The DeepThought Core Architecture Framework

Ulrich Callmeier[1], Andreas Eisele[1], Ulrich Schäfer[2], Melanie Siegel[1]

[1]Computational Linguistics Department
Saarland University
P.O.Box 151150
D-66041 Saarbrücken, Germany
{uc,eisele,siegel}@coli.uni-sb.de

[2]Language Technology Lab
German Research Center for Artificial Intelligence (DFKI GmbH)
Stuhlsatzenhausweg 3
D-66123 Saarbrücken, Germany
ulrich.schaefer@dfki.de

Abstract

The research performed in the DeepThought project aims at demonstrating the potential of deep linguistic processing if combined with shallow methods for robustness. Classical information retrieval is extended by high precision concept indexing and relation detection. On the basis of this approach, the feasibility of three ambitious applications will be demonstrated, namely: precise information extraction for business intelligence; email response management for customer relationship management; creativity support for document production and collective brainstorming. Common to these applications, and the basis for their development is the XML-based, RMRS-enabled core architecture framework that will be described in detail in this paper. The framework is not limited to the applications envisaged in the DeepThought project, but can also be employed e.g. to generate and make use of XML standoff annotation of documents and linguistic corpora, and in general for a wide range of NLP-based applications and research purposes.

Introduction

The challenges of the knowledge society cannot be met without getting at the contents of the vast volume of digital information. The concept of a semantic web is a viable vision; hoping, however, that the semantic structuring of such large volumes of unstructured information can be achieved by human authors or editors, is rather naive. It is therefore necessary to find solutions for natural language processing that on the one hand, output precise and informative semantic information and are, on the other hand, robust and efficient.

The idea of DeepThought is to preserve the advantages of shallow processing, namely robustness and efficiency, while adding more accuracy and depth in a controlled fashion at places where the application has a real demand for such increase in semantic analysis. The goal is to provide a system that combines different types of linguistic processing and that can be used for applications of different aims in a flexible way. E.g., information extraction would need the detection of relevant types of information, not full text understanding. Shallow processing enriches a text with XML annotations (PoS, phrases, named entities, simple relations). Deep processing is only called at places where shallow analysis hypothesizes relevant relations but cannot detect or select the correct relations.

As the project aims at evaluating the idea of combining different types of linguistic processing modules by three different applications, the commonly used core system must be efficient, robust and flexible.

Heart of Gold: A Common Architecture for Applications

When combining different types of NLP modules and their information output in a common architecture, it is useful to provide a common "language" for the module's output. The advantage of such an approach is obvious: Modules can communicate with each other, without the need for output compilation or matching. A well defined output (that was at first available for HPSG processing modules) can be guaranteed also for modules of different granularity of processing.

RMRS (robust minimal recursion semantics; Copestake 2003) has been chosen as the common interchange format. The basic idea is to view the information modules, e.g. a PoS tagger, deliver as an underspecified form of the semantics that deep linguistic parsing delivers.

The DeepThought core architecture framework "Heart of Gold" (HoG) provides a uniform and flexible infrastructure for building applications that use and combine RMRS-based (and other XML-based) natural language processing components. The core architecture is implemented in Java, but components and applications can be written in other programming languages. The system implemented so far builds on existing components like PET (Callmeier 2000) for highly efficient HPSG parsing, SProUT (Drożdżyński et al. 2004) for shallow named entity recognition, RASP (Briscoe and Carroll 2002) for statistical parsing, and others.

The main design goals and features of the architecture framework are:

- The integration of NLP components is flexible and fits the needs of different applications.
- The application interface is simple to allow for easy usage of the HoG by applications.
- RMRS (XML-encoded) is the uniform representation language.
- The HoG is open to other XML standoff annotation formats.
- Non-RMRS-outputting NLP components are integrated through annotation transformation.
- An annotation database is used for the storage and retrieval of computed linguistic analyses.
- HoG is a network-enabled architecture with distributed components.
- It provides a lightweight, platform- and programming language-independent communication using XML-RPC.
- It is based on current technology, such as XML, XML:DB, XPath, XML-RPC, XSLT.

