mathcal{V}_1, \rho_1, \mathcal{S}_1 \rangle$ and $\mathcal{G}_2 = \langle \mathcal{N}_2, \mathcal{V}_2, \rho_2, \mathcal{S}_2 \rangle$ we say that graph $\mathcal{G}_1$ **subsumes** graph $\mathcal{G}_2$ ($\mathcal{G}_2 \sqsubseteq \mathcal{G}_1$) iff there is an *isomorphism* $\gamma : \mathcal{N}_1 \rightarrow \mathcal{N}_2'$, $\mathcal{N}_2' \subseteq \mathcal{N}_2$, such that (1) $\gamma(\rho_1) = \rho_2$, (2) for each $\nu, \nu' \in \mathcal{N}_1$ and each feature ϕ, $\mathcal{V}_1\langle \nu, \phi \rangle = \nu'$ iff $\mathcal{V}_2\langle \gamma(\nu), \phi \rangle = \gamma(\nu')$, and (3) for each $\nu \in \mathcal{N}_1$, $\mathcal{S}_1\langle \nu \rangle = \mathcal{S}_2\langle \gamma(\nu) \rangle$. For each two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ if $\mathcal{G}_2 \sqsubseteq \mathcal{G}_1$ and $\mathcal{G}_1 \sqsubseteq \mathcal{G}_2$ we say that $\mathcal{G}_1$ and $\mathcal{G}_2$ are **equivalent.**

For finite feature graphs, we could define a translation into SRL descriptions using the correspondences between paths in the graph and terms. Thus we can interpret each finite feature graph as a description in SRL. Using the set of all finite feature graphs that subsume a given infinite feature graph, we can also define the interpretation of each infinite feature graph. Moreover, we can define a correspondence between the finite SRL theories and the feature graphs. Thus using the algorithm from [King and Simov, 1998] and adding the information from the signature as a special theory[1] we can represent each finite SRL theory as a set of feature graphs. An inference procedure over feature graphs is under development. It will reflect the semantics of the corresponding SRL theory.

We aim at proving out that feature graphs are adequate for the following important scenarios: (1) **Representation of an HPSG grammar.** The construction of a graph representation of a finite theory demonstrates that using feature graphs as grammar representation does not impose any restrictions over the class of possible finite grammars in SRL. (2) **Representation of an HPSG corpus.** Each sentence in the corpus is represented as a complete feature graph. One can easily establish a correspondence between the elements of the strong generative capacity of an HPSG grammar and the complete feature graphs. Thus complete feature graphs naturally become a good representation for an HPSG corpus. (3) **Representation of the annotation scheme.** We assume that an annotation scheme over the HPSG sort hierarchy can be considered a grammar. The feature graphs of such an annotation scheme will be constrained by the lexicon, which is available to the annotators, by the principles, which are stated as a theory, and by the input sentences. In this respect the level of granularity is flexible and accessible for the annotators. As a result, all the constraints that follow logically from the above sources of information can be exploited effectively during the annotation process.

3.2 TRALE System

For processing of HPSG grammar we use TRALE System. TRALE is a system for parsing, logic programming and constraint resolution with typed feature structures in a manner roughly consistent with

[1] In order to account for the information in the signature we construct a special theory

$$\theta_\Sigma = \{ \bigvee_{\sigma \in \mathcal{S}} [\bigwedge_{\mathcal{A}(\sigma,\phi) \neq \emptyset, \phi \in \mathcal{F}} [: \phi \approx : \phi]] \}.$$

Then for each theory θ we form the theory $\theta^e = \theta \cup \theta_\Sigma$ which is semantically equivalent to the original theory.

their use in HPSG. The system is implemented in Prolog. The first thing which is needed after starting the system is the compilation of a grammar. Each grammar consists of two main parts:

Signature:

A signature defines the ontology, i.e. the types, their appropriate attributes and the hierarchical order of types. In other words, the signature tells us what kind of objects the theory is about and what kinds of things are appropriate for these objects. The type hierarchy is encoded as an intended tree. It expresses the intuitive idea of subtyping. TRALE assumes that subtypes exhaustively cover their supertypes, in other words - that every object of a non-maximal type also belongs to one of the maximal types subsumed by it. Multiple inheritance is also supported.

