Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document
D-93-03

DFKI Workshop on Natural Language Systems: Reusability and Modularity

Saarbrücken, October 23, 1992

Proceedings

Stephan Busemann, Karin Harbusch (Eds.)

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
D-6750 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, SEMA Group, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct *systems with technical knowledge and common sense* which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- Intelligent Engineering Systems
- Intelligent User Interfaces
- Computer Linguistics
- Programming Systems
- Deduction and Multiagent Systems
- Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Friedrich J. Wendl
Director

DFKI Workshop on Natural Language Systems Proceedings

Stephan Busemann, Karin Harbusch (Eds.)

DFKI-D-93-03

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITW-9002 0 und ITW-8901 8).

DFKI Workshop on Natural Language Systems: Reusability and Modularity

edited
by
Stephan Busemann (DISCO)
Karin Harbusch (WIP)

DFKI
Saarbrücken
October 23rd
1992

Contents

Preface

The DFKI internal workshop "Natural Language Systems" was held at the DFKI Saarbrücken on the 23rd of October. It was attended by about 20 participants not only from the DFKI but also from the University of the Saarland (e.g., the "Sonderforschungsbereich 314 — Künstliche Intelligenz und wissensbasierte Systeme", and the Computational Linguistics Department) and the "Institut zur Förderung der Angewandten Informationswissenschaft" (IAI).

As described below the program consisted of 10 talks of 20 minutes each with 5 minutes for clarification questions. Three additional discussion sections were planned to collect questions concerning modularity and reusability in other systems. These features were emphasized in the subtitle of the workshop in order to focus the discussion on these topics.

In respect of the goal of discussing in detail collaborations within the DFKI, we asked the speakers to touch on the following matters in their talks:

- Describe the general ideas of the approach, and detail the interfaces of the module presented.
- Discuss the system's coverage.
- Describe the status of development.
- Sketch relationships to other work in the DFKI, and assess prospects for collaboration with other projects.

As another way of highlighting relations between DFKI projects and focussing on prospects of cooperations, we ordered the talks to alternate those with an emphasis on representation and those emphasizing processing aspects.

To summarize the final discussion, a central topic of common interest between the DISCO and WIP groups was the translation of the Head-Driven Phrase Structure Grammar (HPSG) developed in DISCO into a Tree Adjoining Grammar (TAG) which can be used in the sentence generator TAG-GEN in WIP and potentially in the successor project VERBMOBIL.

Processing based on the DISCO grammar could be more efficient by compiling a "performance grammar" out of the high-level specification. A step in this direction can be seen in the translation of an HPSG grammar into a TAG grammar proposed by Robert Kasper from Ohio State University at the last TAG workshop (June 24th to 26th, 1992, University of Philadelphia). From the perspective of the TAG-GEN group within WIP, the question whether the resulting TAG grammar can be integrated into WIP's incremental sentence generator is essential; i.e., can the transformed HPSG grammar be used directly for incremental processing?

The question of how the HPSG principles can be translated automatically into TAG trees was answered during the discussion. It is apparently the case that the principles in

the DISCO grammar are sufficiently local to be transformed into the domain of locality in TAGs.

In the general discussion on reusability and modularity, one concern was consideration of concrete module reuse between projects. So for NLL, the semantic representation language developed in DISCO, strong evidence was presented that it could be adapted to WIP's purposes. WIP processing requires elaborate representation and manipulation tools for speech acts. One suggestion was that this kind of information can be integrated into NLL.

Another candidate for a reusable module is the presentation planner developed in the WIP project to generate multimodal documents, which is based on Rhetorical Structure Theory (RST). It was suggested to investigate an adaptation of this module for the dialogue planner in the DISCO system.

On a more general level, the question arose in discussion just what should count as a module. It was argued that the interface to a new application system should be encapsulated as a module. An example of a step towards the realization of this idea is the use of compiler techniques as realized for the NLL module. In the discussion the terms 'reusability' and 'modularity' were not restricted to software but were also applied to methods and algorithms in general.

The discussion section ended with a list of intended next steps towards collaboration:

- Work on the translation of HPSG into TAG will be initiated during the stay of Robert Kasper from November 2nd to 18th as guest of the DISCO and WIP projects.
- For NLL the extension to elaborated speech act representations will be tackled.
- WIP's presentation planner will be checked to see whether it can deal with the specific dialogue information needed in DISCO.

Stephan Busemann and Karin Harbusch

Workshop Programme

9.00	–	9.25	Klaus Netter **Structure and Coverage of the DISCO Grammar**
9.25	–	9.50	Judith Klein, Ludwig Dickmann, Abdel Kader Diagne, John and Nerbonne, Klaus Netter **DiTo — A Diagnostic Tool for Syntactic Analysis**
9.50	–	10.20	*Break*
10.20	–	10.45	Anne Kilger **Incremental Generation with Tree Adjoining Grammars in the WIP System**
10.45	–	11.10	Walter Kasper **Integration of Syntax and Semantics in Feature Structures**
11.10	–	11.35	Wolfgang Finkler **Effects of Incremental Output on Incremental Natural Language Generation**
11.35	–	12.00	**Discussion**
12.00	–	13.30	*Lunch*
13.30	–	13.55	Elisabeth André **An Extended RST Planner for the Generation of Multi–Modal Presentations**
13.55	–	14.20	John Nerbonne, Karsten Konrad, Ingo Neis, and Stephan Oepen **NLL — Tools for Meaning Representation**
14.20	–	14.35	**Discussion**
14.35	–	15.05	*Break*
15.05	–	15.30	Karin Harbusch **Lexical Choice Under Constraints**
15.30	–	15.55	Stephan Busemann **Configuration of Generation Systems**
15.55	–	16.20	Günter Neumann **Principles and the Current Status of the COSMA Architecture**
16.20	–	16.35	**Discussion**
16.35	–	17.00	*Break*
17.00	–	18.00	**Final Discussion**

Architecture and Coverage of the DISCO Grammar

Klaus Netter *

Abstract

In this paper we give a rough sketch of the German grammar that was developed in the DISCO project. The description also includes some characteristics of the grammar formalism and of the various processing components corresponding to different descriptive layers in the grammar.

1 General Characteristics

The DISCO grammar is a German grammar whose syntactic part was developed by K. Netter (with support by J. Nerbonne) and which has an integrated semantic representation developed by J. Nerbonne and W. Kasper [Ner92], [Kas93]. The style of the grammar follows very much the spirit of Head Driven Phrase Structure Grammar (HPSG) [PS87], [PS93]. However, it also incorporates insights from other grammar frameworks (e.g., categorial grammar) and extensions to the theory which are not yet part of standard HPSG. The grammar is implemented in a formalism called Type Description Language (TDL) which was developed by H.-U. Krieger and U. Schäfer [SK92].

The grammar provides interfaces to a morphological component and to a speech act recognition module which operates on syntactic and semantic information [HS93]. The feature structure representation of the semantic representation can be translated directly into the meaning representation language $\mathcal{NLL}$. Alternatively, the translation module can operate on the output of a speech act recognition module [NOD+93].

The grammar is at present mainly used for a system modelling discourse between cooperative agents. It provides the NL front end to the COSMA system (COoperative Scheduling Management Agent) whose application domain is appointment scheduling [NOS93].

2 Formalism

The grammar (along with the speech act recognition module and parts of the morphological component providing the grammar input) are written in the typed feature formalism TDL,

*The research underlying this paper was supported by a research grant, FKZ ITW 9002 0, from the German Bundesministerium für Forschung und Technologie to the DFKI project DISCO.

which incorporates the unification engine UDINE developed by R. Backofen. TDL is the exclusive formal device employed to specify grammar rules, lexical entries and all other linguistic knowledge relevant for the grammar. At present, specifications in TDL result in fully expanded feature structures which are then processed by various modules.

In TDL, typed feature structures can be defined through simple or multiple inheritance relations. TDL performs full type expansion at compile-time, i.e., if in a type definition a type inherits from other types or if the value of an attribute is restricted to a type, these types are replaced by the associated feature structures with only limited simplification.[1] In addition to the type hierarchy, TDL also provides parameterized templates as an integrated descriptive device. The parameter specifications of these templates can be exploited to specify feature structures which differ in only few (typically deeply embedded) values, as for example with classes of lexical entries.

Unification of feature structures in TDL and elsewhere in the system is executed by the unfier UDINE, which presumably is one of the most comprehensive unifiers so far implemented. It comprises full negation, including negation of co-references, and full disjunction. UDINE provides for so-called *distributed disjunction* through which disjunctive information can be kept as local as possible in the structural specification. The main advantage of distributed disjunction is that it helps to avoid the translation of structures containing disjunctions into a disjunctive normal form, which, given the size of structures in question, could lead to a serious efficiency problem. UDINE has a mechanism for treating values defined by *functional constraints*. Future extensions of the unifier will allow one to formulate and apply *preferences* in the processing of conjunctive and disjunctive feature specifications.

