# Hybrid Robust Deep and Shallow Semantic Processing for Creativity Support in Document Production

**Hans Uszkoreit, Ulrich Callmeier, Andreas Eisele, Ulrich Schäfer, Melanie Siegel**
Computational Linguistics Department
Saarland University
P.O.Box 151150
D-66041 Saarbrücken, Germany
{hansu,uc,eisele,siegel}@coli.uni-sb.de

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

**Jakob Uszkoreit**
acrolinx GmbH, Novalisstr. 12, D-10115 Berlin, Germany
jakob@acrolinx.de

## Abstract

The research performed in the DeepThought project (http://www.project-deepthought.net) aims at demonstrating the potential of deep linguistic processing if added to existing shallow methods that ensure robustness. Classical information retrieval is extended by high precision concept indexing and relation detection. We use this approach to demonstrate the feasibility of three ambitious applications, one of which is a tool for creativity support in document production and collective brainstorming[1]. This application is described in detail in this paper. Common to all three applications, and the basis for their development is a platform for integrated linguistic processing. This platform is based on a generic software architecture that combines multiple NLP components and on robust minimal recursive semantics (RMRS) as a uniform representation language.

## 1 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.

Deep NLP systems 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 NLP systems do not attempt to achieve such an exhaustive linguistic analysis. They are designed 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 and more robust than deep NLP systems.

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.

The idea of DeepThought is to preserve the advantages of shallow processing, i.e., robustness and efficiency, while adding more accuracy and depth in a controlled fashion at places

[1] The two other applications investigated within the project are: precise information extraction for business intelligence and email response management for customer relationship management.

where the application has a real demand for such increase in semantic analysis (Uszkoreit, 2002). The goal is to provide a system that combines different types of linguistic processing and that can be used for various types of applications in a flexible way. One of these is the novel type of application described in this paper. This application supports creative document production. To this end it combines functionality for document production and editing with advanced semantic information retrieval and question answering. We have designed and implemented a fully functional prototype. In this paper we describe the prototype and the methodology developed for combining the respective virtues of different processing methods. Using some examples we will illustrate the collaboration of NLP components in the new architecture.

## 2 Creative Authoring Support

When new ideas are produced, discussed, and presented, a large proportion of the effort goes into looking up and combining existing pieces of information. The reasons for this are simple: (i) the completely new ideas and facts only constitute a tiny fraction of the total content and (ii) we cannot keep all the cited facts and sources in our memory.

If the lookup of facts, sources, references, pictures can be performed with greater ease and speed, the creative process gains immensely in efficiency. If the authors are not constantly interrupted by searches and if they can spend more time on the truly creative portions of the task, the quality of the results will also considerably improve.

Everyone who has ever authored a document remembers the numerous disruptions in situations when information is missing and it has to be looked up. Only a few years ago, one had to consult books, journals, and archives to find the required data. Today, much of the lookup can be done on the Internet or on other electronically accessible repositories. Nevertheless, any lookup is disruptive. Experienced writers do not stop the creative process each time some piece of information is missing, but rather insert a note for later lookup. Our basic idea is that this lookup can be performed automatically. This can happen while the author continues to write, or even after hours. While search and presentation are automated, the selection and the actual creative tasks are left to the human author.

The need for looking up information also occurs when complex charts or other figures are composed. Only in rare cases does the author really need to draw the pictorial elements from scratch. Today, symbols, icons or other graphical elements are readily available in clip art collections, graphics archives, or on the web. Again, one can delay the search for missing elements by inserting a dummy shape such as an empty rectangle or a circle together with a note.

### 2.1 An example

In a creative meeting, the participants collectively develop a marketing plan for mobile phones. The moderator stands in front of the group entering the contributions onto an electronic flipchart (e.g., a SmartBoard) by means of electronic pen and microphone. She might want to insert information about the functionality of a Nokia 8890 and – using her microphone – dictates the question 'Does the Nokia 8890 possess Bluetooth?' to the application and then pushes the button for 'search'. While the discussion continues the system searches for the answer. Whenever a search is completed, the question will turn in to green or red, depending whether an answer has been found or not. If a green query is clicked, a menu appears that lists the most highly ranked answer candidates. The answers contain links to their source, such that a browser window can be opened that contains more information about the topic (say, in this case, a web page on the features of Nokia 8890). Next, the moderator may want to insert a picture of the phone set on the flipchart . In this case, the analysed query is compared with analyses of natural language descriptions of pictures (such as 'this is a picture of the Nokia 8890'), and the best matches will be in the menu to choose.