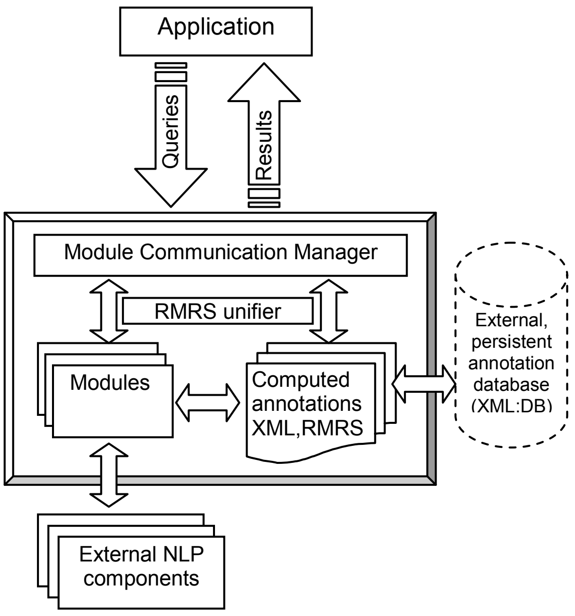
Application
Queries
Results
Module Communication Manager
RMRS unifier
Modules
Computed annotations XML,RMRS
External, persistent annotation database (XML:DB)
External NLP components

Figure 1: The HoG Core Architecture Framework

MoCoMan – The Module Communication Manager

The core architecture framework consists of a Module Communication Manager (MoCoMan) which mediates between the applications and the annotation-producing NLP components (Figure 1). MoCoMan receives a request (text documents, sentences) from an application, sends it to the configured components, receives their analysis results, and returns the results (combined or separately) back to the application. MoCoMan is assisted by an RMRS selector and unifier that combines the results of the components, and an optional annotation database for the persistent storage of computed analyses. MoCoMan is also responsible for the order in which the components are triggered. The idea is to have pluggable strategies depending on configured components and applications. Dynamic parameters like time constraints (e.g. for applications where speech is involved) might also come into play. The implemented default strategy is to let the application specify the depth of desired analysis with the query, and trigger all modules starting from the shallowest (e.g. tokenizer) up to the requested depth.

Modules and Components

Initially, a DeepThought application starts an instance of the core architecture MoCoMan with a configuration setting for the required components; parts of the module configuration facility are taken from the Memphis project (Kasper et al. 2004). MoCoMan then starts (or remotely connects to) the appropriate components, which are typically existing NLP software. From the viewpoint of MoCoMan, components are modules. I.e., in order to integrate a new component in the DeepThought architecture, a module subclass must be implemented and provide an interface to the underlying component. Because a component may be implemented in a language other than Java, there is a generic XML-RPC module class defined that can be used to connect to foreign language components, possibly on a remote server. Modules are also responsible for RMRS translation of non-RMRS-aware components.

Annotation Transformation

For the integration of non-RMRS-aware components, XSLT can be employed to transform component-specific XML output, e.g., of a chunker or a named entity recognition component, into the RMRS format. An example for the integration of a component in this way is the SProUT module that uses XSLT transformations of the XML-encoded typed feature structure output of the named entity grammars along the lines of Schäfer (2003) to generate an XML representation conforming to the RMRS DTD (Copestake 2003).

Session and Annotation Management

MoCoMan provides a session management, so that different input sessions with multiple input documents (texts) can be referenced (Figure 2). MoCoMan manages a collection of sessions for an application, where a session consists of a collection of annotations (each collection corresponds to one input document), that contain computed standoff annotations. Annotation collections and annotations are referenced through context-unique IDs. Sessions, annotation collections and computed annotations can optionally be stored in an XML annotation database.

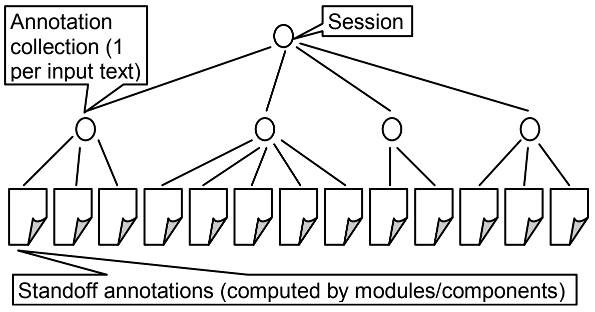
Annotation collection (1 per input text)
Session
Standoff annotations (computed by modules/components)

Figure 2: Multiple annotation collections in a session

Query Parameters for NLP Analysis

After the system configuration is finished, queries can be passed to the MoCoMan. Query parameters comprise an input text, the depth of requested analysis and the token range for analysis in the text. MoCoMan passes the query to the modules that are configured in the architecture instance and that are appropriate for the requested depth of analysis. The RMRS annotations computed by the modules are then returned to the application.

