Constraints for corpora development and validation[1]

Kiril Simov, Alexander Simov, Milen Kouylekov
BulTreeBank project
http://www.BulTreeBank.org
Linguistic Modelling Laboratory - CLPPI, Bulgarian Academy of Sciences
Acad. G.Bonchev Str. 25A, 1113 Sofia, Bulgaria
Tel: (+3592) 979 28 25, (+3592) 979 38 12, Fax: (+3592) 70 72 73
kivs@bultreebank.org, adis_78@dir.bg, mkouylekov@dir.bg

1 Introduction

In this paper we consider corpora as a set of XML documents. The guidelines for the creation of the corpora determine the semantics of the data, stored in them. Usually the guidelines prescribe the actual structure of the corpora, the used symbols, their meaning and the relations among them. Ideally, the software supporting the creation of a corpus has to allow all the constraints that follow from the guidelines to be imposed over the XML representation of the corpus. To the best of our knowledge, such software does not exist yet. The main problems come from the complexity of the data in the corpus and the impossibility it to be to completely formalized.

The XML technology provides two similar mechanisms for imposing constraints over XML documents (with respect to well-formedness): Document Type Definition (DTD) mechanism, and XML Schema. Both of them impose constraints over the content of the elements in the document. These constraints are local to the content of the given element and don't depend on the context of this element. In the paper we describe a tool for imposing additional constraints over XML documents, which allows representation of non-local dependencies among elements in one or several XML documents. The tool is a part of the CLaRK System (see (Simov et. al. 2001)) and it is based on XPath language. The CLaRK System is an XML-based system for corpora development. It incorporates several technologies:

- XML technology (see (XML 2000), (XPath 1999), (XSLT 1999));
- Unicode (see (UNICODE 2003));
- Regular Cascade Grammars (see (Abney 1996), (Simov et al. 2002b));
- Constraints over XML Documents.

On the basis of these technologies the following tools are implemented: XML Editor, Unicode Tokeniser, Sorting tool, Removing and Extracting tool, Concordancer, XSLT tool, Cascaded Regular Grammar tool, etc. The system is implemented in Java and it is freely available at the project site http://www.bultreebank.org/clark/index.html.

The structure of the paper is as follows: in the next section we give a short description of the technologies behind the CLaRK System. The third section describes the constraints implemented in the system. In the next section we demonstrate the applicability of the constraints for semi-automatic disambiguation of Bulgarian morphologically annotated texts. The last section concludes the paper.

2 Technologies behind CLaRK System

As it was mentioned in the Introduction, the CLaRK System is based on the following technologies: XML technology, which supports document editing, storing, querying, etc; Unicode for internal representation of the documents; Regular cascade grammars, which facilitate the implementation of taggers, chunkers, searching; Constraints over XML documents for monitoring. Here we briefly consider each of these technologies.

[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.

2.1 eXtendable Markup Language (XML)

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 basis of the CLaRK System. Additionally, the established standards for corpus descriptions like CES (Corpus Encoding Standard 2001) and TEI (Text Encoding Initiative 1997) are already adapted to meet the XML requirements. 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 in the structure of a document are marked-up by *tags*. Tags either surround the *content* of an element, or mark some points in the document. In the first case, the beginning of an element is marked-up by an opening tag, written as <tagname>, and the end is marked-up by a closing tag, written as </tagname>. The tags of the second kind, the so-called empty elements, are represented as <tagname/>. 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 opening tag of the element or before the closing slash in an empty element.

One important requirement for an XML document is that elements having common content must be strictly included one into another. It means that overlappings between elements is not allowed. Documents that meet this requirement (and some other - for details see (XML 2000)) 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>

There are two mechanisms for imposing constraints over XML documents: DTD (Document Type Definition) and XML Schema. Both define the possible inclusions of elements and the possible sequences of elements within the content of an element. In addition, XML Schema defines the shape of textual elements, which in the DTD are treated as a whole string. An XML document containing elements, whose content obeys the restrictions, stated in a DTD or in an XML Schema, is *valid* with respect to the DTD or the XML Schema.