Theory:

The theory specification can consist of the following types of declarations: relations, macros, lexical entries, lexical rules, extended phrase structure rules, universal constraints and Prolog-like definite clauses over typed feature structures. Each of them has specific syntax.

The main syntactic constructs in TRALE system are **descriptions:**

Descriptions are fundamental throughout TRALE. Both queries and answers are descriptions. Implicational constraints in the theory are descriptions that apply universally, to all objects. Also definite relations are specified with descriptions as arguments. The descriptions used as queries in general represent partial feature structures, i.e., they specify a (possibly empty) set of objects. The answers are fully specified feature structures with all types being maximally specific and all features filled in.

Thus, one very important characteristic of TRALE System is that the parse of a sentence is represented as a feature graph as defined above. They play a basic role in the connection between the two systems.

3.2.1 Signature in the CLaRK System

TRALE system extends SRL signature with supertypes in order to facilitate the writing of grammars. The signature is called type hierarchy in this setup. One example of type hierarchy is as follows:

```
type_hierarchy
   bot
      sign cat:cat agr:number phon:list
         phrase dtr1:sign dtr2:sign
         word
   cat
      np
      vp
      sv
      s
   number
      singular
      plural
   string
.
```

Here each type is on a new line, the appropriateness conditions are listed after the corresponding type and supertype-subtype relations is presented by indentation.

As a rule, a type hierarchy is represented in the CLaRK System as an XML document. The XML document reproduces the tree structure defined in the original file. The content of each tag `<element>` can be a sequence of zero or more `<feature>` tags, followed by zero or more `<element>` tags. Here is an example how the fragment of the type hierarchy above is transformed into an XML document:

```
<type_hierarchy>
    <element name="bot">
        <element name="sign">
            <feature name="cat" appr="cat"/>
            <feature name="agr" appr="number"/>
            <feature name="phon" appr="list"/>
            <element name="phrase">
                <feature name="dtr1" appr="sign"/>
                <feature name="dtr2" appr="sign"/>
            </element>
            <element name="word"></element>
        </element>
        <element name="cat">
            <element name="np"></element>
            <element name="vp"></element>
            <element name="sv"></element>
            <element name="s"></element>
        </element>
        <element name="number">
            <element name="singular"></element>
            <element name="plural"></element>
        </element>
        <element name="string"></element>
    </element>
</type_hierarchy>
```

The DTD defining the structure of this kind of documents is as follows:

```
<!ELEMENT type_hierarchy (element)>
<!ATTLIST type_hierarchy
    top CDATA #IMPLIED>

<!ELEMENT element (feature*, element*)>
<!ATTLIST element
    name CDATA #REQUIRED>

<!ELEMENT feature EMPTY>
<!ATTLIST feature
    name CDATA #REQUIRED
    appr  CDATA #REQUIRED>
```

The compilation of type hierarchies in the CLaRK System can be made in several ways. The system recognizes the TRALE syntax for type hierarchies, which uses line indents to express the hierarchy relations. The system reads the input file line by line until it reaches a line containing a single dot in it. The content of each line which is preceded by the % sign is discarded. Having read the input file successfully, the system produces an XML document which is loaded in the system editor. The structure of this document is as follows:

- each type (sort) is represented by the tag `<element>`. The name of the type (sort) is contained in an attribute called 'name'. Example: `<element name="bot">`.
- each feature of a type is represented by the tag `<feature>`. This tag has two required attributes: 'name' - the name of the feature and 'appr' - the appropriateness of this feature. The value of 'appr' attribute is expected to be among the names of the types defined in the hierarchy. The DTD

describing this kind of documents does not validate this, but the system shows error messages when the value is wrong.