## 3 Linguistic Challenges

The described application opens up a bag of challenges to linguistic processing. Answering

questions requires information of varying granularity. On the one hand, the analysis of query and possible answers must be robust. It may contain named entities, which requires more robust processing than can be provided by deep parsing. As we also allow speech input, the processing must be able to deal with spoken language and recognition errors. On the other hand, the analysis must be as exact as possible. We want to be able to account for negation scope and predicate-argument relations in more complex queries like "I want a picture of a Nokia phone, but not the Nokia 8890" or "show me a picture of the Nokia 8890 and a table of the features of the Siemens S55". Modification anchors are needed to decide, if in the case of "I want a large picture of the Nokia 8890" the user wants a large picture or a picture of the large phone. To account for robustness and exactness of analysis the machinery that processes queries and answers uses an intelligent combination of deep and shallow NLP processing modules as well as standard web-based QA systems as a fall-back strategy. Here we use AnswerBus (Zheng, 2002) for this purpose (http://www.answerbus.com).

The key idea to overcome the outlined problem is the integrated exploitation of linguistic components that allow analysis at different levels of granularity. In this way robustness and efficiency of shallow processing is combined with the increased accuracy provided by deep analysis. The integration is facilitated by the choice of a semantic representation language that allows flexibility in the level of detail that is specified. With the choice of robust mininal recursive semantics (RMRS) (Copestake, 2003), output from various existing modules, ranging from part-of-speech taggers over modules of intermediate depth such as RASP (Briscoe and Carroll, 2002) up to fine-grained syntactic analyses based on HPSG (Pollard and Sag, 1994) can be mapped into compatible representations.

It is then possible to reassemble partial output from multiple components into one coherent representation, which already means that the coverage of the integrated system will be better than for any of its parts.

Components can also be cascaded, so that entities that are recognized by shallow analysis (e.g. product names or named entities in general) do not need to be decomposed by deeper components. This increases the robustness of the system in cases where these entities contain parts that are not covered in the deeper linguistic specification. Furthermore, cascaded processing can reduce the amount of ambiguity the fine-grained analysis has to deal with[2]. Perhaps the simplest strategy to combine the components is to attempt analysis with the deepest relevant level, but fall back to a coarser level of granularity in case of failure.

Once a query has been entered by the user, it is sent to the linguistic core machine ('heart of gold'). From there, the search engine gets back the RMRS-annotated query. Using the repository of RMRS annotated texts, pictures and graphics, it extracts similar annotations and composes the result for presentation. It then sends the result to the text and presentation editor module via the application server.

# 4 Architecture of Creative Authoring Support Application

The Creative Authoring Support Application consists of the following main modules:

- An editor for text and graphics input, request sending functionality and information insertion functionality, as well as speech recognition interaction.
- A server hosting the application logic.
- An information search engine with the functionality of information extraction from RMRSs and interaction with the linguistic core machine and the stored annotated texts, graphics and pictures.
- A connection to a speech recognition system.

A schematic overview of the overall architecture is given in Figure 1.

[2]Robustness can be preserved even in cases where the shallow analysis contains errors. In a priority-based chart parser as the one in PET(Callmeier, 2000), an entity recognized by the shallow component can be given as a "recommendation" to the deep parser with a high priority, in a way that can still be overridden if it is not

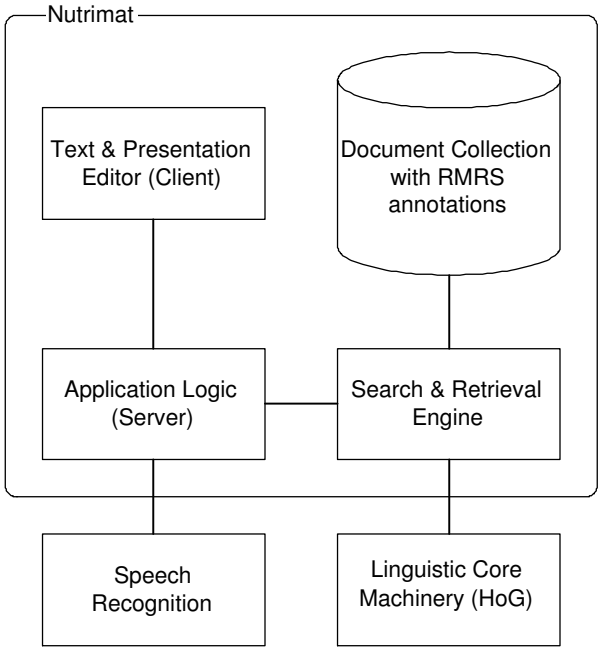


Figure 1: The Software Architecture for Creative Authoring.

