HLT-NAACL 2003 Workshop: Software Engineering and Architecture
of Language Technology Systems , pp. 9-16
Edmonton, May-June 2003

WHAT: An XSLT-based Infrastructure for the Integration of Natural Language Processing Components

Ulrich Schäfer
Language Technology Lab, German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
Ulrich.Schaefer@dfki.de

Abstract

The idea of the Whiteboard project is to integrate deep and shallow natural language processing components in order to benefit from their synergy. The project came up with the first fully integrated hybrid system consisting of a fast HPSG parser that utilizes tokenization, PoS, morphology, lexical, named entity, phrase chunk and (for German) topological sentence field analyses from shallow components. This integration increases robustness, directs the search space and hence reduces processing time of the deep parser. In this paper, we focus on one of the central integration facilities, the XSLT-based Whiteboard Annotation Transformer (WHAT), report on the benefits of XSLT-based NLP component integration, and present examples of XSL transformation of shallow and deep annotations used in the integrated architecture. The infrastructure is open, portable and well suited for, but not restricted to the development of hybrid NLP architectures as well as NLP applications.

1 Introduction

During the last decade, SGML and XML have become an important interchange format for linguistic data, be they created manually by linguists, or automatically by natural language processing (NLP) components.

LT-XML (Brew et al. 2000), XCES (Ide and Romary 2001) and many other are examples for XML-based or XML-supporting software architectures for natural language processing.

The main focus of the Whiteboard project (2000-2002) was to integrate shallow and deep natural language processing components. The idea was to combine both in order to benefit from their advantages. Successful and beneficial integration included tokenization, PoS, morphology, lexical, named entity, phrase chunk and (for German) topological sentence field levels in a fully automated XML-based system. Crysmann et al. (2002) and Frank et al. (2003) show that this close deep-shallow combination significantly increases robustness and performance compared to the (already fast) standalone deep HPSG parser by Callmeier (2000). The only comparable architecture so far was described by Grover et al. (2002), but their integration was limited to tokenization and PoS tagging (the shallow chunker did not guide or contribute to deep analysis).

In this paper, we will focus on one of the central integration facilities, the XSLT-based Whiteboard Annotation Transformer (WHAT), report on the benefits of XSLT-based NLP component integration, and present examples of XSL transformation of shallow and deep annotations used in the integrated architecture.

Because the infrastructure is in general independent of deep or shallow paradigms, it can also be applied to purely shallow or deep systems.

2 Whiteboard: Deep-Shallow Integration

Deep processing (DNLP) systems[1] try to apply as much linguistic knowledge as possible during the analysis of sentences and result in a uniformly represented collection of the knowledge that contributed to the analysis. The result often consists of many possible analyses per sentence reflecting the uncertainty which of the possible readings was intended – or no answer at all if the linguistic knowledge was contradictory or insufficient with respect to the input sentence.

Shallow processing (SNLP) systems do not attempt to achieve such an exhaustive linguistic analysis. They are desigend for specific tasks ignoring many details in input and linguistic framework. Utilizing rule-based (e.g., finite-state) or statistics-based approaches, they are in general much faster than DNLP. Due to the lack of efficiency and robustness of DNLP systems, the trend in application-oriented language processing system development in the last years was to improve SNLP systems. They are now capable of analyzing Megabytes of texts within seconds, but precision and quality barriers are so obvious (especially on domains the systems where not designed for or trained on) that a need for 'deeper' systems re-emerged. Moreover,

[1] In this paper, 'deep' is nearly synonymous to typed unification-based grammar formalisms, e.g. HPSG (Pollard and Sag 1994), although the infrastructure may also apply to other deep linguistic frameworks.

semantics construction from an input sentence is quite poor and erroneous in typical shallow systems.

But also development of DNLP made advances during the last few years, especially in the field of efficiency (Callmeier 2000).

A promising solution to improve quality of natural language processing is the combination of deep and shallow technologies. Deep processing benefits from specialized and fast shallow analysis results, shallow processing becomes 'deeper' using at least partial results from DNLP.

Many natural language processing applications could benefit from the synergy of the combination of deep and shallow, e.g. advanced information extraction, question answering, or grammar checking systems.

The Whiteboard architecture aims at integrating different language technology components. Both online and offline coupling of existing software modules is supported, i.e., the architecture provides direct access to standoff XML annotation as well as programming interfaces. Applications communicate with the components through programming interfaces. A multi-layer chart holds the linguistic processing results in the online system memory while XML annotations can be accessed online as well as offline. Figure 1 gives an overview of the general architecture called WHAM (WHiteboard Annotation Machine).