Metadata

Metadata on date, time, source, processing parameters, processing options and component-specific configurations of the producing component are stored together with the created annotations (Figure 3). This allows to precisely reconstruct the environment under which an annotation was produced. This is an important feature if HoG is used to create linguistically annotated texts for permanent storage.

```
<metadata>
  <id>
    <entry name="created" value="2004-03-04 15:23:15"/>
    <entry name="processingtime" value="00:00,90"/>
    <entry name="sessionid" value="session1"/>
    <entry name="acid" value="collection1"/>
    <entry name="component" value="Sprout"/>
  </id>
  <conf>
    <entry name="sprout.outputpath" value="OUT"/>
    <entry name="sprout.stylesheet"
           value="enamex2rmrs.xsl"/>
    <entry name="module.name" value="Sprout"/>
    <entry name="module.depth" value="1"/>
    <entry name="module.language" value="en"/>
    <entry name="module.rootelement" value="SPROUTPUT"/>
  </conf>
</metadata>
```

Figure 3: Metadata (abbreviated) with information on the generated annotation and module configuration

XML Annotation Database

If a query that has already been computed (i.e., a known input text with the same query parameters) is passed to the MoCoMan, then the pre-computed result is returned. This can be done on the basis of the stored data. Moreover, the DeepThought core architecture framework can optionally provide a database for XML annotation storage. The main purpose is persistent storage of computed annotations for the automatic creation or enrichment of linguistic corpora etc. The annotation database interface uses XML:DB which is a vendor-independent interface to native XML databases.[1]

The XML database interface supports the organization of XML annotation, reflecting the session and annotation collection tree hierarchy of MoCoMan. Standard operations like inserting and deleting collections and XML annotations, and a standardized query language based on XPath are supported. Existing annotations can also be modified using the XUpdate query language.

An important feature of the XML databases is indexing of XML document elements with respect to efficient retrieval. Depending on the structure of the annotation, indexers can be defined through the database interface. This should be done when integrating new components and can be stored as part of the Module configuration, which in turn is part of the annotation metadata. The current status of the annotation database is experimental. Once the RMRS DTD becomes stable, and if large text corpora pre-annotated with RMRS are in focus, a flexible full text search engine like Jakarta Lucene could be considered, with RMRS-specific indexing and RMRS-specific query types defined. The expected advantage is increased performance, traded for a loss in flexibility compared to the powerful XML:DB query framework.

Communication with the HoG

The core architecture framework comprising MoCoMan, modules and the annotation database interface is entirely written in Java and hence, the direct way to communicate is by calling the Java API.

However, a second, open way of communication is supported, namely by XML-RPC. MoCoMan can act as an XML-RPC server. Non-Java applications (e.g. written in Python, as the demonstrator for creativity support for document production and collective brainstorming in DeepThought) and also non-Java components can connect even remotely via network, and hence easily implement a distributed architecture. There are also (abstract) Java classes that can be used to implement clients that remotely access the MoCoMan XML-RPC server.

[1] The current, experimental implementation uses Apache Xindice, but other (e.g. commercially available) XML databases supporting XML:DB could be used instead.

Components integrated so far

Various NLP components are integrated into the core architecture framework. JTok (developed at DFKI by Jörg Steffen) is used for the purpose of tokenization and sentence boundary recognition (it can be easily adapted to other languages). JTok is implemented in Java. SProUT (Drożdżyński et al. 2004), a multilingual, shallow processing component that combines finite state and type feature structure technology and includes morphologic resources and named entity grammars for ten languages, is integrated as well. RMRS output is gained with XML transformations. SProUT is implemented in Java. RASP (Briscoe and Carroll 2002) is a robust statistical parser for English, which is developed in C and LISP on the basis of ANLT. RASP delivers RMRS output of medium NLP depth. PET is a highly efficient deep parser for HPSG grammars. It is developed in C and C++ at Saarland University and DFKI (Callmeier 2002). Using LKB (Copestake et al. 2003) implementations, PET parsing delivers RMRS output from HPSG grammars (cf. Flickinger 2002 for English; Crysmann 2003, Frank et al. 2003 for German).

We will further integrate Chunkie/TnT (Skut and Brants 1998), which will need mechanisms for the generation of RMRS output.

Combining RMRS output of different components

Combining the information computed by the different components is crucial for the benefit of HoG-based applications. We give a short example for the sentence "*Where is the Nokia 8890 used?*". The named entity recognition module SProUT gives RMRS output (in XML format in Figure 4) for the named entity "*Nokia 8890"*.

```
<rmrs cfrom="0" cto="12">
  <label vid="1"/>
  <ep>
    <label vid="1"/>
    <gpred>product_rel</gpred>
    <var sort="x" vid="2"/>
  </ep>
  <rarg>
    <label vid="1"/>
    <rargname>CARG</rargname>
    <constant>Nokia_8890</constant>
  </rarg>
</rmrs>
```

Figure 4: Shallow RMRS generated by SproUT for the named entity "Nokia 8890"

