CLaRK - an XML-based System for Corpora Development[1]

Kiril Simov, Zdravko Peev, Milen Kouylekov, Alexander Simov, Marin Dimitrov[2],
Atanas Kiryakov[3]
The CLaRK Programme
Linguistic Modelling Laboratory - CLPPI, Bulgarian Academy of Sciences
kivs@bgcict.acad.bg, zpeev@bgcict.acad.bg, mkouylekov@dir.bg, adis_78@dir.bg,
marin@sirma.bg, naso@sirma.bg

1 Introduction

In this paper we describe the architecture and the intended applications of the CLaRK system. The development of the CLaRK system started under the Tübingen-Sofia International Graduate Programme in Computational Linguistics and Represented Knowledge (CLaRK). The main aim behind the design of the system is the minimization of the human work during creation of corpora. Creation of corpora is still important task for majority of languages like Bulgarian where the invested effort in such development is very modest in comparison with more intensively studied languages like English, German and French. We consider the corpora creation task as editing, manipulation, searching and transforming documents. Some of these tasks will be done for single document or a set of documents, others will be done on a part of a document. Besides efficiency of the corresponding processing in each state of the work, the most important investment is the human work. Thus, in our view, the design of the system has to be directed to minimization of the human work.

For document management, storing and querying we chose XML technology because of its popularity and its ease for understanding. Very soon XML technology will be part of our lives and it will be the predominant language for data description and exchange on the Internet. Moreover a lot of already developed standards for corpus description, such as CES (Corpus Encoding Standard 2001) and TEI (Text Encoding Initiative 1997) are already adapted to XML requirements. The core of the CLaRK system is an XML Editor which is the main interface to the system. With the help of the editor the user can create, edit or browse XML documents. To facilitate corpus management we enlarge the XML inventory with facilities that support linguistic work. We added the following basic language processing modules: a tokenizer with a module that supports a hierarchy of token types, a finite-state engine that supports the writing of cascade finite-state grammars and facilities that search for finite-state patterns, the XPath query language which is able to support navigation over the whole set of mark-up of a document, mechanisms for imposing constraints over XML documents which are applicable in the context of some events, database over the XML mark-up and tokenized content of the documents.

We envisage several uses for our system: 1) *Corpus markup.* Here users work with the XML tools of the system in order to mark-up texts with respect to an XML DTD. This task usually requires an enormous human effort and comprises both the mark-up itself and its validation afterwards. Using the available grammar resources such as morphological analyzers or partial parsing, the system can state local constraints reflecting the characteristics of a particular kind of texts or mark-up. One example of such constraints can be as follows: a PP according to a DTD can have as parent an NP or VP, but if the left sister is a VP then the only possible parent is VP. The system can use such kind of constraints in order to support the user and minimize his work. 2) *Dictionary compilation for human users.* The system will support the creation of the actual lexical entries whose structure will be defined via an appropriate DTD. The XML tools will be used also for corpus investigation that provides appropriate examples of the word usage in the available corpora. The constraints incorporated in the system will be used for writing a grammar of the sublanguages of the definitions of the lexical items, for imposing constraints over elements of lexical entries and the dictionary as a whole.

[1] The work on the system is currently supported by BulTreeBank project 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.
[2] Currently at the OntoText Lab, Sirma AI LTD, Sofia
[3] Currently at the OntoText Lab, Sirma AI LTD, Sofia

The structure of the paper is as follows: in the next section we give a short introduction to the main notions of XML language and XPath querying language, the third section describes the main components of the CLaRK system and their functionality, the last section outlines some directions for future development.