The application uses a client-server architecture, where the client, implemented using Macromedia Flash, is usable in any common web browser via the network. The application server and the search engine have been implemented in Python. We provide two ways of connecting speech recognition to the system. The straightforward setup uses a client-side recognition engine, e.g. a dictation system that is installed on the user's machine. Speech input without any local installation is possible using a server-side recognition engine, where the audio signal is transmitted to the server and handed to the recognition module, as displayed in the diagram.

The information search is based on a collection of RMRS-annotated texts, pictures and graphics. The query is sent to the HoG and returned with RMRS annotation. Based on this annotation, a search on the RMRS-annotated text, pictures and graphics is performed, using information extraction techniques.

compatible with a spanning analysis of the context, cf. also (Crysmann et al., 2002; Frank et al., 2003).

```
<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 2: Shallow RMRS for the named entity "Nokia 8890".

When a search is initiated in the user interface through marking a text and pushing a search button, the marked search request changes colour while the search is in progress, and again once results are available. The query is sent to the application logic server, which in turn interacts with the search engine, and sends the query to the search engine, accompanied by query context and requested result types (pictures, texts or links). Search results can be texts, pictures or documents. They are annotated with a description (string), and a URL. They are presented to the user for selection in a pull-down menu.

### 4.1 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 gives RMRS output (in XML format in Figure 2) for the named entity "Nokia 8890".

The deep processing, using the NE information, delivers RMRS output as well (which is represented in Figure 3 without XML annotation, due to the amount of space).

The relation printed in bold corresponds to the named entity RMRS gained from the shallow module. Different strategies of RMRS combination can be used, where a simple approach is to use text span information to integrate information about words that are un-

```
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 3: RMRS for the sentence "where is the Nokia 8890 used?", produced by the deep grammar.

known 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.

# 5 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) (Callmeier et al., 2004) 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 and connected through XML-RPC. 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.

## 5.1 The Module Communication Manager

The core architecture framework consists of a Module Communication Manager (MoCoMan) which mediates between an application and the annotation-producing NLP components (Figure 4). MoCoMan receives a request (text documents, sentences) from an application, sends it to the configured components, receives their analysis results, and returns the results back to the application. The interface to the HoG allows requests containing the following parameters.

1. A language identifier for the language of the string to be analysed.
2. The text to be analysed. This can include context surrounding the query.
3. The text span to be analysed. This will mostly be from start to end.
4. Constraints about time, precision and number of readings. As queries should be reasonably short for this application, the

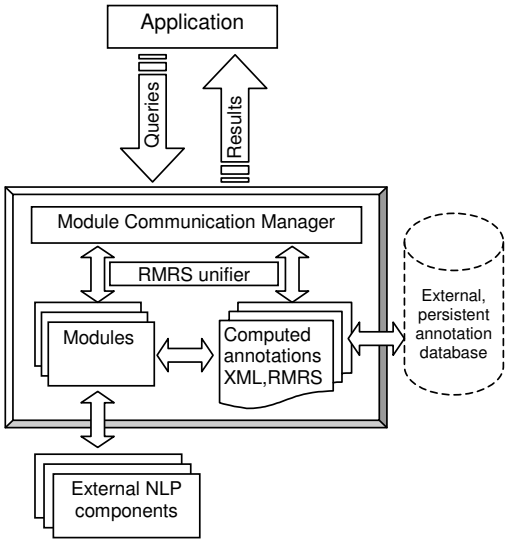


Figure 4: Core Architecture 'Heart of Gold' (HoG)

default constraints will be to get precise analysis and one (the best) reading.

An RMRS selector and unifier can be used to combine the results of the components. An optional annotation database supports the persistent storage of computed analyses. MoCoMan is also responsible for the order in which the components are triggered. 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, with a fallback to the previous component if no result was available from the component with the desired depth.

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. In other words, modules encapsulate and abstract from the real components. Modules are also responsible for RMRS translation of non-RMRS-aware components.

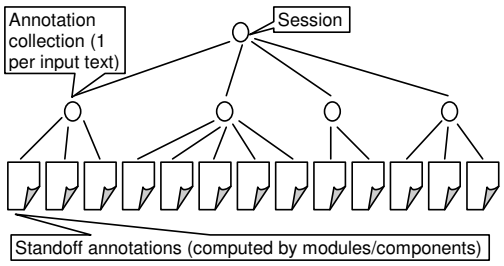


Figure 5: Session and Annotation Management

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). MoCoMan can act as an XML-RPC server. Non-Java applications and also non-Java components can connect via network, and hence easily implement a distributed architecture.