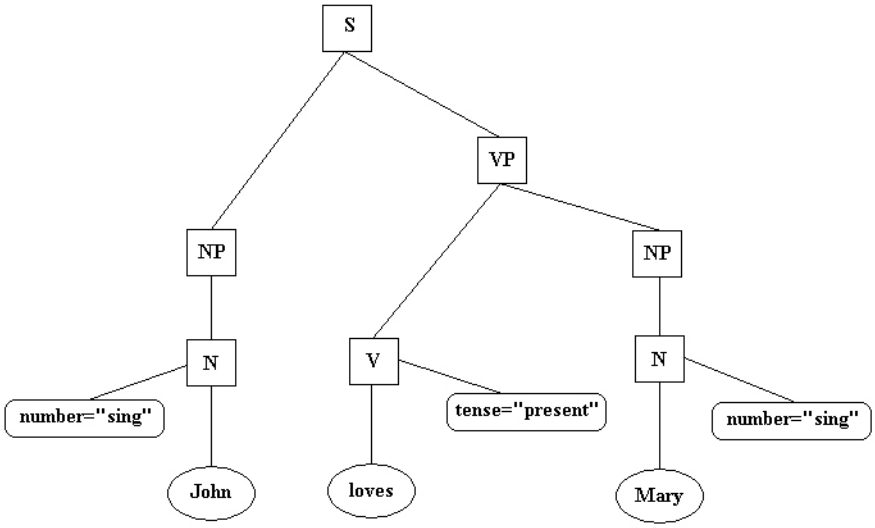
2.2 XML Path Language (XPath)

Another important language, defined within XML world and used within the CLaRK System, is 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. Attributes and their values of each element are represented as an addition to the tree. This section is based on the following document (*XML Path Language (XPath) version 1.0*. W3C Recommendation. http://www.w3.org/TR/xpath).