3 Levels of Processing

Linguistic specifications of the DISCO grammar are processed by four different modules, which, except for the morphology, were implemented by B. Kiefer:

- a *scanner* preprocessing the input string;
- a *morphological component* mapping strings into feature structures;
- a *lexical component* performing the lexicon lookup;
- the actual *parser* yielding as output a feature structure containing parallel morphological, syntactic, semantic and pragmatic information.

Scanner The scanner for the text input is implemented in LEX and YACC and prepares the string for further processing. Among its tasks are recognition of special characters, normalization of capitalization, recognition of sentence boundaries, and segmentation of the string into tokens which are then passed on to the morphology. The scanner can expand

[1] Partial or delayed type expansion, the incremental definition of types, negation, disjunctive types and partitioning are going to be available in the highly extended functionality of the forthcoming TDL redesign.

abbreviations into their full forms, as for example "h" into "Uhr", "Jan." into "Januar", etc. For tokens algorithmically encoding their denotations the scanner also performs a morphological analysis by assigning a feature structure to them. Such tokens are, above all, cardinal and ordinal numbers, e.g., "12" and "12.", but also complex time and date expressions, such as "14:31:15" or "12.03.1993".

Morphology The morphological component receives as input those tokens which have not been analysed by the scanner. It produces as output a feature structure which contains as a key or index the lemma of the respective item in its STEM attribute, as well as other relevant morphosyntactic information which uniquely identifies the form. For example, the output for the forms "alter" and "kam" is the following feature structures.

(1)
$$\begin{bmatrix} \texttt{STEM} \langle alt \rangle \\ \texttt{HEAD} \begin{bmatrix} \texttt{INFL} & adj\text{-}er \\ \texttt{COMPAR} & pos \\ \texttt{CAT} & adj \end{bmatrix} \end{bmatrix}$$

(2)
$$\begin{bmatrix} \texttt{STEM} \langle komm \rangle \\ \texttt{HEAD} \begin{bmatrix} \texttt{CAT} \ verb \\ \texttt{FIN} \begin{bmatrix} \texttt{AGR} \begin{bmatrix} \texttt{NUM} & sg \\ \texttt{PERS} & \{1,3\} \end{bmatrix} \\ \texttt{TENSE} \ past \\ \texttt{MOOD} \ ind \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 1: Feature structure output of morphology

At present, the morphological information is precompiled into a morphological full form lexicon, so that runtime analysis reduces to the lookup of full forms and the initialization of the lexical component with the associated feature structures. The precompilation is performed by the X2MORF system developed by H. Trost and R. Flassig [Tro91]. Part of this system was redesigned by H. Pirker, with the results that the feature part is now also specified in TDL, and that the morphology can be integrated into the system for a full runtime analysis.

Lexical Component The task of the lexical component is to perform lexical lookup on the output of the scanner and morphology, and to apply lexical rules to the result.

The lexical entries (lemmata) of the grammar are indexed by the values of the STEM features which occur in the output of the morphology. In the simplest case, the matching indices are identified and the morphological feature structure is unified into the corresponding lexical entry (or entries). However, there is also the more complex case of so-called *multi-word lexemes* which match not a single morphological structure but a finite list of morphological structures. All but one STEM feature and any other feature in such a morphological sequence may be underspecified.

After the morphological information has been unified with the lexcial information, *lexical rules* are applied to all suitable lexical structures in the output. The lexical rules are declaratively specified as unary phrase structure rules, which map a morphological structure together with the underspecified lexical entry onto a fully specified lexical structure,

which is then fed as a terminal node to the parser. We will come back to the precise structure of the lexicon in the next section.

Parser The parser is a bidirectional bottom up chart parser which operates on a context-free backbone implicitly contained in the grammar. The parser provides parametrized general parsing strategies, as well as giving control over the processing of individual rules. For example, it is possible to set the control strategy to a breadth first strategy, to give priority to certain rules, or to determine in which order types of daughters, e.g., head daughters, adjunct daughters etc., as well as individual daughters in a specific rule are processed. In addition, the parser provides the facility to filter out useless tasks, i.e., tasks where a rule application can be predicted to fail eventually due to an inevitable unification failure. Some of the filter information is hand-coded at present, but great care was taken to ensure that all necessary information for the parser could also be automatically gathered from the grammar.

4 Structures of the Grammar

The grammar basically distinguishes five different types of structures, all of which are of course described as typed feature structures in TDL:

- Lexical Entries
- Multi-Word Lexemes
- Lexical Rules
- Phrase Structure Schemata
- Root Node

These types of structures represent the yield of the sublattice which is rooted in the HPSG type *sign*. Thus, certain type specifications, such as categorial specifications, principles or immediate dominance schemata can be shared across different types of structures through inheritance links.

Lexical Entries are feature structures which qualify as terminal nodes either of a phrasal or of a lexical rule. The majority of the lexical entries are lemmatized entries, such that for each morphological stem there exists only one lexical entry (modulo homonymity). As already mentioned lexical entries are unified with morphological information in the lexical lookup step. Lexical entries come in two different types, as full-fledged lexical entries and as *arche-lexemes*.

For those categories where there is hardly any variation over a stem, except for the information carried by inflectional morphology, the lexical entry is fully specified and marked as a possible terminal to a phrasal rule. These categories comprise nouns, prepositions,

adverbs, and other kinds of particles.[2] Example (3) shows the entry for a noun in which some of the necessary head features, such as **CASE**, **GENDER** and **NUMBER**, are introduced through a corresponding morphological structure.

(3)
$$\left[\begin{array}{l}\textbf{CAT}\left[\begin{array}{ll}\textbf{SYN} & \left[\begin{array}{ll}\textbf{LOCAL} & \left[\begin{array}{ll}\textbf{HEAD} & \left[\textbf{MAJ}\left[\begin{array}{ll}\textbf{AGR} & \boxed{1}\\ \textbf{N} & +\\ \textbf{V} & -\end{array}\right]\right]\\ \textbf{SUBCAT} & \langle\,\rangle\\ \textbf{LEXICAL} & +\end{array}\right]\end{array}\right]\\ \textbf{SEM} & \ldots\\ \textbf{MORPH} & \left\langle\left[\begin{array}{ll}\textbf{STEM} & \langle \textit{kunde}\rangle\\ \textbf{HEAD} & \boxed{1}\end{array}\right]\right\rangle\end{array}\right]\end{array}\right]$$

Figure 2: Fully specified lexical entry of noun

Next to these, there are also categories, such as verbs and adjectives, for which some syntactic properties vary as a function of their morphological inflection. For example, a verb is assigned different syntactic properties depending on whether it occurs as a finite tensed form, as a non-finite form or as an imperative. Similarly, adjectives have different inflectional endings depending on whether they have attributive, predicative or adverbial function. For these categories, the lexical entries are defined as arche-lexemes to which lexical rules must be applied. The entries for these arche-lexemes are either radically underspecified or may contain information which is "transformed" by a lexical rule. For example, the entries for adjectives (4) are underspecified with respect to whether the form is eventually used as a modifier or as a (predicative) complement. The entries for verbs, on the other hand, may have an "overextensive" subcategorization list, which may be reduced, for example, by a lexical rule deriving imperatives or non-finite forms with a non-overt subject.

Empty terminal nodes are defined as specific types of lexical entries which do not correspond to morphologically realized material. They are represented as feature structures just like any other terminal, i.e., there may be more than one empty terminal, they may carry category specific information etc. The use of empty terminals in the current grammar is limited to the derivation of some specific constructions, such as V-initial or empty noun anaphora; they are not employed in the derivation of non-local dependencies.

Multi-Word Lexemes differ from other lexical entries only in their **MORPH** feature, which is specified as a list with more than one morphological structure. Multi-word lexemes serve primarily for the specification of expressions for which one does not want to assume a com-

[2] We ignore or considerably simplify the representation of semantic information here and in the following.

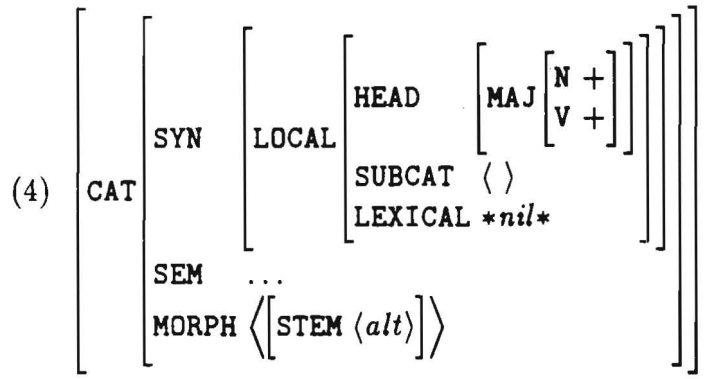


Figure 3: Entry for adjectival arche-lexeme

positional semantics, or which follow very idiosyncratic syntactic construction rules. Multi-word lexemes are mapped onto a sequence of morphological structures with a fixed length. Except for one STEM feature any other element in this pattern may be underspecified.