There are two main points of the architecture that are important to stress. First, the different paradigms of DNLP and SNLP are preserved throughout the architecture, e.g. there is a shallow and a deep

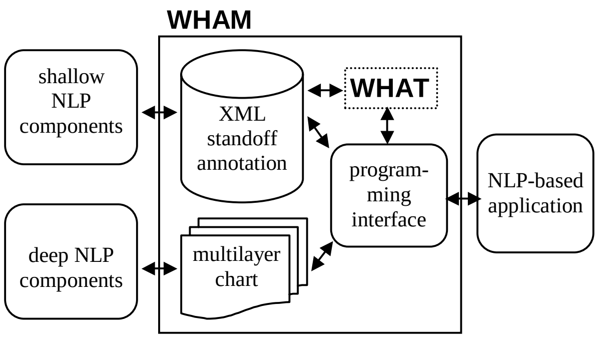

Figure 1: Whiteboard Architecture: WHAM

programming interface.

The second point is that the WHAM offers programming interfaces which are not simply DOM interfaces isomorphic to the XML markup they are based on, but hierarchically defined classes. E.g., a fast index-sequential storage and retrieval mechanism based on XML is encapsulated through the shallow programming interface. However, while the typed feature structure-based programming interface to deep components is stable, it turned out that the XML-based interface was too inflexible when new, mainly shallow, components with new DTDs had to be integrated. Therefore, a more flexible approach had to be devised.

3 WHAT: The Whiteboard Annotation Transformer

The main motivation for developing an XSLT-based infrastructure for NLP components was to provide flexible access to standoff XML annotations produced by the components.

XSLT (Clark 1999) is a W3C standard language for the transformation of XML documents. Input of an XSL transformation must be XML, while output can be any syntax (e.g., XML, text, HTML, RTF, or even programming language source code, etc.). The power of XSLT mainly comes from its sublanguage XPath (Clark and DeRose 1999), which supports access to XML structure, elements, attributes and text through concise path expressions. An XSL stylesheet consists of templates with XPath expressions that must match the input document in order to be executed. The order in which templates are called is by default top-down, left to right, but can be modified, augmented, or suppressed through loops, conditionals, and recursive call of (named) templates.

WHAT, the WHiteboard Annotation Transformer, is built on top of a standard XSL transformation engine. It provides uniform access to standoff annotation through queries that can either be used from non-XML aware components to get access to information stored in the annotation (V and N queries), or to transform (modify, enrich, merge) XML annotation documents (D queries).

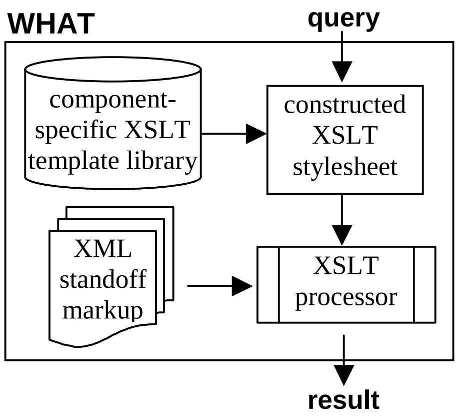
WHAT
query
component-specific XSLT template library
constructed XSLT stylesheet
XML standoff markup
XSLT processor
result

Figure 2: WHAT Architecture

While the WHAT is written in a programming language such as Java or C, the XSL query templates that are specific for a standoff DTD of a component's XML output are independent of that programming language, i.e., they must only be written once for a new component and are collected in a so-called template library.

3.1 WHAT Queries

Based on an input XML document (or DOM object), a WHAT query that consists of

- component name,
- query name, and
- query-specific parameters such as an index or identifier

is looked up in the XSLT template library for the specified component, an XSLT stylesheet is returned and applied to the XML document by the XSLT processor. The result of stylesheet application is then returned as the answer to the WHAT query. There are basically three kinds of results:

- strings (including non-XML output, e.g. RTF or even programming language source code)
- lists of unique identifiers denoting references to nodes in the XML input document
- XML documents

In other words, if we formulate queries as functions, we get the following query signatures:

- getValue: $C \times D \times P^* \rightarrow S^*$
- getNodes: $C \times D \times P^* \rightarrow N^*$
- getDocument: $C \times D \times P^* \rightarrow D$

where C is the component, D an XML document, P* a (possibly empty) sequence of parameters, S* a sequence of strings, and N* a sequence of node identifiers.
We now give examples for each of the query types.