The HPSG processing, using the NE information, delivers RMRS output as well (which is represented in Figure 5 without XML annotation, due to the amount of space).

```
h1
 int_m_rel(h1,h3)
          PSV(h1,x4)
          TPC(h1,e5)
          ARG1(h1,u19)
          ARG2(h1,x4)
 prpstn_m_rel(h3,h6)
          qeq(h6,h9)
 unspec_loc_rel(h9,e5)
          ARG1(h9,e2)
          ARG2(h9,x10)
          ING(h9,h1)
 place_rel(h11,x10)
 which_q_rel(h12,x10)
          RSTR(h12,h13)
          BODY(h12,h14)
          qeq(h13,h11)
 _the_q(h15,x4)
          RSTR(h15,h17)
          BODY(h15,h16)
          qeq(h17,h18)
 named_n_rel(h18,x4)
          CARG(h18,Nokia_8890)
 _use_v_1(h1,e2)
          PSV(h1,x4)
          TPC(h1,e5)
          ARG1(h1,u19)
          ARG2(h1,x4)
```

Figure 5: RMRS for the sentence "where is the Nokia 8890 used?", produced by the English HPSG grammar

The relation printed in bold corresponds to the named entity RMRS gained from the SProUT module. Different strategies of RMRS combination can be used, where a simple approach is to use token span information to integrate information about words that are unknown to one component, but recognized by another (Nokia 8890 in the example). Subtype information from the type hierarchy can be exploited in order to find matching relations. In the example, **`product_rel`** from the shallow named entity recognition component is a subtype of the **`named_n_rel`** in the deep RMRS result.

Conclusion

The DeepThought core architecture framework (HoG) for the combination of natural language processing modules allows relatively simple connection and inclusion of new modules and can be used by different NLP applications in a flexible way. The centre of HoG is the module and communication manager (MoCoMan), which organizes the information flow between modules and applications. The underlying idea in NLP module combination is the usage of RMRS as a common output format. Next steps in the ongoing project DeepThought will be various evaluations: It will be evaluated with what module combination each of the applications work best and whether the approach indeed combines the advantages of different types of NLP components when used with applications.
A further application of the architecture framework is the creation of large, richly NLP-annotated texts as a basis for question answering and similar functionality.

Acknowledgements

We would like to thank Hans Uszkoreit, Jörg Steffen, Özgür Demir, Robert Barbey, Atsuko Shimada, Matthias Grossklos and our DeepThought project partners (http://www.project-deepthought.net)
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 Union – (Contract N° IST-2001-37836). The authors are solely responsible for its content, it does not represent the opinion of the European Union and the Union is not responsible for any use that might be made of data appearing therein. This work has partly been supported by a grant from the German Federal Ministry of Education and Research (FKZ 01 IW C02).

References

Drożdżyński, W., Krieger, H.-U., Piskorski, J., Schäfer, U., and Xu, F. (2004). Shallow Processing with Unification and Typed Feature Structures – Foundations and Applications. In: *Künstliche Intelligenz* (1). http://www.kuenstliche-intelligenz.de/archiv/2004_1/sprout-web.pdf.

Briscoe, E. and Carroll, J. (2002) Robust accurate statistical annotation of general text. In: *Proceedings of the 3rd International Conference on Language Resources and Evaluation*, Las Palmas, Gran Canaria (pp. 1499-1504).

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

Copestake, A. (2003). *Report on the Design of RMRS.* Technical Report D1.1b, University of Cambridge, UK.

Crysmann, B. (2003). On the efficient implementation of German verb placement in HPSG, *Proceedings of RANLP 2003*, Borovets, Bulgaria.

Frank, A., Becker, M., Crysmann, B., Kiefer, B. and Schäfer, U. (2003). Integrated Shallow and Deep Parsing: TopP meets HPSG. In: *Proceedings of ACL-2003* (pp. 104-111). Sapporo, Japan.

Flickinger, D. (2002). On building a more efficient grammar by exploiting types. In Stephan Oepen, Dan Flickinger, Jun'ichi Tsujii and Hans Uszkoreit (eds.) *Collaborative Language Engineering*, Stanford: CSLI Publications, pp. 1-17.

Kasper, W., Steffen, J., Piskorski, J., Buitelaar, P. (2004). Integrated Language Technologies for Multilingual Information Services in the MEMPHIS Project. In: *Proceedings of LREC-2004*, Lissabon, Portugal.

Schäfer, U. (2003). WHAT: An XSLT-based Infrastructure for the Integration of Natural Language Processing Components. In: *Proceedings of the Workshop on the Software Engineering and Architecture of Language Technology Systems, HLT-NAACL03* (pp. 9–16). Edmonton, Canada.

Skut, W. and Brants, T. (1998). Chunk tagger – statistical recognition of noun phrases. In: *Proceedings of the ESSLLI Workshop on Automated Acquisition of Syntax and Parsing*. Saarbrücken.