Phenomena which can be profitably described by multi-word lexemes are idiomatic and phraseological expressions, such as salutations, like *Sehr geehrte Herren*, or the sublanguage of dates and time expressions, as in *Dienstag, der 13. Januar 1993*. In example (5), time expressions such as *14 Uhr 45* are described. The values of the two cardinal numbers are underspecified and coindexed with attributes in the semantic structure representing a generalized time predicate.

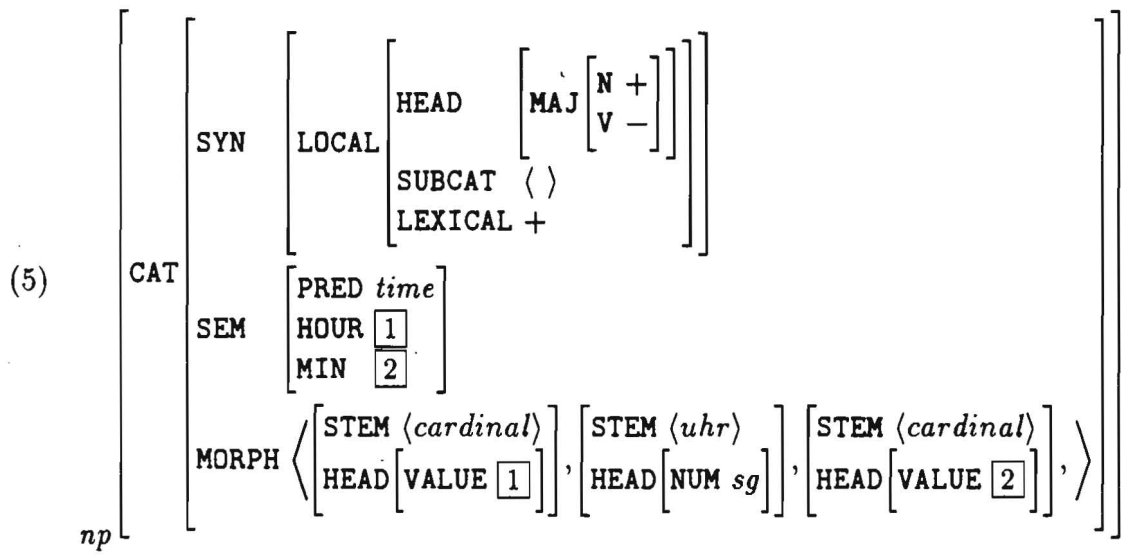


Figure 4: Multi-word lexeme

From the phrase structure perspective, multi-word lexemes look exactly like any other terminal node. Since any terminal may be defined to correspond to the category of a phrasal node, say NP or even S, the "lexical" grammar for these sublanguages blends in neatly with the the more general rule schemata defining regular constructions.

The major advantage of this device is accordingly that it allows one to constrain the description of very specific and exceptional sublanguages to the local domain of the lexicon. The introduction of additional, idiosyncratic rule schemata, which could always have global side effects, can be avoided. Thus, it is almost fully at the discretion of the grammar writer to describe a (non-recursive) pattern by means of phrase structure or else to employ a multi-word lexeme; and to choose where to draw the boundary between the two types of description.

Lexical Rules in our system basically have the form of unary phrase structure rules. These rules can be seen as mapping rules, which take as arguments the specifications of a morphological full form unified with the lexical entry of a corresponding arche-lexeme and map them to fully specified lexical structures, qualifying as the terminal nodes of phrase structure schemata.

As mentioned above, the specification of an arche-lexeme represents the generalization over all the fully specified forms of a given lemma, where this is possible. Thus, in contrast to lexical rules as they have been suggested in, say, LFG, where lexical rules operate on fully specified lexical entries, the input to our lexical rules can be underspecified. By means of lexical rules one can also model non-monotonic processes in the derivation of a lexical item, since the specifications in the daughter of the rule do not have to be identical with the information specified in the mother category. For example, through a lexical rule the subcategorization requirements of an arche-lexeme can be modified as they are derived from an arche-lexeme. Since lexical rules are defined declaratively and do not differ from any other unary phrase structure schema, they can be applied at run-time without any restrictions on order in which they are processed.[3]

In (6), we give as an illustration the lexical rule deriving an attributive adjective from an adjectival arche-lexeme (e.g., (4) above), together with an appropriately inflected morphological form (e.g., (2) above). The effects of this rule are basically monotonic, since it does not change information in the HEAD or SUBCAT features of the arche-lexeme. However, it introduces a large amount of information that would otherwise have to be encoded redundantly in the individual lexical entries. In the distributed disjunction (marked $1) the relevant agreement features are added; they are selected as a function of the inflectional ending of the adjective (as specified by the INFL feature). The rule also introduces the relevant MOD feature through which a nominal projection is selected in adjunction, and the specification that this form has to be used non-predicatively [PRD −].

Rule Schemata In standard HPSG, rule schemata are defined as immediate dominance schemata constraining the phrase structure in combination with linear precedence rules. Phrase structures additionally have to satisfy certain principles, such as the Head Feature

[3]In these latter aspects our approach presumably differs from a suggestion by Andreas Kathol to define arche-lexemes as (generalized) supertypes to which the individual full-forms are defined as subtypes. As far as we can see, Kathol's approach would require pre-compilation or classification for analysis, as well as non-monotonic inheritance for the representation of changes in a lexical specification.

(6)

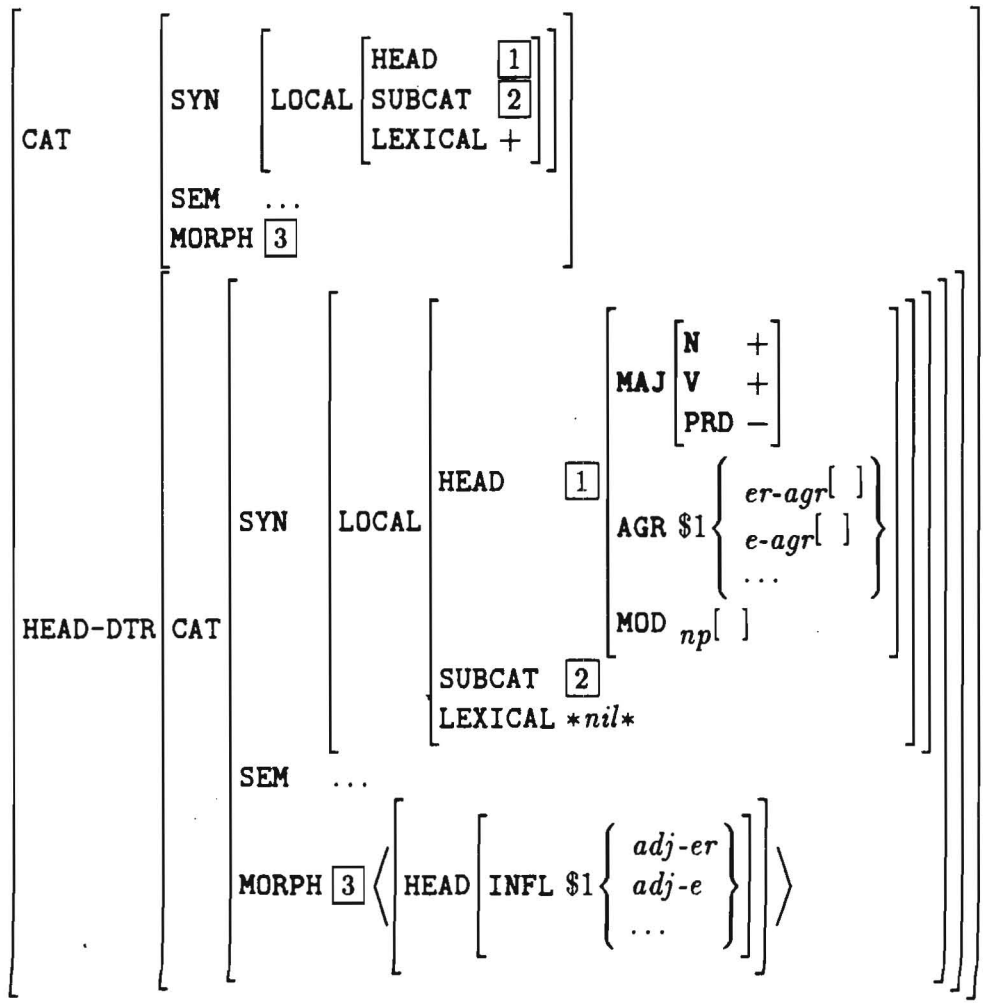


Figure 5: Lexical Rule for Attributive Adjectives

Principle (HFP) or the Semantics Principle, which are encoded as implicational constraints over typed feature structures.