3.1.1 V-queries (getValue)

V-queries return string values from XML attribute values or text. The simplest case is a single XPath lookup. As an example, we determine the type of named entity 23 in a shallow XML annotation produced by the SPPC system (Piskorski and Neumann 2000).
The WHAT query

```
getValue("NE.type", "de.dfki.lt.sppc", 23)
```

would lead to the lookup of the following query in the XSLT template library for SPPC

```
<query name="getValue.NE.type" component="de.dfki.lt.sppc">
  <!-- returns the type of named entity as number -->
  <xsl:param name="index"/>
  <xsl:template match="/WHITEBOARD/SPPC_XML//NE[@id=$index]">
    <xsl:value-of select="@type"/>
  </xsl:template>
</query>
```

On appropriate SPPC XML annotation, containing the named entity tag e.g. `<NE id="23" type="location"`…> somewhere below the root tag, this query would return the String `"location"`.
By adding a subsequent lookup to a translation table (through XML entity definitions or as part of the input document or of the component-specific template library), it would also be possible to translate namings, e.g. in order to map SPPC-annotation-specific namings to HPSG type names.
We see from this example how the WHAT helps to abstract from component-specific DTD structure and namings. However, queries need not be that simple. Complex computations can be performed and the return value can also be numbers, e.g., for queries that count elements, words, etc.

3.1.2 N-queries (getNodes)

An important feature of WHAT is navigation in the annotation. N-queries compute and return lists of node identifiers that can again be used as parameters for subsequent (e.g. V-)queries.
The sample query returns the node identifiers of all named entities (NE tags) that are in the given range of tokens (W tags). The template calls a recursive auxiliary template that seeks the next named entity until the end of the range is reached. The WHAT query

```
getNodes("W.NEinRange", "de.dfki.lt.sppc",3,19)
```

would lead to the lookup of the following query in the XSLT template library for SPPC.

```
<query name="getNodes.W.NEinRange" compon.="de.dfki.lt.sppc">
<!-- returns NE nodes starting exactly at token $index to
    (at most) token $index2 -->
  <xsl:param name="index"/> <xsl:param name="index2"/>
  <xsl:template match="/">
   <xsl:variable name="startX"
select="/WHITEBOARD/SPPC_XML//W[@id=$index]/ancestor::NE"/>
   <xsl:if test="$startX//W[1]/@id = $index">
    <xsl:call-template name="checknextX">
      <xsl:with-param name="nextX" select="$startX"/>
      <xsl:with-param name="lastW" select="$index2"/>
    </xsl:call-template>
   </xsl:if>
  </xsl:template>
  <xsl:template name="checknextX">
   <!-- auxiliary template (recursive) -->
   <xsl:param name="nextX"/>
   <xsl:param name="lastW"/>
   <xsl:variable name="Xtokens" select="$nextX//W"/>
   <xsl:if test="number(substring($Xtokens[last()]/@id, 2))
&lt;= number(substring($lastW, 2))">
    <xsl:value-of select="$nextX/@id"/>
    <xsl:text> </xsl:text>
    <xsl:call-template name="checknextX">
     <xsl:with-param name="nextX"
select="/WHITEBOARD/SPPC_XML//NE[@id=concat('N', string(1 +
number(substring($nextX/@id,2))))]"/>
     <xsl:with-param name="lastW" select="$lastW"/>
    </xsl:call-template>
   </xsl:if>
  </xsl:template>
</query>
```

Again, the query forms an abstraction from DTD structure. E.g., in SPPC XML output, named entity elements enclose token elements. This need not be the case for another shallow component; its template would be defined differently, but the query call syntax would be the same.

3.1.3 D-queries (getDocument)

D-queries return transformed XML documents - this is the classical, general use of XSLT. Complex transformations that modify, enrich or produce (standoff) annotation can be used for many purposes. Examples are

- conversion from a different XML format
- merging of several XML documents into one
- auxiliary document modifications, e.g. to add unique identifiers to elements, sort elements etc.
- providing interface to NLP applications (up to code generation for a programming language compiler…)
- visualization and formatting (Thistle, HTML, PDF, …)
- perhaps the most important is to do (linguistic) computation and transformation in order to turn a WHAT query into a kind of NLP component itself. This is e.g. intensively used in the shallow topological field parser integration we describe below. Multiple queries are applied in a sequence to transform a topological field tree into a list of constraints over syntactic spans that are used for initialization of the deep parser's chart. One of these WHAT queries has more than 900 lines of XSLT code.