2 XML technology

XML stands for *eXtendable Markup Language* (see XML 2000) and it emerged as a new generation language for data description and exchange for Internet use. The language is more powerful than HTML and easier to implement than SGML. Starting as a markup language, XML evolved into a technology for structured data representation, exchange, manipulation, transformation, and querying. The popularity of XML and its ease for learning and use made it a natural choice for the basis of CLaRK system. This section presents in informal way some of the most important notions of the XML technology. For more rigorous and full presentation the reader is directed to the corresponding literature on the following address: http://www.w3c.org/

XML defines the notion of structured document in terms of sequences and inclusions of elements in the structure of the document. The whole document is considered as an element which contains the rest of the elements. The elements of the structure of a document are marked-up by means of *tags*. Tags can surround the *content* of an element or tags can mark some points in the document. In the first case the beginning of an element is marked-up by the so called open tag written as <tagname> and the end is marked-up by a closing tag written as </tagname>. For example, TEI documents include on top level the following two elements:

```
<TEI.2>
<TeiHeader> ... content of the TEI header element ... </TeiHeader>
<text>
... Content of the text element ...
</text>
</TEI.2>
```

The tags of the second kind, so called empty elements, are represented as <tag/>. For example, a line break within a sentence can be represented in the following way:

```
<s> ... first line of text ... <lb/> ... second line of text ... </s>
```

Each element can be connected with a set of *attributes* and their *values*. The currently assigned set of attributes of an element is recorded within the open tag of the element or before the closing slash in an empty element. Some of the attribute-value pairs are by default assigned to some tags and thus it is not obligatory to list them.

One important requirement for an XML document is that elements having common content must strictly include one into another. This means that overlap of the elements is forbidden. Such documents are called *well-formed.* For example, the following document *is not well-formed* and thus it is not an acceptable XML document:

```
<doc><el1> ... <el2> ... </el1> ... </el2></doc>
```

XML technology defines a set of mechanisms for imposing constraints over XML documents. Such kind of a very basic mechanism is the so called DTD (Document Type Definition). A DTD defines the inclusion of elements and the possible sequences of elements within the content of an element. These definitions are given as ELEMENT statements in the DTD. Each ELEMENT statement has the following format:

```
<!ELEMENT tagname content_definition>
```

where *tagname* is the name of the element and *content_definition* is a definition of the content of this kind of elements. Besides some reserved words as EMPTY and ANY, the content definition is represented as a regular expression over tag names. This regular expression determines the tags and their order in the content of the enclosing element. Additionally, the DTD can contain definitions of

the allowed attributes for the elements, entities declaration and others. For more details the interested reader is directed to the literature on the corresponding notions – see the above address.

An XML document containing elements whose content obeys the restrictions stated in a DTD is said to be *valid* with respect to this DTD.

Another important language defined within XML world and used within the CLaRK system is 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 itself is the root of the tree and the children of a node represent the content of the corresponding element. Attributes and their values of each element are represented as an addition to the tree. Each expression in XPath language is evaluated with respect to a chosen node in the tree, called *context node*, and consists minimally of two parts: *axis* and *node test*. Optionally one can impose an additional predicate expression. Axis part determines the direction with respect to the context node in which the expression will be evaluated. Node test determines the type of the nodes that we are interested in. This can be text, tag names and so on. Among all nodes of the required type in the specified direction there can be nodes that we want to choose. These nodes are further filtered by a predicate expression which can be evaluated as true or false over a node. The XPath syntax allows for recursive XPath expressions and also union of XPath expressions. The complete definition of the XPath language can be found in XPath (1999). Here we give a very simple example, the following expression

```
/descendant-or-self::gram[contains(string(child::text()),"V")]
```