The XPath language plays a crucial role in the constraints of the CLaRK System, because it can be used for checking whether some configurations of data are presented in the document. In the implementation of the language in the system we have extended the above standard with variables and access to multiple XML documents. For instance, the following XML document:

```
<S>
    <NP><N number="sing"> John </N></NP>
    <VP>
        <V tense="present"> loves </V>
        <NP><N number="sing"> 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 contents 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 XPath language uses the tree-based terminology to point 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 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 nodes; 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 connected to 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. Some examples: let the "VP" node in the above tree is the context node, then its children are the "V" node and "NP" node that immediately dominates the node "Mary"; its descendant nodes (or simply descendants) are the nodes: "V", "loves", "NP", "N", "Mary"; its parent is the node "S"; at the same time this node is an ancestor of the node "VP"; its sibling is the left "NP" node and more precisely - this is its preceding sibling. If the context node is "V", then its ancestor nodes (or simply ancestors) are the nodes "VP" and "S"; its attribute node is the node 'tense="present"'; its following sibling is the right "NP" node. The result of the evaluation of one XPath expression could be a set of nodes in the tree (node-set), a string, a number, or a Boolean value.

The most important concept of the XPath language is the *location path*. Informally, the location path is a description of how to reach some nodes in the tree from the context node. More formally, the location path is a kind of expression. The location path selects a set of nodes relative to the context node. The result of evaluation a location path expression is a node-set containing the nodes selected by the location path. Location paths can recursively contain expressions that are used to filter sets of nodes.

Each location path expression consists of a sequence of location steps. Each location step has the following syntax:

axis::node-test[predicate]

where *axis* determines the part of the XML tree, in which the expression will select some nodes. The possible axes are *self, child, parent, descendant, descendant-or-self, ancestor, ancestor-or-self, following-sibling, following, preceding-sibling, preceding, attribute*. Each axis selects a set of nodes as it was informally described above. The *node-test* determines which of the nodes, selected by the axis, are of interest to us. Some of the node-tests are *text()* for selection of the text nodes only, *tag-name* for selection of nodes with this name etc. The predicate part *[predicate]* is optional. It can reduce the number of the selected nodes by rejecting those nodes for which the predicate is not satisfied. Such predicate could be equal to, not equal to, greater than, etc. A location path can start with the slash symbol (/) and, in this case, it is evaluated from the root of the tree and is called absolute location path. Otherwise the location path is relative.

Additionally to location paths, an XPath expression can be formed by function call. There are several functions provided by the XPath language. They include position(_) - which determines the position of a node in its context, string(_) - which converts the argument to a string, substring(_,_) - which returns a substring of a given string, and many others.

In the CLaRK System we have augmented XPath language with new predicates, a new axis and variables in order to facilitate the corpus construction and validation. The axis that we introduced is *document()*. It selects a different document for the evaluation of the next XPath expression. The new predicates include *content(RE),* which checks whether the content of the node satisfies the regular expression *RE*. Variables play a role of memory and can store any value, returned by some XPath expression. They can be used within other XPath expressions afterwards.

2.3 Unicode tokenization

XML DTD considers the content of each text element as a whole string; the XML Schema introduce a set of datatypes and regular expressions, which can impose constraints over the textual nodes in an XML document. From the point of view of corpora development the textual nodes usually are considered as a list of textual elements, such as wordforms, punctuation and other tokens. In order to provide possibility for imposing constraints over the textual node and to segment them in meaningful way, the CLaRK System supports a user-defined hierarchy of tokenisers. At the very basic level the user can define a tokeniser in terms of a set of token types. In this basic tokeniser each token type is defined by a set of UNICODE symbols. Above this basic level tokenisers, the user can define other tokenisers, for which the token types are defined as regular expressions over the tokens of some other tokeniser, the so called parent tokeniser. For each tokeniser an alphabetical order over the token types is defined. This order is used for operations like comparison between two tokens, sorting and similar.

2.4 Regular Grammars

The regular grammars are the basic mechanism for linguistic processing of the content of an XML document within the system. The regular grammar processor applies a set of rules over the content of some elements in the document and incorporates the categories of the rules back in the document as XML mark-up. The content is processed before the application of the grammar rules in the following way: textual nodes are tokenized with respect to some appropriate tokeniser, the element nodes are textualized on the basis of XPath expressions that determine the important information about the element. The recognized word is substituted by a new XML mark-up, which can or can not contain the word.

3 Constraints

The constraints that we implemented in the CLaRK System are generally based on the XPath language. 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 the constraint is used for validity check, similar to the validity check, which is based on

DTD or XML schema. In the second mode, the constraint is used to support the change of the document in order it to satisfy the constraint. In this section we present the constraints that are implemented in the system.
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 result fom the evaluation is approving (a non-empty list of nodes, a non-empty string, the true Boolean value, or a positive number), 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 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. There are three types of constraints, implemented in the system: regular expression constraints, number restriction constraints, value restriction constraints.

3.1 Regular expression constraints