We can show only a short example here, a query that inserts unique identifier attributes into an arbitrary XML document without id attributes.

```
<query name="getDocument.generateIDs">
 <!-- generate unique id for each element -->
 <xsl:template match="*">
  <xsl:copy select=".">
   <xsl:attribute name="id">
    <xsl:value-of select="generate-id()"/>
   </xsl:attribute>
   <xsl:for-each select="@*">
    <xsl:copy-of select="."/>
   </xsl:for-each>
   <xsl:apply-templates/>
  </xsl:copy>
 /xsl:template>
</query>
```

Note that this is an example for a stylesheet that is completely independent of a DTD, it just works on any XML document – and thus shows how generic XSL transformation rules can be.

Another example is transformation of XML tree representations into Thistle trees (arbora DTD; see Calder 2000). While the output DTD is fixed, this is again not true for the input document which can contain arbitrary element names and branches. Thistle visualizations generated through WHAT are shown in Fig. 4, 5 and 6 below.

3.2 Components of the Hybrid System

The WHAT has been successfully used in the Whiteboard architecture for online analysis of German newspaper sentences. For more details on motivation and evaluation cf. Frank et al. (2003) and Becker and Frank (2002). The simplified diagram in Figure 3 depicts the components and places where WHAT comes into play in the hybrid integration of deep and shallow processing components (V, N, D denote the WHAT query types). The system takes an input sentence, and runs three shallow systems on it:

- the rule-based shallow SPPC (Piskorski and Neumann 2000) for named entity recognition,
- TnT/Chunkie, a statistics-based shallow PoS tagger and chunker by (Skut and Brants 1998),
- LoPar, a probabilistic context-free parser (Schmid 2000), which takes PoS-tagged tokens as input, and produces binary tree representations of sentence fields, e.g., topo.bin in Fig. 4. For a justification for binary vs. flat trees cf. Becker and Frank (2002).

The results of the components are three standoff annotations of the input sentence. Then, a sequence of D-queries is applied to flatten the binary topological field trees (result is topo.flat, Fig. 5), merge with shallow chunk information from Chunkie (topo.chunks, Fig. 6), and apply the main D-query computing bracket information for the deep parser from the merged topo tree (topo.brackets, Fig. 7).

Finally, the deep parser PET (Callmeier 2000), modified as described in Frank et al. (2003), is started with a chart initialized using the shallow bracket information (topo.brackets) through WHAT V and N queries. PET also accesses lexical and named entity information from SPPC through V queries.

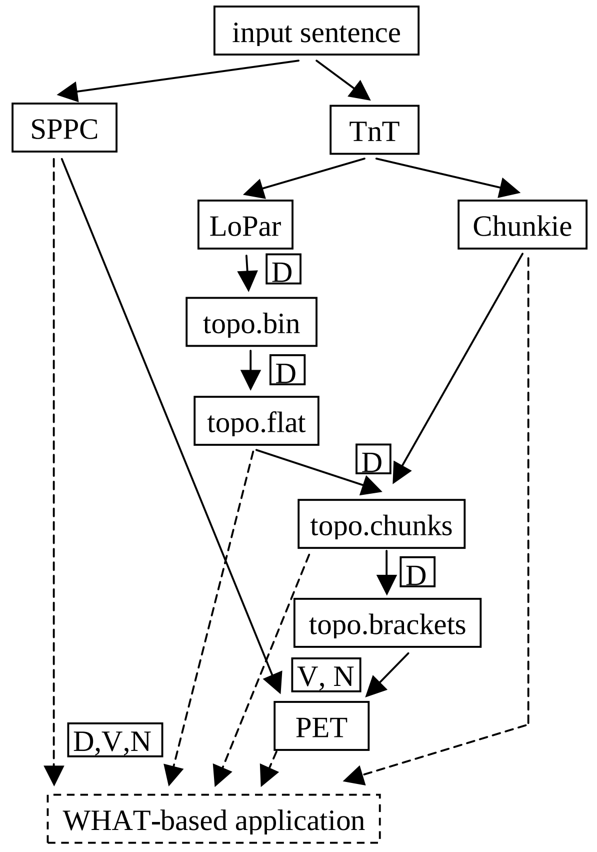

Figure 3: WHAT in the hybrid parser

Again, WHAT abstraction facilitates exchange of the shallow input components of PET without needing to rewrite the parser's code.

The dashed lines in Figure 3 indicate that a WHAT-based application can have access to the standoff annotation through D, V or N queries.

The Thistle diagrams below are created via D queries out of the intermediate topo.* trees.