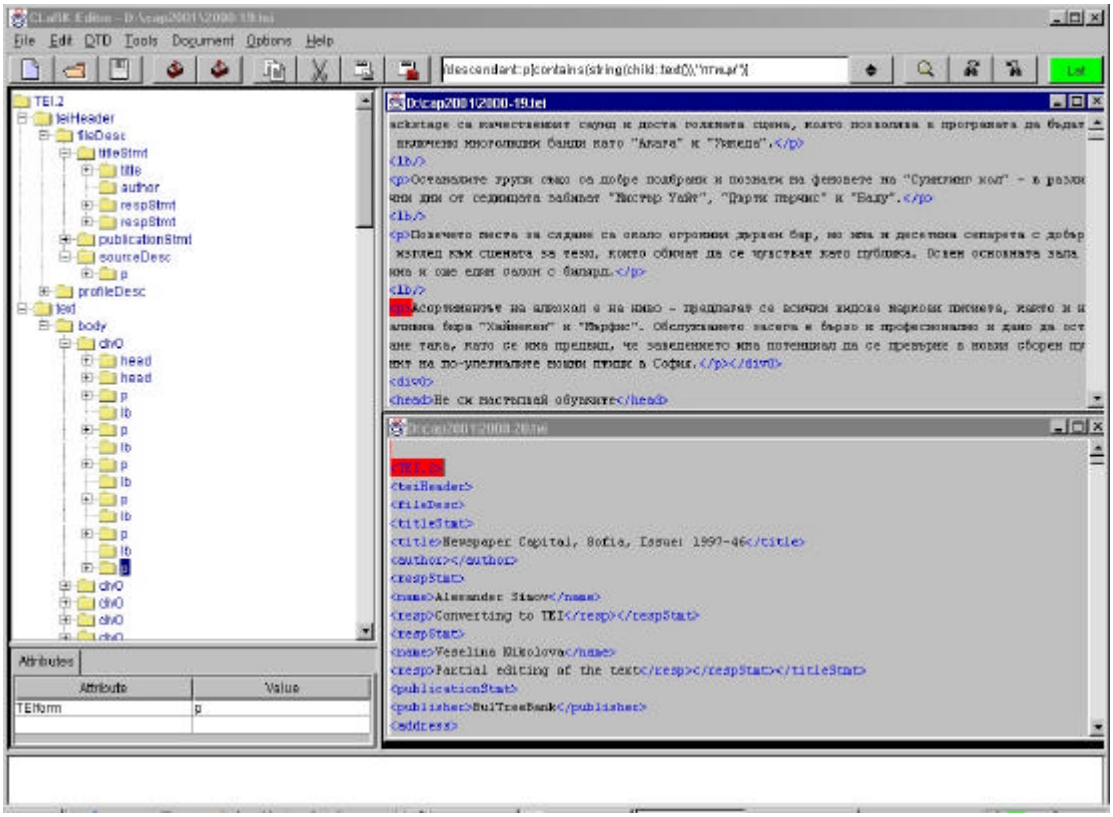
will be evaluated to a list of all <gram> nodes such that their textual value contains the letter "V".

3 CLaRK System

At the heart of the CLaRK system is the XML technology as a set of utilitiess for structuring, manipulation and management of data. 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 of single XML documents but of an integrated set of documents and constraints over them. The main goal of this development is to allow the user to add to the XML documents a desirable semantics reflecting the user's goals. Inside the system, the core structure of the representation of the XML documents follows the DOM Level1 specification (DOM 1998). When an XML document is imported (or created) in the system it is stored in this internal representation and in this way the user has access to it only via the facilities of the system. This restriction allows us to support the consistency of the data represented in the system. We plan to exploit this feature of the system even further in future for automatic support of construction of XML documents that reflect the content of a corpus or a set of documents.

The CLaRK system includes the following components: *XML Engine*, *XML Editor*, *Database*, *Document Transformation*, *Tokenizer*, *Constraints Engine*, *XPath Engine* and *FSA Engine*. In this section we describe each of these components in turn. Some of these components are not directly accessible by the user and they are used in the other components to support the corresponding functionality. The most important components of this type are the *XPath Engine* and the *FSA Engine*. The first is a module which evaluates XPath expressions over a document and the second is a module dealing with compilation of regular expressions into finite-state automata, determinization and minimization of the compiled automata.