Our grammar differs from this setup insofar as the application of both LP-constraints and principles are encoded as inheritance relations in the type lattice. To be precise, the antecedent of an implication, say the type *headed-structure* in the HFP, is defined as a subtype to a type which specifies the consequent of the principle. The individual phrase structure rules are then defined as subtypes of a more general rule schema, of an appropriate LP-constraint, as well as of all applicable principles. For example, a rule specifying the combination of a non-verbal head (preposition, determiner or noun) with a complement would be defined as a subtype of a general head–complement schema, of an LP-constraint specifying that non-verbal heads precede their complements, and of the types representing the Head Feature Principle and that part of the Semantics Principle which applies to head complement structures.

The rules in the current grammar are all defined as binary branching structures, which,

on the one hand, complicates the specification of more global word order constraints somewhat, but also considerably simplifies the description of other phenomena, such as the interleaving of adjuncts and complements.[4]

Root Node All structures recognized by the parser have to unify with a specific feature structure at their root (equivalent to a start symbol). One function of this root node is to impose certain general, category unspecific well-formedness constraints on structures, such as, for example, full saturation of the subcategorization list or functional completeness.

The way we employ this device does not constrain permissable phrases to the category sentence, but permits maximal projections of all major categories to pass as well-formed expressions on their own. It also provides the interface to the speech act recognition module by collecting relevant syntactic, morphological and semantic information under a specific feature. The value of this attribute represents feature configurations which identify possible sentence types, such as Yes/No-Question, Imperative, Declarative sentence, etc.

5 Coverage of the Grammar

The coverage of the grammar comprises a fair number of the standard constructions of German, as well as more detailed coverage in some specific areas.

On the level of nominal phrases, the grammar contains a quite comprehensive description of the different combinations of specifiers—determiners and numerals—and their morphosyntactic interaction with adjectives. The set of possible nominal modifiers comprises prenominal simple and complex adjectival phrases, postnominal prepositional phrases, adverbs and possessives. Determinerless constructions, such as bare plurals and mass nouns and the complementary phenomenon of nominal ellipsis or empty nominal heads are covered in an integrated analysis [Net93].

Adjectival categories may occur as simple and complex adjective phrases in attributive, predicative and adverbial functions. Prepositional Phrases may be headed by simple or agglutinated prepositions (like *am*) and may have the functions of adverbials, complements and predicatives. The same holds for genuine adverbs.

The different types of verbs covered include main verbs, modal verbs, copula verbs and separable prefix verbs, all of which may occur in all finite forms as well as the bare infinitival form. The extension to other non-finite forms is to a large degree already foreseen and trivial to achieve. The subcategorization frames of verbs may contain all kinds of nominal, prepositional and adjectival complement.

On the clause level all possible positions of the finite verb can be analyzed. Non-local dependencies are limited to topicalization (or left-dislocation) of complements of adjuncts in the "Vorfeld" including also non-verbal pied piping constructions. (There is no treatment for extraposition or right-dislocation, yet.) The analysis of these constructions does not involve empty nodes but operates on a mechanism similar to lexical rules. Currently, the order of complements in the "Mittelfeld" is fixed to a basic order; however, complements

[4]See [Net92] for a discussion of clausal structures.

may be freely mixed with adjuncts. Clause union phenomena are partially covered, so that, for example, negation particles may take wide or narrow scope over modal verbs. All major sentence types (Y/N- and Wh-interrogatives, imperatives and declaratives) are covered and appropriately classified.

Among the various areas in which the grammar still falls short of our goals are two in which we hope to be able to make significant extensions in the near future. These are a more comprehensive treatment of non-finite constructions (including passivization and attributive participle formation), and the entire field of complex sentences.

References

[HS93] **Hinkelman, E. / Spackman, S. P.**: Abductive Speech Act Recognition and the COSMA System. In: Black, W. / Gallagher, J. / Sabah, G. / Wachtel, T. (eds.), *Proceedings of the Second ESPRIT PLUS Workshop in Computational Pragmatics*, New York, 1993. Academic Press

[Kas93] **Kasper, W.**: Integration of Syntax and Semantics in Feature Structures. This volume

[Ner92] **Nerbonne, J.**: Constraint-Based Semantics. In: Dekker, P. / Stokhof, M. (eds.), *Proceedings of the 8th Amsterdam Colloquium*, pp. 425–444. Institute for Logic, Language and Computation, 1992. also DFKI RR-92-18

[Net92] **Netter, K.**: On Non-Head Non-Movement. An HPSG Treatment of Finite Verb Position in German. In: Görz, G. (ed.), *Proceedings of KONVENS 92*. Springer, Berlin/Heidelberg/New York, 1992

[Net93] **Netter, K.**: Towards a Theory of Functional Heads: German Nominal Phrases. In: Nerbonne, J. / Netter, K. / Pollard, C. (eds.), *German Grammar in HPSG*. Chicago University Press, Chicago, 1993

[NOD+93] **Nerbonne, J. / Oepen, S. / Diagne, A. K. / Konrad, K. / Neis, I.**: $\mathcal{NLL}$—Tools for Meaning Representation. This volume

[NOS93] **Neumann, G. / Oepen, S. / Spackman, S. P.**: Design and Implementation of the COSMA System. Technical report, Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken, Germany, 1993

[PS87] **Pollard, C. / Sag, I.**: *Information-Based Syntax and Semantics. Vol. I: Fundamentals.* CSLI Lecture Notes, Number 13. Center for the Study of Language and Information, Stanford, 1987

[PS93] **Pollard, C. / Sag, I.**: *Information-Based Syntax and Semantics. Vol. II: Agreement, Binding and Control.* CSLI Lecture Notes. Center for the Study of Language and Information, Stanford, 1993

[SK92] **Schäfer, U. / Krieger, H.-U.**: TDL *extra-light User's Guide: Franz Allegro Common LISP Version*. DISCO, 1992

[Tro91] **Trost, H.**: X2MORF: A Morphological Component Based on Augmented Two-Level Morphology. Technical Report RR-91-04, Deutsches Forschungsinstitut für Künstliche Intelligenz, Saarbrücken, Germany, 1991

A Diagnostic Tool for German Syntax*

Judith Klein, John Nerbonne, Klaus Netter,
Adbel Kader Diagne† and Ludwig Dickmann‡

†Deutsches Forschungszentrum für Künstliche Intelligenz, GmbH
Stuhlsatzenhausweg 3, D-6600 Saarbrücken 11, FRG
phone: (+49 681) 302-5309
e-mail: klein@dfki.uni-sb.de
‡Institut für Computerlinguistik, Universität des Saarlandes
Im Stadtwald, D-6600 Saarbrücken 11, FRG

Abstract

In this paper we describe an ongoing effort to construct a catalogue of syntactic data exemplifyin the major syntactic patterns of German. The data consist of artificially and systematicall constructed expressions, including also ungrammatical sentences. The data are organized into relational database and annotated with some basic information about the phenomena illustrate and the internal structure of the sample sentences. This paper also contains a description of th abstract data model, the design of the database and the query language used to access the data We invite other research groups to participate in our effort, so that the diagnostics tool ca eventually become public domain. Several groups have already accepted this invitation, an progress is being made

1 Introduction

This paper describes an ongoing effort to construct a catalogue of syntactic data which is intended eventually to exemplify the major syntactic patterns of the German language. Our purpose in developing the catalogue and related facilities is to obtain an empirical basis for diagnosing errors in natural language processing systems analyzing German syntax, but the catalogue may also be of interest to theoretical syntacticians and to researchers in speech and related areas. The data collection differs from most related enterprises in two respects: (i) the material consists of systematically and artificially constructed sentences rather than naturally occurring text, and (ii) the material is annotated with information about the syntactic phenomena illustrated which goes beyond tagging parts of speech. The catalogue currently treats verbal government, sentential coordination, fixed verbal structures (*Funktionsverbgefüge* or FVG i.e., semi-idiomatic constructions with a semantically almost void verbal head and some more or less fixed nominal, prepositional or adjectival complement) and relative clauses. Its total coverage is about 1300 German sentences.

The data consists of linguistic expressions (mostly short sentences designed to

*This work was supported by a research grant, ITW 9002 0, from the German Bundesministerium für Forschung und Technologie to the DFKI project DISCO and by IBM Germany through the project LILOG-SB conducted at the University of Saarbrücken.

exemplify one syntactic phenomenon) together with annotations describing selected syntactic properties of the expression. The annotations of the linguistic material serve (i) to identify and label construction types in order to allow selected systematic testing of specific areas of syntax and (ii) to provide al linguistic knowledge base supporting the research and development of natural language processing (NLP) systems.
In order to probe the accuracy of NLP systems, especially the detection of unwanted overgeneration, the test material includes not only genuine sentences, but also some syntactically ill-formed strings.
The syntactic material, together with its annotations is being organized into a relational database in order to ease access, maintain consistency, faciliate the extension of the syntactic material and allow variable logical views of the data.