The CLaRK System supports also conversion from an XML document containing signature to TRALE's signature. Before conversion the document is checked for consistency (hierarchy problems, recursive definitions, appropriateness). This prevents the system from generation of an invalid output. The XML representation of type hierarchies allows for their semi-automatic or fully automatic generation by using the CLaRK tools (XSL and XPath Transformations, Grammars, Value Constraints, etc.).

3.2.2 Feature Graphs in the CLaRK System

As we already mentioned above, the result of parsing a sentence in TRALE is represented as feature graphs. Thus in order to process further the results of the TRALE work in CLaRK, we need to have a representation of feature graphs. Feature graphs are also represented in the CLaRK System as XML documents. As feature graphs can be represented isomorphically as AVMs (Attribute Value Matrices), the XML representation is very similar to them. Even we can consider the XML representation to be a representation of AVMs. The representation is designed signature independent. So any changes in the signature or/and in the theory of TRALE will not need changes in the representation. Here we will describe how we present AVMs in XML. Each frame of the AVM is represented by the tag `<struc>`. It has one attribute 'type' containing the name of the type this frame stands for. Example: `<struct type="word">`. The features (attributes) of each frame are represented as tags `<feature>` imbedded as content of `<struct>`. The `<feature>` tag has an attribute "fn" which has as value the name of the feature. The value of the feature is represented as content of the tag `<feature>`. The value can be a frame as well (`<struct>`), or one of the atomic predefined types (empty lists, not empty lists, reference, etc.). For some of them we have a representation in the DTD (empty list - 'elist', not empty list - 'nelist', empty set - 'eset', etc.). Here is a fragment of the DTD defining this kind of XML documents:

```
<!ELEMENT struc (feature)*>
<!ATTLIST struc
        type CDATA #REQUIRED>

<!ELEMENT item (elist|reentr|nelist|struc|any|eset|neset)>

<!ELEMENT feature (elist|reentr|nelist|struc|any|eset|neset)>
<!ATTLIST feature
        fn CDATA #REQUIRED>

<!ELEMENT nelist (item)+>
<!ELEMENT elist EMPTY>
<!ELEMENT item (elist|reentr|nelist|struc|any|eset|neset)>
```

Here we give a short example fragment of an AVM and its corresponding XML representation. The original AVM:

$$\begin{bmatrix} \text{word} & \\ \text{PHON} & \langle\ \text{old}\ \rangle \\ \text{ARG-ST} & \text{elist} \\ \text{LR} & \boxed{\text{list}} \\ \text{SYNSEM} & \begin{bmatrix} \text{canon-synsem} & \\ \text{LEX} & \boxed{\text{bool}} \end{bmatrix} \end{bmatrix}$$

And the corresponding XML fragment:

```
<struc type="word">
    <feature fn="phon">
        <nelist>
            <item>
                <any>old</any>
            </item>
        </nelist>
    </feature>
    <feature fn="arg_st">
        <elist/>
    </feature>
    <feature fn="lr">
        <struc type="list"></struc>
    </feature>
    <feature fn="synsem">
        <struc type="canon_synsem">
            <feature fn="lex">
                <struc type="bool"></struc>
            </feature>
        </struc>
    </feature>
</struc>
```

3.3 Connection between CLaRK and TRALE

The connection between the CLaRK system and TRALE is based on exchange of feature graphs. First, the CLaRK System allows writing of feature graphs descriptions in XML format (see above). These XML descriptions can be sent (after internal transforming in an appropriate format) to the TRALE system as queries. The CLaRK System can support the writing of descriptions in various ways. One of them is the use of Value constraints automatically generated on the basis of a compiled (in XML) type hierarchy.

To ensure maximal consistency of the queries, the CLARK system processes the type hierarchy in the following way. First, for each type (sort) all inherited features are collected and after adding them to the list of the specific features for this type, a set of constraints are generated. These constraints have as actions: 'insert an attribute node' ('Some Attributes' constraints) and 'insert an element node' ('Some Children' constraints). When the user creates a new XML feature graph, and s/he applies the constraints at a certain point of the document, depending on the context, the constraints generate a list of all entries eligible for this place. Usually this is useful when adding features to a certain type. The constraints generate a list of all possible features for this type. Then a list of all already existing features is created and the difference of the two lists is considered. It is offered to the user to select a list entry and then the selected value is inserted in the document. This prevents the user from inserting incorrect data.