The following screen shot gives an overview of the main interface to the system. On the left side of the screen the tree view of the current document is displayed, under it is the attribute value table shows the attributes and their values of the selected element. On the right side of the screen the textual representation of two documents is given. The message window is at the bottom of the screen.



3.1 XML Engine

XML Engine offers a full set of facilities for processing XML documents. This includes *DTD compiler* which compiles the element, entities and attribute definitions in a DTD and represents them in an internal format. For the elements the internal format is a set of finite-state automata corresponding to the content definition of the elements. These automata are determined and minimized during the compilation. Attributes and entities are stored as hash-tables. The second element of the XML engine is the *XML parser*. This parser transforms an XML document into internal for the system DOM representation. During the parsing process the parser checks the well-formedness of the document and reports the corresponding errors. The third component is the *Validator*. This module checks the validity of the document with respect to a DTD. Each document which is loaded in the system has to be attached to a DTD. Once a document is parsed to the internal representation of the system, it can be saved in this internal representation and the next time when it is used it will not be necessary to be parsed again. The Validator is active for the currently loaded document in the editor and when the user is modifies the document the Validator reports the changes in the validity of the document with pointers to the corresponding wrong elements of the document.

3.2 XML Editor

Access to the system is via a structure-driven editor which allows the user to edit and manipulate XML documents. Each loaded into the editor document is presented to the user in two or more views. One of these views reflects the tree structure of the document as described in the previous section. The other views of the document are textual. Each textual view shows the tags and the text content of the document. The tags in the textual view are separate elements from the rest of the text and can not be edited. The user has the possibility to attach to each textual view a filter which determines the tags and the content of which elements to be displayed in the view. This option allows the user to hide some of the information in the document and to concentrate on the rest of the information. With different textual views of the same document the user can attach different filters.

The editor supports a full set of editing operations, such as copy, cut, paste and so on. These operations are consistent with the XML structure of the document. Thus the user can copy or delete a

whole subtree of the document. Some of these operations as search and replace are defined in terms of XPath expressions. This allows the user to search not only in textual content of the document but also with respect to the XML mark-up. The most powerful operation here is the *XPath replace*. This operation is used for various commands for restructuring the document. Generally, the scenario is the following: (1) a list of nodes (subtrees, text elements) is chosen by the *Source* XPath expression. In this way the elements which will be copied or moved in the document are defined; (2) a list of nodes is chosen by the *Target* XPath expression. In this way the place(s) where the source elements will be copied or moved are defined; (3) the elements from source list are attached to the elements of the target list. There are several options defining the way of performing of the above action. These concern such things as whether the elements of the source are copied or cut from the document before they are attached to the target, the mapping between the source and target elements - it is possible for the source elements to be attached to each element of the target, or each element of the source to be attached to the corresponding element of the target. Via different options this operation becomes very powerful means for document modification or entering new information in case the source is given not as an XPath expression but as a fragment of XML document or text. In future we plan to allow an evaluation of the source and the target expressions over different documents and thus to allow their merging.

The editor allows editing of the document textual content or its structure. The editing of the structure is supported by the attached to the document DTD. When the cursor is located at some point in the document structure, the user can enter a child, a sibling or a parent of the pointed element. In both cases the DTD is consulted and the list of the allowed for this position tags is offered to the user.

3.3 Document Transformation

The system offers a general mechanism for the transformation of some XML documents into other XML documents. This is done by implementing XSLT language (XSLT 1999). The transformations can be applied in two modes: globally and locally. When a transformation is applied globally it is applied to the whole document. In future we plan some transformation to be applied to a set of documents. When the user wants to apply a transformation locally he/she first selects an appropriate fragment of the document and then the transformation is applied only to this fragment. This last option provides a mechanism for construction of a set of transformations which the user applies depending on the current task and thus avoiding the necessity to write very specific conditions on the applicability of the transformation.