In this kind of constraints the selection and condition are represented as one XPath expression. It determines the nodes, which the constraint will be applied to. The action, attached to each constraint of this kind, is defined as a regular expression, which is evaluated over the content of the selected elements. If the word, formed by the content of the 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. The content of the element is treated as the content of the element when a grammar is applied to it (see above). The user can navigate over the nodes that do not satisfy the constraint or those that satisfy it.

These constraints can be used for simulation of XML Schema constraints over textual nodes. In addition to checking the content, these constraints - via the XPath expression - can also determine the context of the elements, which they will be applied to. In this way they can be used for imposing regular constraints in addition to these in the DTD making them more specific on the basis of the surrounding context. This is very useful, for example, when someone is compiling a dictionary. Usually the DTD defines some very general content model for the elements, but in a concrete lexical entry a more specific model is realized. For example, let us assume that the <Paradigm> element in a lexical entry is defined as a disjunction of the following kind:

<!ELEMENT Paradigm ((sg,pl)|(pres,past,particple))>

The first part of the definition concerns the paradigm of nouns and the second - the paradigm of verbs. The information about the part of speech is presented in another element with tag POS and thus it is not available to the DTD validator. In order to support the consistency, we could write two constraints, which to express the dependency between the contents of the element POS and the element Paradigm:

Selector: /descendant::Paradigm[preceding-sibling::POS/text()="N"]
Action: (sg,pl)

Selector: /descendant::Paradigm[preceding-sibling::POS/text()="V"]
Action: (pres,past,particple)

The first constraint is satisfied by each Paradigm element, whose neighbor - POS element preceding it, has textual value N and thus the lexical entry is of a noun and the content of the element Paradigm is a sequence of one sg element and one pl element. Similarly for the second constraint.

3.2 Number restriction constraints

Here again the selection is defined as an XPath expression. The action is evaluation of another XPath expression with respect to each node, selected by the first expression. Then the newly selected nodes are counted and checked whether their number is in a given range. The range is given by two numbers MIN and MAX. If the number is in this range, then the constraint is satisfied by the node, selected with

the first expression. The MIN and MAX values can be dynamically determined by two XPath expressions, which return numbers as their values.

These kinds of constraints can be useful for checking equal number of nodes of different type within a given context. For instance, let us suppose that for a finite set of words of the language a^nb^n it is necessary to construct an XML representation. Then we can define an element <word> in the following way:

<!ELEMENT word (a*,b*)>

By regular expressions, however, it is impossible to specify that the *a*s are always the same number as the *b*s. It becomes very simple with number restriction constraints:

Selector:	/descendant::word
Min:	count(child::a)
Max:	count(child::a)
Action:	Min <= count(child::b) <= Max

and similar constraint for the number of *a*s.

The number constraints are very appropriate for stating general constraints over the whole document. Such constraints include arbitrary complex XPath expressions in their predicate part and otherwise they always select the root note if the predicate is satisfied in the document and require that there is exactly one such element.

3.3 Value restriction constraints

There are four types of these constraints: *parent*, *all children*, *some children*, *some attributes*. In validation mode all of them check whether the closest surrounding of the node satisfies some conditions. In *information entering support mode* they offer to the user a possibility for entering information in order to satisfy the constraint. Let us take a look over them in turn.

Parent constraints put additional restrictions over the possible parent of a node. In validation mode they are checked together with the constraints that are imposed in the DTD. Otherwise, *parent constraints* apply when the user inserts a new parent of a node. Then the system tries to calculate which are the possible parents for the node on the basis of the definitions in the DTD. *The parent constraints* reduce the number of the possible choices. *All children constraints* check whether the children of a node are among a list of possible elements. The list of possible elements can be relative to the context of the node and in this case they are selected by an XPath expression. If the content of the node is textual, then it is first tokenized and then checked whether each token is in a set of tokens, defined by the constraint. The *all children constraints* cannot be used in *information entering support mode*. *Some children constraints* also impose restrictions over the children of a node, but instead of all children; in this case some of them are necessary to be members of the possible children, determined by the constraint. In the mode of user entering information the constraints show to the user a list of the possible values for a new child of the node and he/she has to choose one. The new value can be incorporated in any position of the content of the node. Again, if the content is textual, then it is first tokenized. Similarly, some attribute constraints impose restrictions on the value of an attribute of the node. The difference is that in this case the values have to be textual.

The three kinds of constraints, which can be used for changing the content of the document, are exploitable also as rules. They become rules when they determine exactly one value on the basis of the context. In this case the user is not consulted and the value is entered automatically into the document.

4 Applications of constraints: semi-automatic disambiguation

In this section we present several uses of the constraints for actual creation and validation of corpora. The examples are based on the uses within the BulTreeBank project (see (Simov, Popova and Osenova 2001), (Simov et al 2002a)). Here we demonstrate how constraints of the kind "some children" are used to support the manual disambiguation of the morpho-syntactic tags of wordforms in the text. For

each wordform we encode the appropriate morpho-syntactic information from the dictionary as two elements: <aa> element, which contains a list of morpho-syntactic tags for the wordform, separated by a semicolon, and <ta> element, which contains the actual morpho-syntactic tag for this use of the wordform. Obviously the value of <ta> element has to be among the values in the list presented in the element <aa> for the same wordform. Here is one example:

```
<w><ph>Chovek</ph>     <aa>Ncmsi</aa>                  <ta></ta></w>
<w><ph>s</ph>          <aa>R</aa>                      <ta></ta></w>
<w><ph>minalo</ph>     <aa>Ansi;Ncnsi;Vpptcao-sni</aa> <ta></ta></w>
<w><ph>i</ph>          <aa>C</aa>                      <ta></ta></w>
<w><ph>istinski</ph>   <aa>Amsi;A-pi</aa>             <ta></ta></w>
<w><ph>char</ph>       <aa>Ncmsi</aa>                  <ta></ta></w>
```

where *Ncmsi* is a morphosyntactic tag for a common noun, masculine, singular, indefinite; *Ncnsi* is a morphosyntactic tag for a common noun, neuter, singular, indefinite; *Amsi* is a tag for an adjective, masculine, singular, indefinite; *Ansi* is a tag for an adjective, neuter, singular, indefinite; *A-pi* is a tag for an adjective, plural, indefinite; *Vpptcao-sni* is a tag for a participle, aorist, neuter, singular, indefinite; *C* is a tag for a conjunction; *R* is a tag for a preposition;

The constraint supporting disambiguation is defined in the following way:

```
Type:       Some Children
Selector:   ta
Condition:  self::*
Source:     preceding-sibling::aa/text()
```

The selector says that the constraint is about *ta* elements. There is no condition over the context of these elements. The source part determines that the appropriate values for the *ta* element are among the tokens (under one appropriate tokeniser) in the textual content of the preceding *aa* element. For the above example, the constraint will automatically fill the four unambiguous words and it will ask the user to choose among the following sets of values for the other two words: { Ansi Ncnsi Vpptcao-sni } and { Amsi A-pi }. The result after the correct choices is:

```
<w><ph>Chovek</ph>     <aa>Ncmsi</aa>                  <ta>Ncmsi</ta></w>
<w><ph>s</ph>          <aa>R</aa>                      <ta>R</ta></w>
<w><ph>minalo</ph>     <aa>Ansi;Ncnsi;Vpptcao-sni</aa> <ta>Ncnsi</ta></w>
<w><ph>i</ph>          <aa>C</aa>                      <ta>C</ta></w>
<w><ph>istinski</ph>   <aa>Amsi;A-pi</aa>             <ta>Amsi</ta></w>
<w><ph>char</ph>       <aa>Ncmsi</aa>                  <ta>Ncmsi</ta></w>
```

As it was already mentioned, one can use the constraints of this kind as rules. For instance, for the same data as above, we could encode a rule that considers the NP internal agreement in order to disambiguate for ambiguous adjectives:

```
Type:       Some Children
Selector:   ta
Condition:  preceding-sibling::aa[text()="Amsi;A-pi"
                        and
                        parent::w/following-sibling::w/aa[text()="Ncmsi"]]
Source:     Amsi
```

The rule says: if the word in question can be an adjective, masculine, singular, indefinite, or plural, indefinite and the next word is a noun, masculine, singular, indefinite, then choose this word to be an adjective, masculine, singular, indefinite. The source in this case is treated as a string, not as an XPath expression. Using such kind of rules, we have written about 20 rules, which resolve more than 30% of all ambiguities in Bulgarian newspaper texts. All of the presented so far constraints can be used in validation mode.

5 Conclusion

In the paper we presented several ways of imposing constraints over XML documents which can be used during the implementation of a corpus and during validation of a corpus. They are based on XPath language as a means for checking the presence of some information in the XML files that encode the corpus. In CLaRK System three general kinds of constraints are implemented: regular expression constraints, number restriction constraints, value restriction constraints. They allow two modes of usage: validation mode and information entering mode. The first one just checks if the constraint is satisfied in the document(s) and reports errors otherwise. The second mode is available only for value constraint (except the all children constraints). In this mode the constraints assist the user to change the document in such a way that it to satisfies the constraints. When the context in which the constraints are applied allows only one choice for some value, then the constraints encode rules.

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: "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 K, Peev Z, Kouylekov M, Simov A, Dimitrov M, Kiryakov A. 2001. *CLaRK - an XML-based System for Corpora Development*. In: Proc. of the Corpus Linguistics 2001 Conference. Lancaster, England. pp: 558-560.

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

Simov K, Kouylekov M, Simov A 2002 *Cascaded Regular Grammars over XML Documents*. In: Proc. of the 2nd Workshop on NLP and XML (NLPXML-2002), Taipei, Taiwan. September 1, 2002.

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

UNICODE 2003 Unicode Home Page. http://www.unicode.org/

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