After the query is evaluated within TRALE, the result is converted into XML format in the following way:

1. The result. The output of the TRALE system represents a bracket description of a feature graph. The description consists of one main structure; one or more reentrances which represent the shared structures in the main structure and a basic tree representation of the structure. We are obtaining this input through opened sockets between the two systems.

2. Conversion. The input is parsed and is converted in one XML document. The root element of the new document is named by the tag `AVM`. It holds in attribute "query" the query for which the result is an "answer". Then the system recursively transfers the bracket format into XML format, discussed above.

3. Multiple result. Some queries can have more that one answer in TRALE system. They are presented as one XML document containing XML presentation of all feature graphs returned by TRALE.

This mechanism allows very flexible relation between the functionalities offered by the two systems. In the CLaRK System it is not necessary the queries to be written as feature graphs. It is enough that the necessary data allows conversion in this format via XLS Transformations. In opposite direction, it is not necessary for the user to work directly with feature graphs. She or he could transform the results of TRALE system into any other formats. The next section presents sentence annotation in CLaRK System using the connection between the two systems.

3.4 Sentence annotation in the CLaRK System

Here we present an example of sentence annotation using the connection between the two system. The overall annotation process includes the following steps:

Partial parsing step:

1. Sentence extraction from the corpus. The source of the sentence extraction is the BulTreeBank text corpus. As supporting modules, the CLaRK concordancer and CLaRK grammar engine are relied upon.

2. Automatic pre-processing. Each sentence needs first to be pre-processed on all the levels, that precede deeper syntactic annotation. This includes: (1) Morphosyntactic tagging; (2) Named entity recognition; (3) Part-of-speech disambiguator; (4) Partial parsing.

We aim at a result of a 100 % accurate partial parsed sentence. The accuracy is checked and validated by a human annotator with the assistance of the appropriate modules in the CLaRK System. These generally are system of constraints.

HPSG step:

The result from the previous step is encoded into an HPSG compatible representation with respect to the HPSG sort hierarchy. This encoding can be done by using cascaded grammars (see [Osenova 2002]) and other tools in the CLaRK System. Then it is sent to TRALE, which takes the partial sentence analyses as an input and evaluates all the attachment possibilities for them. The output is encoded as feature graphs.

Resolution step:

Here 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 minimise the number of the incoming possibilities.

To avoid the possible combinatorial explosion on the last level, we plan to apply special compilation techniques which will distribute syntactic information locally onto the overall structure and will encode part of that information as constraints over the structure.

Annotation Process Example

Here we give an example with an ambiguous sentence in order to highlight better the annotation steps and techniques, described above. Note that the HPSG output is simplified and instead of feature graphs we are using context grammar trees in order to demonstrate the main idea.

Sentence extraction

```
<S>the man saw the boy with the telescope in the garden</S>
```

Automatic pre-processing

The chosen sentence is tokenized, then the part of speech is assigned to all words in it and some partial grammar for determination of the non-recursive noun phrases is run. The result is:

```
<S>
  <NP><D>the</D><N>man</N></NP>
  <V>saw</V>
  <NP><D>the</D><N>boy</N></NP>
  <P>with</P> <NP><D>the</D><N>telescope</N></NP>
  <P>in</P> <NP><D>the</D><N>garden</N></NP>
</S>
```

HPSG grammar processing.