2 Goals of a Diagnostic Tool

Our goal in collecting and annotating syntactic material is to develop a diagnostic tool for natural language processing systems, but we believe the material may be of interest to other researchers in natural language, particularly syntactic theoreticians. Finally, although this is not an evaluation tool by itself, our work points to possiblities for evaluating systems of syntactic analysis by allowing the systematic verification of claims about, and investigation of, the coverage and precision of systems. If we are to realize the full benefits of syntactic analysis, then we must ensure that correct analyses are provided. The development of a diagnostic tool serves just this purpose—pointing out where analyses are correct, and where incorrect. The diagnostic tool assesses correctness of syntactic analysies—it supports the recognition of bugs in the linguistic analysis.

3 The Diagnostic Facility

We include here a brief description of **DiTo** - our diagnostic facility; more detailed documentation, especially for the various areas of coverage of the syntactic catalogue, is likewise available. (Cf. Diagne [2], Klein and Dickmann [5])

3.1 Sentence Suite

As noted in the introduction, our material consists of sentences we have carefully constructed to illustrate syntactic phenomena; we have not attempted to collect examples from naturally occurring text. Several considerations weighed in favor of using the the artificially constructed data:

- since the aims are error detection, support of system development, and evaluation of systematic coverage, we need optimal control over the test data. Clearly, it is easier to construct data than to collect it naturally when we have to examine (i) a systematic range of phenomena or (ii) very specific combinations of phenomena.

- we wished to include negative (ill-formedness) data in order to test more precisely (cf. discussion in Section 2.1 on "spurious ambiguity" and also on the needs of generation). Negative data is not available naturally.

- we wished to keep the diagnostic facility small in vocabulary. This is desirable if we are to diagnose errors in a range of systems. The vocabulary used in the diagnostic tool must either (i) be found in the system already, or (ii) be added to it easily. But then the vocabulary must be limited.
- we wished to exploit existing collections of data in descriptive and theoretical linguistics. These are virtually all constructed examples, not naturally occurring text.
- data construction in linguistics is analogous to the control in experimental fields—it allows the testing of maximally precise hypotheses.

The vocabulary for the test suite has been taken from the domain of personnel management wherever possible. We chose this domain because it is popular in natural language processing, both as a textbook example and as an industrial test case. The domain of personnel management would also be useful in case we are to diagnose errors in semantics as well as syntax (which we are not attempting to do at present, but which is an interesting prospect for the future). It presents a reasonably constrained and accessible semantic domain. Where no suitable vocabulary from the domain of personnel management presented itself, we have *extended the vocabulary in ad hoc ways.*

3.2 Syntactic Annotations

In choosing which annotations about the sentences might be sensible, we have been guided by two considerations. First, the catalogue will be much more useful if examples from *selected* areas can be provided on demand. For example, it is useful to be able to ask for examples of coordination involving deletion of the subject in the first conjunct—as opposed to simply coordination (an area of coverage). This means that we need to provide annotations about which area of coverage a given sentence (or ill-formed string) is intended to illustrate. With regard to these annotations, we have merely attempted to use standard (traditional) linguistic terminology.

Besides this information, the annotations contain information about the precise structure of the sentence such as the position of the finite verb (e.g., as 'fifth word') and the positions of other phrases. In selecting these properties as worthy of annotation, we were motivated primarily by a wish to focus on properties about which there would be little theoretical dispute, which would be relatively easy to test, and which would still provide a reasonable reflection of a system's accuracy.

3.3 Example: Verbal Government

One of the phenomena which the data collection already covers is the area of verbal government, i.e., verbal subcategorization frames. The aim was to compile a comprehensive list of *combinations of obligatory complements* of verbs, forming the basis of different sentence patterns in German. We ignore both adjuncts and optional complements in restricting ourselves to obligatory complements.
We attempted to find instances of all possible combinations of nominal, prepositional, sentential, but also adjectival complements. The result of the collection is a list of about 70 combinations which are exemplified in about 220 sample sentences (440 sentences including the negative examples). Every combination of

complements is illustrated by at least one example. The sentences illustrate for example:

- nominal complements only:

  (1) *Der Manager gibt dem Studenten den Computer.*
  the manager gives the student the computer

- nominal and prepositional complements with semantically empty (2) or non-empty prepositions (3):

  (2) *Der Vorschlag bringt den Studenten auf den Lösungsweg.*
  the suggestion takes the student to the solution

  (3) *Der Manager vermutet den Studenten in dem Saal.*
  the manager assumes the student in the hall

In addition, each government type is paired with a set of ill-formed sentences, which illustrate three types of errors relevant for verbal government:

- an obligatory complement is missing;
- there is one complement too many;
- one of the complements has the wrong form.

3.4 Database

The syntactic material, together with its annotations is being organized into a relational database. Our goal in developing the database was to (i) provide a concise organization of syntactic data, (ii) ease access to syntactic information, (iii) maintain consistency, (iv) allow variable logical views of data, and (v) faciliate an efficient extension of the syntactic material to treat further areas.

3.4.1 Abstract Data Model

In this section we will present the conceptual schema of the database, without giving details on the implementation. Figure 1 represents the Entity Relationship (ER) schema diagram of the database. Its current content (syntactic material) treats—enumerating in the order of their development—verbal government, coordination, fixed verbal structures (Funktionsverbgefüge or FVG). i.e., semi-idiomatic constructions with a semantically almost void verbal head and some more or less fixed nominal, prepositional or adjectival complement and relative clauses.

The major relation in the database is the entity type SENTENCE. An entry of SENTENCE contains (i) an identifier **s-id** for tuple which is unique within the relation (primary key), (ii) a sentence that exemplifies the given properties **s-example** (according to the underlying area of application), (iii) the sentence length **s-length**, (iv) a specification of the wellformedness of the sentence example **wf**, (v) an error-code for ill-formed sentences **error_code**, and (vi) additional comment **s-comment**.

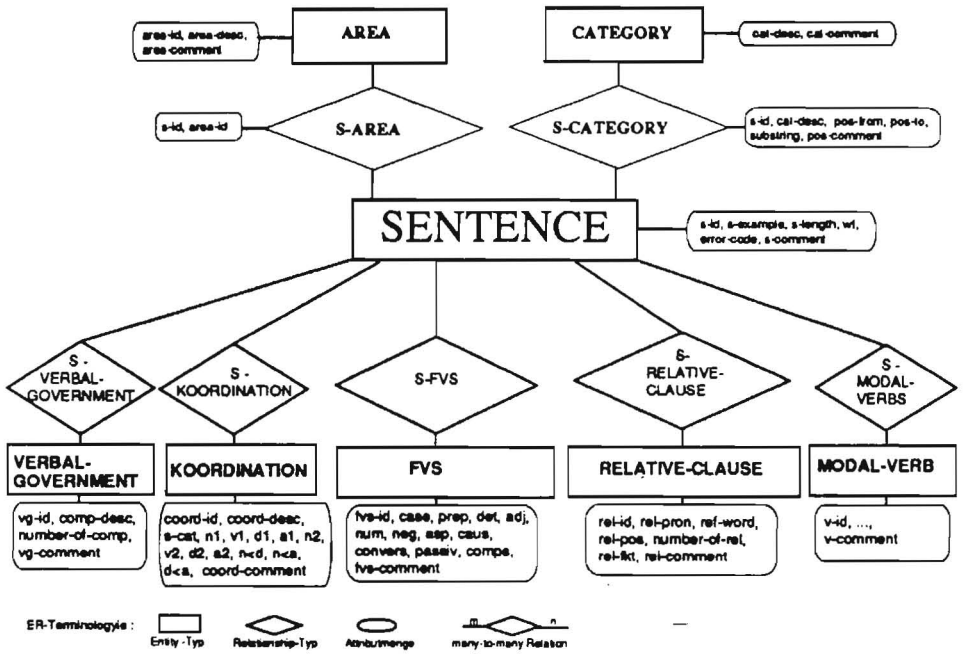


Figure 1: The ER-Schema diagram of the database

The following example shows a database entry for sentence (4).

(4) *Der Manager hindert den Studenten daran, den Plan zu erklären.*

SENTENCE

s-id	s-example	s-length	wf	error-code	s-comment
1210	(4)	10	1	0	null

The CATEGORY relation may contain any category specification, e.g., 'np'. Its participation in the S-CATEGORY relation is partial. A tuple of S-CATEGORY specifies the position and lexical form of a given category in a given sentence. A category may occur in several sentences and a sentence may contain several categories (m:n relationship).