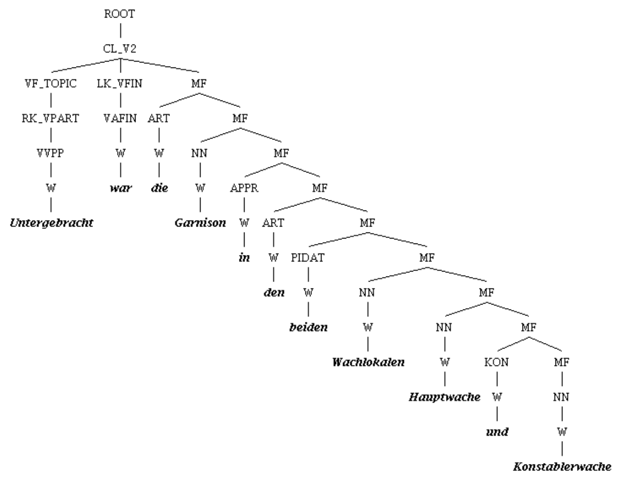

Figure 4. A binary tree as result of the topoparser (topo.bin).

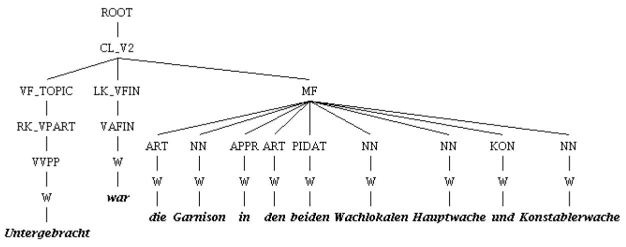
ROOT
CL_V2
VF_TOPIC
LK_VFIN
MF
RK_VPART
VAFIN
ART NN APPR ART PIDAT NN NN KON NN
VVPP
W
Untergebracht
war
die Garnison in den beiden Wachlokalen Hauptwache und Konstablerwache

Figure 5. The same tree after flattening (topo.flat).

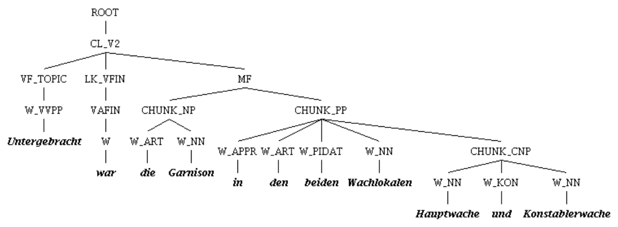

Figure 6. The topo tree merged with chunks (topo.chunks).

```
<TOPO2HPSG type="root" id="5608">
 <MAP_CONSTR id="T1"  constr="v2_cp"                 left="W1"  right="W13"/>
 <MAP_CONSTR id="T2"  constr="v2_vf"                 left="W1"  right="W2"/>
 <MAP_CONSTR id="T3"  constr="vfronted_vfin+rk"      left="W3"  right="W3"/>
 <MAP_CONSTR id="T4"  constr="vfronted_vfin+vp+rk"   left="W3"  right="W13"/>
 <MAP_CONSTR id="T5"  constr="vfronted_vp+rk"        left="W4"  right="W13"/>
 <MAP_CONSTR id="T6"  constr="vfronted_rk-complex"   left="W7"  right="W7"/>
 <MAP_CONSTR id="T7"  constr="vl_cpfin_compl"        left="W9"  right="W13"/>
 <MAP_CONSTR id="T8"  constr="vl_compl_vp"           left="W10" right="W13"/>
 <MAP_CONSTR id="T9"  constr="vl_rk_fin+complex+f"   left="W12" right="W13"/>
 <MAP_CONSTR id="T10" constr="extrapos_rk+nf"        left="W7"  right="W13"/>
</TOPO2HPSG>
```

Figure 7. The extracted brackets (topo.brackets)

3.3 Accessing and Transforming Deep Annotation

In the sections so far, we showed examples for shallow XML annotation. But annotation access should not stop before deep analysis results. In this section, we turn to deep XML annotation. Typed feature structures provide a powerful, universal representation for deep linguistic knowledge.

While it is in general inefficient to use XML markup to represent typed feature structures during processing (e.g. for unification, subsumption operations), there are several applications that may benefit from a standardized system-independent XML markup of typed feature structures, e.g., as exchange format for

- deep NLP component results (e.g. parser chart)
- grammar definitions
- feature structure visualization or editing tools
- feature structure 'tree banks' of analysed texts

Sailer and Richter (2001) propose an XML markup where the recursive embedding of attribute-value pairs is decomposed into a kind of definite equivalences or non-recursive node lists (triples of node ID, type name and embedded lists of attribute-node pairs). The only advantage we see for this kind of representation is its proximity to a particular kind of feature structure implementation.