The set of all possible syntactic analyses of the partially parsed sentence are returned by the HPSG grammar. This is a set of feature graphs. Each graph in the set is converted to an XML representation. Let us suppose for our example that the grammar returns only three analyses:

Analysis 1:

```
<S>
 <NP><D>the</D><N>man</N></NP>
 <VP>
   <VP>
       <VP> <V>saw</V> <NP><D>the</D><N>boy</N></NP> </VP>
       <PP> <P>with</P> <NP><D>the</D><N>telescope</N> </NP> </PP>
   </VP>
   <PP> <P>in</P> <NP><D>the</D><N>garden</N></NP> </PP>
 </VP>
</S>
```

Analysis 2:

```
<S>
 <NP><D>the</D><N>man</N></NP>
 <VP>
   <VP>
     <V>saw</V>
     <NP>
       <NP><D>the</D><N>boy</N></NP>
       <PP> <P>with</P> <NP> <D>the</D><N>telescope</N> </NP> </PP>
     </NP>
   </VP>
   <PP> <P>in</P> <NP><D>the</D><N>garden</N></NP> </PP>
 </VP>
</S>
```

Analysis 3:

```
<S>
   <NP><D>the</D><N>man</N></NP>
   <VP>
      <VP> <V>saw</V> <NP><D>the</D><N>boy</N></NP> </VP>
      <PP>
         <P>with</P>
         <NP>
            <NP> <D>the</D><N>telescope</N> </NP>
            <PP> <P>in</P> <NP> <D>the</D><N>garden</N> </NP> </PP>
         </NP>
      </PP>
   </VP>
</S>
```

Manual annotation

The intersection of all possible analyses is calculated and represented as an XML document:

```
<S> <NP><D>the</D><N>man</N></NP>
    <V>saw</V>
    <NP><D>the</D><N>boy</N></NP>
    <P>with</P>
    <NP><D>the</D><N>telescope</N></NP>
    <PP>
      <P>in</P>
      <NP><D>the</D><N>garden</N></NP>
    </PP>
</S>
```

Note that the intersection of the three analyses contains more syntactic information than the partial analysis. This is so, because the PP 'in the garden' is shared by all the three analyses. The information from the possible analyses, which is not included into the intersection, is stored as constraints over this document. Then the annotator is asked to complete the intersection to a full analysis. For example, if we mark 'the telescope in the garden' as a single NP, then exactly one analysis is determined and it is not necessary to add more information. Note that in this simple example all the analyses are visible, but in a real case there could be thousands of analyses and their inspection one by one would not be feasible for the annotator without a software support. Also the user has the possibility to state negative information saying that certain element does not possess certain properties.

3.5 Further Developments

We have implemented only feature graph based connection between the two systems. At the moment we are working on establishing full connection. It is oriented towards the exchange of all syntactic constructions which the TRALE System works with, including type hierarchies, theories, general queries (using negation, disjunction and etc.), and others.

This will allow the user to use tools in the CLaRK System over HPSG grammars. For example, the descriptions in the theory could be sorted with respect to different data and in this way the user will have different views over the grammar. Also some simple rules for grammar consistency could be stated as constraints in CLaRK and thus checked without the TRALE system.

4 Conclusion

A corpus, which is built along the lines of the described framework guarantees a certain level of control over the annotation work. It is impossible for the annotator to deviate from the constraints imposed by the annotation scheme. Additional benefit comes from the fact that both - the annotation scheme and the corpus itself - have logical interpretations. It provides the experts with the possibility to write validation theories, which can be used to control the application of the annotation guidelines. Also a logical inference can be used to check the corpus for inconsistency. The combination of the guided annotation restrictions and the formalization of the encodings ensures a better management and processing over the corpus.

In the paper we described an interface between the CLaRK System and TRALE, based on feature graphs. This interface ensures consistency of the annotation of sentences with respect to an HPSG type hierarchy and a grammar. In future we envisage this interface to be extended to full integration of the two systems.

References

[Abney, 1996] Steve Abney. 1996. *Partial Parsing via Finite-State Cascades.* In: *Proceedings of the ESSLLI'96 Robust Parsing Workshop.* Prague, Czech Republic.