S-CATEGORY

s-id	cat-desc	pos-from	pos-to	substring	pos-comment
1210	np	1	2	der manager	null
1210	f-verb	3	3	hindert	null
1210	np	4	5	den studenten	null
1210	cor	6	6	daran	null
1210	np	7	8	den plan	null
1210	inf-comp	7	10	den plan zu erklären	null

A new application area is recorded in the database by defining a new entity type whose attributes and values depend on the underlying syntactic phenomenon. In accordance with this specification new sentences that exemplify the underlying syntactic phenomenon will then be inserted into the SENTENCE relation. Finally, a new relationship type with the participants SENTENCE and the new entity type must be defined to relate each sentence to the syntactical phenomenon it describes (cf. VERBAL-GOVERNMENT, S-VERBAL-GOVERNMENT).

3.4.2 Database System

The database is administered in the programming language **awk**. Some of the reasons which speak in favor of **awk** are:

- **awk** is in the public domain running under UNIX and should run in other environments; in particular, it runs on MS-DOS.
- Its ability to handle strings of characters as conveniently as most languages handle numbers makes it for our purposes more suitable than standard relational database systems; i.e., more powerful data-validation, increasing availability of information with a minimal number of relations and attributes.

Compared to standard databases **awk** has a restricted area of application and does not provide fast access methods to information, but it is a good language for developing a simple relational database where character strings are involved. Additional resources and tools like a report generator and query languages were easily implemented. The database includes a reduced **sql**-like query language.

The material is organized in a relational database, such that queries can ask either for sentences matching combinations of descriptive parameters (cf. the first two queries) or for a description or classification of a sentence (cf. last query):

1. retrieve all grammatical sentences with verb-scond featuring left-deletion in the second conjunct
   retrieve s-id s-example where wf = 1 and match(coord-desc, N1_V1_D1_A1) and n2 = 2

   s-id: 3007
   s-example: Der Professor schenkte der Sekretaerin den Blumenstrauss und verkaufte dem Kommilitonen den Roman.
   s-id: 3025
   s-example: Der Professor schenkte der Sekretaerin den Blumenstrauss und dem Kommilitonenen den Roman.
   ...

2. retrieve all sentences with a nominative np, a dative np and an accusative np
   retrieve s-id s-example where comp-desc = "nom_dat_acc"

   s-id: 1022
   s-example: Der Manager gibt dem Studenten den Computer.
   s-id: 1023
   s-example: Der Manager verdankt dem Studenten den Computer.
   s-id: 1235
   s-example: *Der Manager gibt.
   s-id: 1236
   s-example: *Der Manager gibt dem Studenten.
   ...

3. retrieve the position and the lexical form of all NPs of sentence 1022.
   retrieve cat-desc cat-position substring where s-id = 1022 and cat-desc = "np"

   cat-desc: np
   pos-from: 1
   pos-to: 2
   substring: der manager

cat-desc: np
pos-from: 4
post-to: 5
substring: dem studenten

cat-desc: np
pos-from: 6
post-to: 7
substring: den computer

The query language has been developed under SunOS using the GNU utilities **flex** and **yacc**. **flex** is a lexical analyzer generator designed for processing of character input streams. **yacc**, a LALR(1) parser generator, is an ancronym for Yet Another Compiler Compiler. It provides a general tool for describing an input language to a computer programm.

3.4.3 Auxiliary Materials

The database of syntactic material is accompanied by a few auxiliary development tools. First, in order to support further development of the catalogue and database, it is possible to obtain a list of words used (so that we minimize vocabulary size). Besides tools for consistency checking have been developed. Documentation is available on each of the areas of syntactic coverage included. This is to cover (minimally) the delimitation of the area of coverage, the scheme of categorization, and the sources used to compile the catalogue. Furthermore a small amount of auxiliary code may be supplied to support development of interfaces to parsers. This need not do more than dispatch sentences to the parser, and check for the correctness of results. As figure 2 shows DiTo currently supports this only by writing sentences to a specified file.

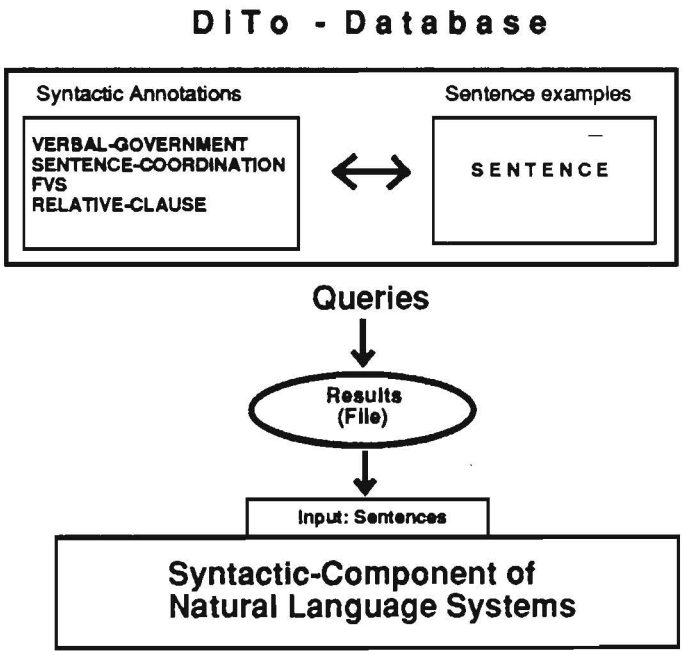


Figure 2: DiTo interface

4 Current State, Future Plans

We have contacted research groups in NLP and machine translation in the interest of exchanging specialized (and annotated) data sets in exchange for the rest of the database. Several groups are now involved in active cooperation: Institut für angewandte Informationswissenschaft (IAI), Saarbrücken, Institut für Computerlinguistik at the University of Koblenz (ICL-Koblenz), and the Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt.
Brigitte Krenn [6] at the IAI has compiled data on the structure of semi-idiomatic verbal constructions (*Funktionsverbgefüge*); Martin Volk [12] at Koblenz has developed a test suite of relative clauses, and Renate Henschel and Elke Teich at the GMD have proposed a data collection and annotation effort on the syntax of modal and auxiliary verbs.
A diagnostic and evaluation tool of this sort ought to be commonly developed, used and maintained. Diagnosis improves in quality and general acceptance as further groups become involved, which in turn enables an increase in the quality and comparability of systems—the more so as the tool itself improves from common use. We invite further interested groups to contact us about collaborations.

References

[1] I. Batori and M. Volk: Das Verhältnis von natürlichsprachlichen Korpora zu systematischen Sammlungen konstruierter Texte. Workshop presentation, *Repräsentatives Korpus der deutschen Gegenwartssprache*, 15-16.10.1992. to appear in a report of the *Institut für Kommunikationsforschung und Phonetik*, Bonn.

[2] A. K. Diagne: DiTo - *DMS. The DiTo Database Management System. Concepts, Implementation Issues and User Guide.* DFKI Technical Document D-92-05, DFKI, Saarbrücken 1992.

[3] D. Flickinger, J. Nerbonne, I. Sag and T. Wasow: Towards evaluation of natural language processing systems. Technical report, Hewlett-Packard Lab., 1987.

[4] G. Guida and G. Mauri: Evaluation of natural language processing systems: Issues and approaches. *Proceedings of the IEEE*, 74(7):1026–1035, 1986.

[5] J. Klein and L. Dickmann: DiTo - *Datenbank. Daten-Dokumentation zu Verbrektion und Koordination.* DFKI Technical Document D-92-04, DFKI, Saarbrücken 1992.

[6] B. Krenn: Funktionsverbgefüge: Eine Datenbeschreibung unpub. Documentation, Institut für angewandte Informationsforschung, Saarbrücken.

[7] J. Klein, L. Dickmann, A. K. Diagne, J. Nerbonne and K. Netter: DiTo: Ein Diagnostikwerkzeug für die syntaktische Analyse. In *Tagungsband KONVENZ 92.* Springer: Berlin, 1992, pp.380-385.

[8] J. Nerbonne, K. Netter, A. K. Diagne, J. Klein and L. Dickmann: A Diagnostic Tool for German Syntax. in press

[9] M. Palmer and T. Finin: Workshop on the Evaluation of Natural Language Processing Systems. In: *Computational Linguistics 16(3)*, 1990, pp.175-181.

[10] W. Read, A. Quilici, J. Reeves, M. Dyer and E. Baker: Evaluating natural language systems: A sourcebook approach. In *COLING '88*, pages 530–534, 1988.

[11] M. Volk and H. Ridder: GTU – eine Grammatik Testumgebung mit Testsatzarchiv to appear in: *LDV-Forum 1.1992*

[12] M. Volk: Kurzbeschreibung der Testsatzsammlung zu den Relativsätzen unpub. Documentation, Universität Koblenz.

Incremental Generation with Tree Adjoining Grammars in the WIP System

Anne Kilger

1 Introduction

A new trend in developing more efficient and flexible natural language generators consists in exploiting an incremental processing scheme. The text generation group of the WIP project has designed and implemented such a system, thereby examining the requirements of incremental generation upon the underlying syntactic representation formalism.