### 5.2 Session and Annotation Management

MoCoMan provides a session management, so that different input sessions with multiple input documents (texts) can be referenced (Figure 5). 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. Metadata on date, time, source, processing parameters, processing options and component-specific configurations of the producing component are stored together with the created annotations. This allows to precisely reconstruct the environment under which an annotation was produced.

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 data gained during a session. 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. The current, experimental implementation uses Apache Xindice, but other XML databases supporting XML:DB could be used instead. The underlying generic annotation database interface layer of MoCoMan can also be used to control other storage mechanisms, e.g. in connection with a full-text search engine for the annotations.

### 5.3 Integrated NLP Components

Various shallow and deep 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.

- The PoS tagger TnT and the chunker Chunkie for English and German (Skut and Brants, 1998) are extended in order to generate RMRS output.

- PET is a highly efficient deep parser for HPSG grammars. It is developed in C and C++ at Saarland University and DFKI (Callmeier, 2000). Using LKB (Copestake, 2002) implementations, PET parsing delivers RMRS output from HPSG grammars, cf. (Flickinger, 2002) for English; (Crysmann, 2003), (Frank et al., 2003) for German.

## 6 Conclusion

We have presented a new application for creativity support in document authoring. The user is assisted in composing a text, possibly enriched with pictures taken from a local repository or the web. Domain-specific questions and commands related to the content of the document can be answered on the basis of the repository, thus helping the user to perform the authoring task faster and with fewer disruptions.

The application uses robust semantic representation (RMRS) gained from a hybrid combination of deep and shallow NLP components. The application benefits from robustness and efficiency of the shallow components, as well as from increased accuracy provided by the deep HPSG parser.

The underlying XML-based, network-enabled architecture is open and generic, and can be used to integrate additional NLP components and build the foundation for various other applications. In combination with ontologies, the existing framework can be extended and form the basis for further challenging applications in the context of Question Answering and the Semantic Web.

DeepThought is an ongoing project. We will be able to present additional results and a demonstrator at the conference.

## 7 Acknowledgements

We would like to thank Bernd Kiefer, Jörg Steffen, Özgür Demir, Robert Barbey, Nayla Badr and our DeepThought project partners.

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 No

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

Edward J. Briscoe and John Carroll. 2002. Robust accurate statistical annotation of general text. In *Proceedings of LREC-2002*, pages 1499–1504, Las Palmas, Gran Canaria.

Ulrich Callmeier, Andreas Eisele, Ulrich Schäfer, and Melanie Siegel. 2004. The DeepThought core architecture framework. In *Proceedings of LREC-2004*, Lisbon, Portugal.

Ulrich Callmeier. 2000. PET — A platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering*, 6 (1):99 – 108.

Ann Copestake. 2002. *Implementing Typed Feature Structure Grammars*. CSLI publications, Stanford, CA.

Ann Copestake. 2003. Report on the design of RMRS. Technical Report D1.1b, University of Cambridge, Cambridge, UK.

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

Berthold Crysmann. 2003. On the efficient implementation of german verb placement in HPSG. In *Proceedings of RANLP-2003*, Borovets, Bulgaria.

Witold Drożdżyński, Hans-Ulrich Krieger, Jakub Piskorski, Ulrich Schäfer, and Feiyu Xu. 2004. Shallow processing with unification and typed feature structures — foundations and applications. *Künstliche Intelligenz*, 1:17–23. http://www.kuenstliche-intelligenz.de/archiv/2004_1/sprout-web.pdf.

Dan Flickinger. 2002. On building a more efficient grammar by exploiting types. In Dan Flickinger, Stephan Oepen, Hans Uszkoreit, and Jun-ichi Tsujii, editors, *Collaborative Language Engineering. A Case Study in Efficient Grammar-based Processing*, pages 1–17. CSLI Publications.

Anette Frank, Markus Becker, Berthold Crysmann, Bernd Kiefer, and Ulrich Schäfer. 2003. Integrated shallow and deep parsing: TopP meets HPSG. In *Proceedings of ACL-2003*, pages 104–111, Sapporo, Japan.

Walter Kasper, Jörg Steffen, Jakub Piskorski, and Paul Buitelaar. 2004. Integrated language technologies for multilingual information services in the MEMPHIS project. In *Proceedings of LREC-2004*, Lisbon, Portugal.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. University of Chicago Press, Chicago.

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

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

Hans Uszkoreit. 2002. New Chances for Deep Linguistic Processing. In *Proceedings of COLING 2002*, pages xiv–xxvii, Taipei, Taiwan.

Zhiping Zheng. 2002. AnswerBus question answering system. In *Human Language Technology Conference (HLT-2002)*, San Diego, CA.