[DOM, 1998] DOM. 1998. *Document Object Model (DOM) Level 1. Specification Version 1.0.* W3C Recommendation. http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001

[Carroll, Minnen, and Briscoe, (in press)] J. Carroll, G. Minnen, T. Briscoe. Parser Evaluation Using a Grammatical Relation Annotation Scheme. In A. Abeillé (ed.), *Treebanks: Building and Using Syntactically Annotated Corpora*, Dordrecht: Kluwer. (in press)

[Götz and Meurers, 1997] Thilo Götz and W. Detmar Meurers. 1997. *The ConTroll system as large grammar development platform.* In *Proceedings of the ACL/EACL post-conference workshop on Computational Environments for Grammar Development and Linguistic Engineering.* Madrid, Spain.

[Harrison et. al., 1991] P. Harrison, St. Abney, E. Black, D. Flickinger, C. Gdaniec, R. Grishman, D. Hindle, B. Ingria, M. Marcus, B. Santorini, and T. Stralkowski. Evaluating syntax performance of parser/grammars of English. In *Proc. of the Workshop on Evaluating Natual Language Processing Systems.* pages 71-77. Berkeley, Ca, USA. 1991.

[King, 1989] Paul J. King. *A Logical Formalism for Head-Driven Phrase Structure Grammar.* Doctoral thesis, Manchester University, Manchester, England. 1989.

[King and Simov, 1998] Paul J. King and Kiril Iv. Simov. The automatic deduction of classificatory systems from linguistic theories. In *Grammars*, volume 1, number 2, pages 103-153. Kluwer Academic Publishers, The Netherlands. 1998.

[Kübler and Hinrichs, 2001] Kübler S. and E. Hinrichs. *From Chunks to Function-Argument Structure: A Similarity-Based Approach.* In *Proceedings of ACL-EACL 2001*, Toulouse, July 2001.

[Osenova 2002] Petya Osenova. 2002. *Bulgarian Nominal Chunks and Mapping Strategies for Deeper Syntactic Analyses.* In Proceedings from the Workshop on Linguistic Theories and Treebanks, 20-21 Sept., Sozopol, Bulgaria (in this volume).

[Pollard and Sag, 1987] Carl J. Pollard and Ivan A. Sag. 1987. *Information-Based Syntax and Semantics*, vol. 1. CSLI Lecture Notes 13. CSLI, Stanford, California, USA.

[Pollard and Sag, 1994] Carl J. Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar.* University of Chicago Press, Chicago, Illinois, USA.

[Simov, 2001] K. Simov. 2001. *Grammar Extraction from an HPSG Corpus.* In: Proc. of the RANLP 2001 Conference, Tzigov chark, Bulgaria, 5–7 Sept., pp. 285–287.

[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, pages: 558-560.

[Simov, Popova and Osenova, 2001] K. Simov, G. Popova, and P. Osenova. 2001. *HPSG-based syntactic treebank of Bulgarian (BulTreeBank).* In: *"A Rainbow of Corpora: Corpus Linguistics and the Languages of the World",* edited by Andrew Wilson, Paul Rayson, and Tony McEnery; Lincom-Europa, Munich, pp. 135-142.

[Simov, 2002] Kiril Iv. Simov. 2002. *Grammar Extraction and Refinement from an HPSG Corpus.* In *Proc. of the ESSLLI Workshop on Machine Learning Approaches in Computational Linguistics*, Trento, Italy. August 5-16, 2002. pages 38-55. http://www.BulTreeBank.org/

[Simov et al., 2002a] K.Simov, P.Osenova, M.Slavcheva, S.Kolkovska, E.Balabanova, D.Doikoff, K.Ivanova, A.Simov, M.Kouylekov. 2002. *Building a Linguistically Interpreted Corpus of Bulgarian: the BulTreeBank.* In: Proceedings from the LREC conference, Canary Islands, Spain.

[Simov et al., 2002b] K.Simov, M.Kouylekov, A.Simov. *Incremental Specialization of an HPSG-Based Annotation Scheme.* In: Proceedings of the Workshop on "Linguistic Knowledge Acquisition and Representation: Bootstrapping Annotated Language Data", the LREC conference, Canary Islands, Spain. 2002.

[TEI, 2001] Text Encoding Initiative. 1997. *Guidelines for Electronic Text Encoding and Interchange.* Sperberg-McQueen C.M., Burnard L (eds).

[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