Some transformations, corresponding to small changes in the DTD of documents (such as reordering of elements), are generated automatically.

3.4 Tokenizer

XML considers the content of each text element as a whole string that is unacceptable for corpus processing where one usually requires to distinguish wordforms, punctuation and other tokens in the text. In order to cope with 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, so called parent tokenizer. In the system tokens are used in different processing modules. For each tokenizer an alphabetical order over the token types is defined. This order is used for operations as comparing two tokens, sorting and similar.

Sometimes in different parts of one document the user will want to apply different tokenizers. For instance in a multilingual corpus the sentences in different languages will need to be tokenized by different tokenizers. In order to allow this functionality, the system allows for attaching tokenizers to the documents via the DTD of the document. To each DTD the user can attach a tokenizer which will be used for tokenization of all textual elements of the documents corresponding to the DTD. Additionally the user can overwrite the DTD tokenizer for some of the elements attaching to them other tokenizers.

3.5 Constraints

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 which 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.

At the moment the following constraints are implemented in the system:

FSA constraints

In this kind of constraints the action is defined as a regular expression which is evaluated over the content of each element selected by the selector. If the word formed by the content of element can be recognized as belonging to the language of the regular expression then the constraint is evaluated as true. Otherwise it is evaluated as false and an appropriate message is given. Because the content of the elements can contain text and tags, one problem here is how to determine the word which corresponds to the content of the element. For example, if all wordforms in a sentence are surrounded by <w> tag then the content of a sentence element will be a list of <w> tags which is obviously not acceptable. In order to overcome this problem we allow the “letters” used in the definition of the regular expressions to be of three kinds: *tag value*, *token type*, and *token value*.

Tag value is a string which is the result from the evaluation of an XPath expression over an element. The appropriate XPath expression for each tag is attached to the DTD. When the content of an element is converted into a word to be checked by the FSA constraint, for each non-textual element of the content the corresponding XPath expression is evaluated and the first node in the returned list is considered as a string. If this first node is a text node then the first token is taken as a value. If the node is an attribute then the first token of the attribute value is taken. Otherwise the tag of the node is taken as value.

Token type and token value “letters” correspond to the tokenized textual content of the element.

When the constraint is applied to an element, the element's immediate children are first processed, tokenizing any textual data, evaluating all tag values and then sent to the FSA representing the regular expression. Token's string value has a higher priority over token's category. Since the evaluation of the FSA is linear, the FSA may reject some sequences that are valid. For example consider the category LAT to be the category for all Latin words. Then having the regular expression

(LAT,<a>)|("to",<b>)

and evaluating it on the element

<el>to<a/></el>

the FSA will answer that the content of *el* is not valid.

Number Constraints

This kind of constraints are defined in terms of an XPath expression, which is evaluated to a list of nodes, and MIN and MAX values where MIN and MAX are natural numbers. The constraint is satisfied (evaluated as true) if the length of the list returned by the XPath expression is between MIN and MAX.

Value Constraints

These constraints determine the possible children or the parent of an element in a document. These constraints apply when the user enters a new child or a new parent of an element. In both cases a list of possible children or parents are determined by the DTD, but depending on the context in the document an additional reduction of these lists is possible. In case the only possible child of an

element is a text then these constraints determine the possible text values for the element. Let us take as an example the following definitions in a DTD:

```
<!ELEMENT np ((np, pp) | ...) >
<!ELEMENT vp ((vp, pp) | ...) >
```

which in part define that a PP can be attached to a NP or a VP. Then let us take the partially marked-up sentence:

```
<s>
<np>The man</np><v>saw</v><np>the boy</np><pp>in the garden</pp>
</s>
```

For the PP "in the garden" there are still two possibilities for a parent - a NP or a VP. But if the user enters a new information than "saw the boy" is a VP then for the PP "in the garden" there is only one possible parent - a VP. This information can be encoded in the system as a value constraint for the parent of PP elements. In future versions of the system we envisage such kind of constraints to be compiled from grammar represented in a grammar development environment.