We adopt an SGML markup for typed feature structures originally developed by the Text Encoding Initiative (TEI) which is very compact and seems to be widely accepted, e.g. also in the Tree Adjoining Grammar community (Issac 1998). Langendoen and Simons (1995) give an in-depth justification for the naming and structure of a feature structure DTD. We will focus here on the feature structure DTD subset that is able to encode the basic data structures of deep systems such as LKB (Copestake 1999), PET (Callmeier 2000), PAGE, or the shallow system SProUT (Becker et al. 2002) which have a subset of TDL (Krieger and Schäfer 1994) as their common basic formalism[2]:

```
<?xml version="1.0" ?>
<!-- minimal typed feature structure DTD -->
<!ELEMENT FS ( F* ) >
<!ATTLIST FS type  NMTOKEN #IMPLIED
             coref NMTOKEN #IMPLIED >
<!ELEMENT F  ( FS ) >
<!ATTLIST F  name  NMTOKEN #REQUIRED >
```

The FS tag encodes typed Feature Structure nodes, F encodes Features. Atoms are encoded as typed Feature structure nodes with empty feature list. An important

[2] Encoding of type hierarchies or other possibly system or formalism-specific definitions are of course not covered by this minimal DTD.

point is the encoding of coreferences (reentrancies) between feature structure nodes which denote structure sharing. For the sake of symmetry in the representation/DTD, we do not declare the coref attribute as ID/IDREF, but as NMTOKEN.
An application of WHAT access or transformation of deep annotation would be to specifiy a feature path under which a value (type, atom, or complex FS) is to be returned. The problem here are the coreferences which must be dereferenced at every feature level of the path. A general solution is to recursively dereference all nodes in the path.
We give only a limited example here, a query to access output of the SProUT system. It returns the value (type) of a feature somewhere under the specified attribute in a disjunction of typed feature structures, assuming that we are not interested here in structure sharing between complex values.

```
<query name="getValue.fs.attr" component="de.dfki.lt.sprout">
 <xsl:param name="disj"/>
 <xsl:param name="attr"/>

 <xsl:template match='DISJ[$disj]'>
  <xsl:variable name="node" select='.//F[@name=$attr]/FS'/>
    <xsl:choose>
      <xsl:when test="$node/@type">
        <xsl:value-of select="$node/@type"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:if test="$node/@coref">
          <xsl:call-template name="deref">
             <xsl:with-param name="coref"
                            select="$node/@coref"/>
          </xsl:call-template>
        </xsl:if>
      </xsl:otherwise>
    </xsl:choose>
  <xsl:apply-templates/>
 </xsl:template>

 <xsl:template name="deref">
  <xsl:param name="coref"/>
  <xsl:for-each select=".//FS[@coref=$coref]">
     <xsl:if test='@type'>
        <xsl:value-of select="@type"/>
     </xsl:if>
  </xsl:for-each>
 </xsl:template>
</query>
```

To complete the picture of abstraction through WHAT queries, we can imagine that the same types of query are possible to access e.g. the same morphology information in both shallow and in deep annotation, although their representation within the annotation might be totally different.

3.4 Efficiency of XSLT Processors

Processing speed of current XSLT engines on large input documents is a problem. Many XSLT implementations lack efficiency (for an overview cf. xmlbench.sourceforge.net). Although optimization is possible (e.g. through DTD specification, indexing etc.), this is not done seriously in many implementations. However, there are several WHAT-specific solutions that can help making queries faster. A pragmatic one is pre-editing of large annotation files. An HPSG parser e.g. focuses on one sentence at a time and does not exceed the sentence boundaries (which can be determined reliably by shallow methods) so that it suffices to split shallow XML input into per-sentence annotations in order to reduce processing time to a reasonable amount.
Another solution could be packing several independent queries into a 'prepared statement' in one stylesheet. However, as processing speed is mainly determined by the size of the input document, this does not speed up processing time substantially.
WHAT implementations are free to be based on DOM trees or plain XML text input (strings or streams). DOM tree representations are used by XSLT implementations such als libxml/libslt for C/Perl/Python/TCL or Xalan for Java. Hence, DOM implementations of WHAT are preferable in order to avoid unnecessary XML parsing when processing multiple WHAT transformations on the same input and thus help to improve processing speed.
As in all programming language, there a multiple solutions for a problem. An XSL profiling tool (e.g. xsltprofiler.org) can help to locate inefficient XSLT code.