Sequential systems are built of components that – in a strictly serial order – receive their input, start their computation and hand over the complete output to the next component. The run time of the global system results from adding the run times of the single components. Sequential systems cause a long initial delay while completely planning their output. During *incremental processing*, one component starts working on first parts of a stepwise given input and hands over parts of the output to the successor component as fast as possible. This allows the different components to work in parallel reducing the global run time so that messages can be verbalized much more efficiently than by simple sequential systems.

Most of the incremental generators today are only *partially incremental*: They deal with a stepwise given input but delay their output until their processing is finished. *Fully incremental* generators additionally produce their output in an incremental way. This increases efficiency and flexibility since they can start uttering first parts of the sentence before the processing is finished and even before the input is complete ([Finkler & Schauder 92]). Fully incremental systems can be used in situations where the verbalization of some ongoing event must be produced in parallel to the event itself (e.g., during simultaneous interpretation).

TAG-GEN ([Harbusch et al. 91], [Schauder 92]) has been designed as a fully incremental syntactic generator. It realizes part of the How-to-Say task of a natural language generator and is located between the microplanner - which decides about the structure of the sentences and the choice of words - and a simple articulation module. The following sections show some details of TAG-GEN that allow the realization of fully incremental generation.

2 Incremental Syntactic Generation

There are several demands that an incremental syntactic generator must fulfill which have been listed by, e.g., [Kempen & Hoenkamp 82] and [Neumann & Finkler 90]. They restrict the range of possible architectures for such a system and also have influenced the design of TAG–GEN.

The first demand for incremental generators is *lexical guidance*. One subtask of natural language generation consists in choosing lexical items that verbalize the given message – which is prepared by the so–called macroplanner – in an adequate way. If they are chosen before syntactic generation is started the syntactic constraints of the single items can guide the combination of the syntactic structures. This kind of processing is also well suited for incremental syntactic generation because the single items suggest a direct way of distributing the input into several packages of information thereby allowing for incrementally handing over input elements to the generator.

In order to be most flexible, an incremental generation scheme has to provide for input elements arriving in an arbitrary order. If they do so, there is no guarantee for top–down generation which could simply be realized by sequences of downward expansions. Rather, it must be possible to expand the actual structure in different ways. If an element is given that structurally encloses the existing tree, *upward expansion* must take place to attach it at the top of the structure. If an element specifies the actual structure at some point in more detail *downward expansion* has to take place. Modifications of structures may lead to the *insertion* of some parts into the existing tree.

During syntactic generation two main tasks have to be solved. Firstly, the hierarchical structure of the sentence must be constructed. Secondly, it must be linearized in order to create the surface string that is to be uttered. Kempen and Hoenkamp demand *hierarchical and positional knowledge* to be described separately in the grammar. This separation is the presupposition for handling the two tasks in two steps which is advantageous for incremental generation: Rules describing hierarchical relations can be selected independently from decisions about word order. Those decisions can be made lateron on the basis of stepwise given information in the input, e.g., the order of input elements.

The fourth requirement is closely related to the second. Input elements that arrive in an arbitrary order lead to syntactic structures, parts of which can be built independently from one another. For those cases, it seems useful to allow different branches of the global syntactic tree to be expanded *simultaneously*.

The next section illustrates, how the system TAG–GEN fulfills the four demands described above.

3 Tree Adjoining Grammars for Incremental Syntactic Generation

The syntactic generator TAG–GEN is based on Tree Adjoining Grammars with Substitution (cf. [Joshi 83,Schabes et al. 88]) as the underlying syntactic representation formalism.

The extended domain of locality makes TAGs well suited for the representation of natural language structures.

There are two types of syntactic rules in a TAG: *Initial trees* consist of root nodes and internal nodes that must be associated with nonterminals. Their leaves are either associated with terminals or with nonterminals plus a downward arrow marking a *substitution node*. Substitution nodes have to be replaced by other initial trees during the derivation. *Auxiliary trees* define an additional leaf, the so–called *foot node*, that has to be associated with the same nonterminal label as the root node.

TAGs define two combination operations: *Substitution* means to replace a substitution node by an initial tree the root of which has to be labeled with the same nonterminal. During *adjunction* an internal node is replaced by an auxiliary tree. The foot node is used as the new father for the subtree of the node of adjunction.

Initial trees are most of all used to encode complete phrases, auxiliary trees often introduce modifiers. By associating the trees with feature structures syntactic constraints can be encoded in a direct and compact way (TAGs with Unification, cf. [Kilger 92]).

TAGs as the basis for an incremental syntactic generator fulfill the demands listed in Section 2. The different kinds of *expansions* are realized by the two combination operations substitution and adjunction. *Lexical guidance* is made possible by using Lexicalized TAGs (cf. [Schabes et al. 88]). Each tree of an LTAG is associated with at least one terminal which defines the head of the represented phrase and serves as an anchor in the lexicon. Our realization also supports the *simultaneous construction* of parts of the syntactic tree. Lexicalized trees form units that are well suited to be managed by objects of a distributed parallel model (cf. [Finkler & Neumann 89]) which is the basis of TAG–GEN. The substitution nodes of the single trees serve as natural interfaces between the objects which communicate by message–passing. *Two–level generation* can be realized by using CDL–TAGs (cf. [Kilger 93]). They divide each syntactic rule into a mobile and some linear precedence rules defining possible positions of the nodes.

4 The TAG–GEN System

The Interface Component (see Figure 4) interprets and prepares the input for syntactic processing. Since the input packages are centered around lexical items according to the principle of lexical guidance, they can easily be transferred to syntactic structures using Lexicalized TAGs. The trees that are handed over to the objects of the distributed parallel system are computed in three steps: Firstly, the unambiguous reading that is encoded in the input is used to access to the TAG trees with the respective anchor in the lexicon. Secondly, additional input information (e.g., 'voice' for verbs) is used to filter out one of the alternative trees that result from the first step. Thirdly, some information from the lexicon is added to the chosen tree: There is some syntactic information encoded in the lexicon in order to make the grammar less redundant, e.g., detailed specifications of the complements that are represented in the tree. It is added to the chosen tree by expanding the respective feature structures of the head node and of the substitution nodes representing the complements.

There is an object hierarchy defined on the basis of dependency relations that hold between the single lemmas. During further processing each object communicates with its regent and its dependents from this hierarchy.

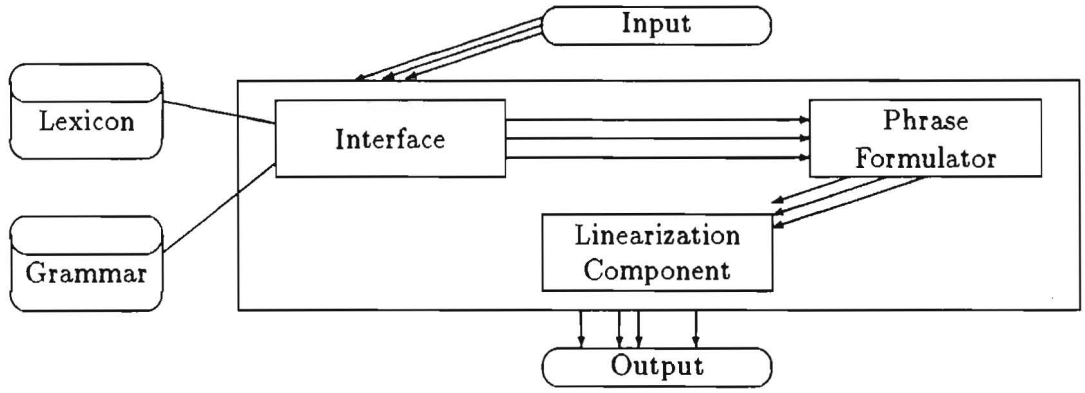


Figure 1: The Architecture of TAG-GEN

As soon as an object has been created in the Interface it changes its state to the *Phrase Formulator*. The task of the objects in the Phrase Formulator is to build the hierarchical structure of a sentence from their local knowledge. The global structure is constructed by means of the two combination operations substitution and adjunction. They are realized by the objects of the parallel distributed model by exchanging copies of local information. During substitution, feature structures of the substitution node and the root node of the substitution tree are exchanged. Adjunction is realized as two coupled substitutions. The node of adjunction is seen as a quasi node (cf. [Vijay-Shanker 92]) the top part of which is connected with the root node of the auxiliary tree, the bottom part is connected with the foot node. The result of the construction process in the Phrase Formulator is a derived tree that is represented in a distributed fashion at the single objects. Substitution nodes, (quasi) nodes of adjunction, root nodes and foot nodes represent the interfaces between the objects of the distributed parallel system. Each object which has integrated its local tree into the global structure changes its state to the *Linearization Component*.