3.6 Cascade Finite State Automata Grammars

Having a finite state automata facilities implemented in the system it is relatively simple to use them for regular grammar development. The CLaRK system incorporates mechanisms for writing cascade finite state grammars as defined in Abney (1996). The evaluation of the regular expressions follow the longest match strategy. Again the regular expression can be defined over tag values, token types and token values. The new category for each recognized word can be presented in the document as any kind of mark-up, but usually this is done by surrounding tag with appropriate attributes.

3.7 Database

A relational database over the content of the documents imported into the system is established in order to support an evaluation of limited XPath expressions over a set of documents. In the database information about the tags, the attributes and the tokens of the documents is stored. The documents are stored separately as files in internal format. The evaluation of a query with respect to the database returns a list of documents satisfying the query. These documents are further processed if this is necessary.

3.8 Other facilities

Here we will describe two more functions of the system which are useful in the process of corpora development. The first is concerned with sorting elements of a document according to some keys defined over these elements. The sorting is defined in terms of two XPath expressions. The first expressions determine which elements will be sorted. This expression is evaluated with respect to the root of the document as a context node. The second XPath expression defines the key for each element and it is evaluated for each node returned by the first XPath expression. The list of nodes returned by the first expression is sorted according to the keys of the nodes. Afterwards the nodes are returned in the document in the new order.

A concordance tool is implemented on the bases of the XPath engine and the sorting module. The first step in a concordance construction is to extract the relevant information from the current document. This is done by an XPath expression which is evaluated and the returns list of nodes is stored as a separate document. The extracted elements are ordered in an appropriate way with the help of the sorting module. For example, in case of appropriately marked-up corpus one can extract all verbs and order them with respect to the first noun on the right side of the verb (not necessarily the first word on the right side).

The system supports definition of different keyboards for supporting of different languages. At the moment we support the standard American keyboard and Bulgarian keyboard for entering Cyrillic letters.

4 Future developments

The ClaRk system will be very intensively used within the BulTreeBank Project which just has started at the Linguistic Modelling Laboratory – see Simov, Popova, Osenova (2001). We plan to extend the system in the following directions:

External programs. A mechanism for calling external programs which receive as input fragments of an XML document and returns also fragments of XML document. We envisage the actual communication to the external programs to be implemented via transformations of fragments of documents to and from special interface XML documents. In this way an external program will be declared within the system only once and the user will be able to use the program with XML documents with different structure.

Schemes of dependencies between elements in several documents. This is in connection with databases. We can consider each XML DTD as a conceptual scheme over XML documents. Then we can use a set of DTDs to describe interconnected schemes. We plan to implement support for such schemes. Because the task can prove to be very hard we will start with one basic DTD and auxiliary DTDs defining interconnections in table format.

We plan to extend the set of events and actions available to the user for defining the constraints. The target here will be a macro language for definitions of actions. Also we plan to make the constraints more active and as an activating event will be used the result from evaluation of some other contraint. In this way we will have mechanisms for propagation of information from one constraint to others.
We also plan to add statistical facility for evaluating the quantity characteristic of the documents. The result of this facility will be a table of the relative frequency of some mark-up to some other mark-up in the document. Also we plan to add some other views over documents that are not naturally represented in textual or tree view of XML document such as graph view reflecting the ID references inside a document or other interpretations of the content of a document.

References

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

Corpus Encoding Standard 2001 *XCES: Corpus Encoding Standard for XML*. Vassar College, New York, USA. http://www.cs.vassar.edu/XCES/

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

Simov K, Popova G, Osenova P 2001 *HPSG-based syntactic treebank of Bulgarian (BulTreeBank)*. In: Proceedings of Corpus linguistics 2001, Lancaster, UK.

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

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

XPath 1999 *XML Path Lamguage (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