3.5 Related Work

As argued in Thompson and McKelvie (1997), standoff annotation is a viable solution in order to cope with the combination of multiple overlapping hierarchies and the efficiency problem of XML tree modification for large annotations.
Ide (2000) gives an overview of NLP-related XML core technologies that also strives XSLT.
We adopt the pragmatic view of Carletta et al. (2002), who see that computational linguistics greatly benefits from general XMLification, namely by getting for free standards and advanced technologies for storing and manipulating XML annotation, mainly through W3C and various open source projects. The trade-off for this benefit is a representation language somewhat limited with respect to linguistic expressivity.
NiteQL (Evert and Voormann 2002) can be seen as an extension to XPath within XSLT, has a more concise syntax especially for document structure-related expressions and a focus on timeline support with specialized queries (for speech annotation). The query language in general does not add expressive power to XSLT and the implementation currently only supports Java XSLT engines.
Because of unstable standardization and implementation status, we did not yet make use of XQuery (Boag et al. 2002). XQuery is a powerful, SQL-like query language on XML documents where XPath is a subset rather than a sublanguage as in XSLT.

3.6 Advantages of WHAT

WHAT is

- based on standard W3C technology (XSLT)
- portable. As the programming language-specific wrapper code is relatively small, WHAT can easily be ported to any programming language for which an XSLT engine exists. Currently, WHAT is implemented in Java (JAXP/Xalan) and C/C++ (Gnome libxml/libxslt). Through libxml/libxslt, it can also easily be ported to Perl, Python, Tcl and other languages
- easy to extend to new components/DTDs. This has to be done only once for a component through XSLT query library definitions, and access will be available immediately in all programming languages for which a WHAT implementation exists
- powerful (mainly through XPath which is part of XSLT).

WHAT can be used

- to perform computations and complex transformations on XML annotation,
- as uniform access to abstract from component-specific namings and DTD structure, and
- to exchange results between components (e.g., to give non-XML-aware components access to information encoded XML annotation),
- to define application-specific architectures for online *and* offline processing of NLP XML annotation.

4 Conclusion and Future Work

We have presented an open, flexible and powerful infrastructure based on standard W3C technology for the online and offline combination of natural language processing components, with a focus on, but not limited to, hybrid deep and shallow architectures.

The infrastructure is part of the Whiteboard architecture and is employed and will be continued in several successor projects. The infrastructure is well suited for rapid prototyping of hybrid NLP architectures as well as for developing NLP applications, and can be used to both access NLP XML markup from programming languages and to compute or transform it.

Because WHAT is an open framework, it is worth considering XQuery as a future extension to WHAT. Which engine to ask, an XSLT or an XQuery processor, could be coded in each <query> element of the WHAT template library.

WHAT can be used to translate to the ingenious Thistle tool (Calder 2000) for visualization of linguistic analyses and back from Thistle in editor mode, e.g. for manual, graphical correction of automatically annotated texts for training etc.

A proximate approach is to combine WHAT with SDL (Krieger 2003) to declaratively specify WHAT-based NLP architectures (pipelines, loops, parallel transformation) that can be compiled to Java code.

The proximity to W3C standards suggests using WHAT directly for transformation of NLP results into application-oriented (W3C) markup, or to use W3C markup (e.g. RDF) for semantic web integration in NLP, VoiceXML, etc.

5 Acknowledgements

I would like to thank my collegues, especially Anette Frank, Bernd Kiefer, Hans-Ulrich Krieger and Günter Neumann, for cooperation and many discussions.

This work has been supported by a grant from the German Federal Ministry of Education and Research (FKZ 01IW002).

This document was generated partly in the context of the DeepThought project, funded under the Thematic Programme User-friendly Information Society of the 5th Framework Programme of the European Community – (Contract N° IST-2001-37836). The author is solely responsible for its content, it does not represent the opinion of the European Community and the Community is not responsible for any use that might be made of data appearing therein.

6 References

Markus Becker and Anette Frank. 2002. *A Stochastic Topological Parser of German.* Proceedings of COLING-2002, pp 71-77, Taipei.

Markus Becker, Witold Drożdżyński, Hans-Ulrich Krieger, Jakub Piskorski, Ulrich Schäfer, Feiyu Xu. 2002. *SProUT - Shallow Processing with Typed Feature Structures and Unification*. Proceedings of the International Conference on NLP (ICON 2002). Mumbai, India.

Scott Boag, Don Chamberlin, Mary F. Fernandez, Daniela Florescu, Jonathan Robie and Jérôme Siméon. 2002. *XQuery 1.0: An XML Query Language*. http://www.w3c.org/TR/xquery

Chris Brew, David McKelvie, Richard Tobin, Henry Thompson and Andrei Mikheev. 2000. *The XML Library LT XML. User documentation and reference guide.* LTG. University of Edinburgh.

Joe Calder. 2000. *Thistle: Diagram Display Engines and Editors.* Technical report. HCRC. University of Edinburgh.

Ulrich Callmeier. 2000. *PET - A platform for experimentation with efficient HPSG processing techniques*. Natural Language Engineering, 6 (1) (Special Issue on Efficient Processing with HPSG: Methods, systems, evaluation). Editors: D. Flickinger, S.Oepen, H. Uszkoreit, J. Tsujii, pp. 99 – 108. Cambridge, UK: Cambridge University Press.

Jean Carletta, David McKelvie, Amy Isard, Andreas Mengel, Marion Klein, Morten Baun Møller. 2002. *A generic approach to software support for linguistic annotation using XML.* Readings in Corpus Linguistics, ed. G. Sampson and D. McCarthy, London and NY: Continuum International.

James Clark (ed.). 1999. *XSL Transformations (XSLT)* http://www.w3c.org/TR/xslt

James Clark and Steve DeRose (eds.). 1999. *XML Path Language (XPath)* http://www.w3c.org/TR/xpath

Anne Copestake. 1999. *The (new) LKB system*. ftp://www-csli.stanford.edu/~aac/newdoc.pdf

Berthold Crysmann, Anette Frank, Bernd Kiefer, Hans-Ulrich Krieger, Stefan Müller, Günter Neumann, Jakub Piskorski, Ulrich Schäfer, Melanie Siegel, Hans Uszkoreit, Feiyu Xu. 2002. *An Integrated Architecture for Shallow and Deep Processing*. Proceedings of ACL-2002, Philadelphia, PA.

Stefan Evert with Holger Voormann. 2002. *NITE Query Language.* NITE Project Document. Stuttgart.

Anette Frank, Markus Becker, Berthold Crysmann, Bernd Kiefer and Ulrich Schäfer. 2003. *Integrated Shallow and Deep Parsing*. Submitted manuscript.

Claire Grover, Ewan Klein, Alex Lascarides and Maria Lapata. 2002. *XML-based NLP Tools for Analysing and Annotating Medical Language*. Proceedings of the Second International Workshop on NLP and XML (NLPXML-2002). Taipei.

Nancy Ide. 2000. *The XML Framework and its Implications for the Development of Natural Language Processing Tools*. Proceedings of the COLING Workshop on Using Toolsets and Architectures to Build NLP Systems, Luxembourg.

Nancy Ide and Laurent Romary. 2001. *A Common Framework for Syntactic Annotation.* Proceedings of ACL-2001. pp. 298-305. Toulouse.

Fabrice Issac. 1998. *A Standard Representation Framework for TAG.* In Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4), Philadelphia, PA.

Hans-Ulrich Krieger. 2003. *SDL – A Description Language for Specifying NLP Systems.* DFKI Technical Report. Saarbrücken.

Hans-Ulrich Krieger and Ulrich Schäfer. 1994. *TDL - A Type Description Language for Constraint-Based Grammars.* Proceedings of COLING-94. Vol. 2 pp. 893-899, Kyoto.

D. Terence Langendoen and Gary F. Simons. 1995. *A rationale for the TEI recommendations for feature-structure markup*. Computers and the Humanities 29(3). Reprinted in Nancy Ide and Jean Veronis, eds. The Text Encoding Initiative: Background and Context, pp. 191-209. Dordrecht: Kluwer Acad. Publ.

Jakub Piskorski and Günter Neumann. 2000. *An Intelligent Text Extraction and Navigation System*. In proceedings of 6th RIAO-2000, Paris.

Carl J. Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. Chicago: University of Chicago Press.

Manfred Sailer and Frank Richter. 2001. *Eine XML-Kodierung für AVM-Beschreibungen* (in German). In Lobin H. (ed.) Proceedings of the Annual Meeting of the Gesellschaft für linguistische Datenverarbeitung, Giessen. pp. 161 – 168.

Helmut Schmid. 2000. *LoPar: Design and Implementation*. Arbeitspapiere des Sonderforschungsbereiches 340, No. 149. University of Stuttgart.

Wojciech Skut and Thorsten Brants. 1998. *Chunk tagger – statistical recognition of noun phrases*. In Proceedings of the ESSLLI Workshop on Automated Acquisition of Syntax and Parsing. Saarbrücken.

Henry S. Thompson and David McKelvie. 1997. *Hyperlink Semantics for standoff markup of read-only documents.* In Proc SGML EU 1997.