The task of the objects in the Linearization Component is to traverse their trees in order to create the surface string according to the linear precedence rules defined in the grammar. CDL-TAGs (TAGs with context dependent disjunctive linearization rules, cf. [Kilger 93]) are a first approach to solve some problems of linearization during incremental generation. The description of syntactic structures is separated into mobiles representing hierarchical relations and linear precedence rules. Those rules consist of lists of possible permutations of nodes making it possible to describe word order variations in a flexible way. The disadvantage of listing permutations is that it is hard to define positions of optional elements that might be integrated by means of adjunction. Future work will concentrate on the development of a formalism that is able to describe features of word order in natural language and at the same time supports incremental syntactic processing.

The second important feature of CDL-TAGs is that they allow to associate different

sets of disjunctive rules with keys referring to specific 'linearization contexts' where the rules are valid. This compact representation increases the flexibility of the generator since it allows to choose the mobile independent from its linearization context which may be specified later in the input.

Processing in the Linearization Component consists of two kinds of activities. Each object that enters the Linearization Component first asks its regent for the permission to utter its part of the sentence. This question is forwarded from regent to regent until an object is found that is able to decide about it. This kind of message passing leads to a global synchronization of output activities that allows to produce output incrementally. Simultaneously, each object locally traverses its tree and utters the collected strings as soon as it receives the permission for output.

5 Conclusion

TAG-GEN is a syntactic generator on the basis of the representation formalism Tree Adjoining Grammars. It can be used for a thorough study of fully incremental natural language generation. Incremental syntactic processing is realized on a hierarchical and a positional level that are both based on a distributed parallel model of active cooperating objects. In the Phrase Formulator, substitution and adjunction are implemented by means of message passing as virtual combination operations. A first approach to incremental linearization is facilitated by CDL-TAGs, a new extension of the TAG formalism.

Work on both the Linearization Component and a central control component for revisions will be intensified to improve the TAG-GEN generator. Thereby, teamwork with other DFKI projects could be helpful. E.g., new ideas about cooperating agents which are developed by AKA-MOD could influence the distributed parallel model underlying TAG-GEN. The large HPSG grammar for German that is actually created in the DISCO project can eventually be transformed into a lexicalized TAG (cf. [Kasper 92]). This would provide the TAG-GEN group with a broader range of syntactic structures that can be used to test the flexibility of the generator.

References

[Finkler & Neumann 89] W. **Finkler** and G. **Neumann**. *POPEL-HOW – A Distributed Parallel Model for Incremental Natural Language Production with Feedback.* In: 11th International Joint Conference on Artificial Intelligence, pp. 1518–1523, Detroit, MI, August 1989.

[Finkler & Schauder 92] W. **Finkler** and A. **Schauder**. *Effects of Incremental Output on Incremental Natural Language Generation.* In: B. Neumann (ed.), 10th European Conference on Artificial Intelligence, pp. 505–507, Vienna, Austria, August 1992. Wiley. also published in this volume.

[Harbusch et al. 91] K. **Harbusch**, W. **Finkler**, and A. **Schauder**. *Incremental Syntax Generation with Tree Adjoining Grammars.* In: W. Brauer and D. Hernández

(eds.), 4th International GI Congress on Knowledge–Based Systems, pp. 363–374, Munich, Germany, October 1991. Springer.

[Joshi 83] A. K. **Joshi**. *Factoring Recursion and Dependencies: an Aspect of TAG and a Comparison of Some Formal Properties of TAGs, GPSGs, PLGs and LFGs.* In: 21st Annual Meeting of the Association for Computational Linguistics, pp. 7–15, Cambridge, MA, 1983.

[Kasper 92] R. **Kasper**. *Compiling Head-Driven Phrase Structure Grammar into Lexicalized Tree Adjoining Grammars.* In: Proceedings of the 2nd International TAG+ Workshop, Philadelphia, PA, 1992.

[Kempen & Hoenkamp 82] G. **Kempen** and E. **Hoenkamp**. *Incremental Sentence Generation: Implications for the Structure of a Syntactic Processor.* In: J. Horecky (ed.), 9th International Conference on Computational Linguistics. North–Holland Publishing Company, 1982.

[Kilger 92] A. **Kilger**. *Realization of Tree Adjoining Grammars with Unification.* DFKI Document TM–92–08, German Research Center for Artificial Intelligence (DFKI), 1992.

[Kilger 93] A. **Kilger**. *TAGs with Context Dependent Disjunctive Linearization Rules.* Dfki document, German Research Center for Artificial Intelligence (DFKI), 1993. to appear.

[Neumann & Finkler 90] G. **Neumann** and W. **Finkler**. *A Head–Driven Approach to Incremental and Parallel Generation of Syntactic Structures.* In: 13th International Conference on Computational Linguistics, pp. 288–293, Helsinki, Finland, 1990.

[Schabes et al. 88] Y. **Schabes**, A. **Abeillé**, and A. K. **Joshi**. *Parsing Strategies with Lexicalized Grammars: Application to Tree Adjoining Grammar.* In: 12th International Conference on Computational Linguistics, Budapest, Hungary, 1988.

[Schauder 92] A. **Schauder**. *Incremental Syntactic Generation of Natural Language with Tree Adjoining Grammars.* Technical Memo D–92–21, DFKI, Saarbrücken, Germany, 1992.

[Vijay-Shanker 92] K. **Vijay-Shanker**. *Using Descriptions of Trees in a Tree Adjoining Grammar.* Computational Linguistics, 1992. to appear.

Integration of Syntax and Semantics in Feature Structures

Walter Kasper

1 Introduction

Unification based grammar formalism which have become the standard during the last decade in computational linguistics employ to a varying degree feature structures as means of expressing constraints. Older grammar formalisms in this tradition usually consist of a backbone of context-free phrase structure rules annotated with feature structures which are projected along the phrase structure tree.[1] Different formalisms and linguistic theory differ with respect to the kind of phrase structure, they assume, what kind of information should be represented in the feature structures, and what kind of projection principles and rules they assumed. Also, the underlying unification formalisms differed in the kinds of operations, data structures and constraints they provided additionally to the core of the attribute-value-language, such as negation, sets, disjunction, existential constraints, completeness constraints etc.

In contrast to these formalisms, HPSG encodes all information in typed feature structures without recourse to phrase structure rules and instead uses a set of general principles to constrain possible derivations. The basic notion of HPSG is that of a *sign* which includes information of all levels of linguistic analysis: phonology, morphology, syntactic, semantic. This allows at the same time to specify the interdependencies between these levels in a concise and declarative way.

In the following sections we will describe the integration of the semantics interface into the grammars used by the ASL- and DISCO-projects. The grammars are based on different syntactic theories and formalisms: the ASL grammar is a phrase structure grammar with annotated feature structures based on *Trace & Unification Grammar* (TUG; Block/Schachtl(1992)). The grammar used in DISCO on the other hand is an HPSG grammar. Nevertheless, the same semantical representations are built up in these different grammars. This makes it in principle possible to exchange the two grammars in an NLP-system without having to re-design other interfaces. This also demonstrates the practical usefulness of integrating the semantics interface into the grammar.

[1] Formalisms of this kind are PATR-II (Shieber et al.(1983)), Lexical-Functional Grammar (LFG; Bresnan/Kaplan(1982)), Categorial Unification Grammar (CUG; Uszkoreit(1986)).

2 Scope of Integration

For each of the unification-based formalisms there have been investigations to include semantics and to build semantic representations by using the same techniques as used for syntactic structures. There are several advantages to such an approach, additionally to having a uniform formalism for syntactic and semantic information:

- semantic constraints can be exploited already during syntactic analysis, and constrain the analysis.
- it allows to describe the interactions of different levels of analysis in a declarative way.
- it allows nearly arbitrarily partiality of semantic information which usually would be impossible to achieve in standard logical formalisms because of their rigid conditions on wellformedness.
- it relaxes the requirement of having to specify in a rigid way how syntactic analyses of phrases must look like in order to be meaningful to a separate semantic interpreter.[2]

The current unification formalisms do not allow full descriptions of semantic representations. It is especially non-local phenomena like the scope of quantifiers and semantic operators which are not easy or even impossible to describe within these formalisms. Therefore it has become practise — followed also in our system — to specify the semantic representation only partially up to a level of 'quasi-logical form' which leaves such non-local phenomena out of consideration. These are left to a second step of semantic interpretation. Such quasi-logical forms contain the following kind of information:

- predicate-argument-structure
- thematic roles of arguments
- the modification relation
- sortal selectional restrictions on predicate-argument-structures and modification. These have proved to be very efficient means for disambiguation especially of PP-attachment ambiguities.

Additionally, quasi-logical forms can specify other kinds of information relevant for the second step of interpretation.

3 Semantic Feature Structures

A grammar consists of at least a lexicon and syntactic rules for combining lexemes to more complex structures. Accordingly, if semantics is to be integrated int the gram-

[2]Problems of this kind have been discussed in Kasper(to appear) with respectto LFG.

mar, two things are required: first, to specify the lexical semantics of the lexemes, and secondly, to specify the rules of how lexical semantic information is combined to build up the semantics of the complex phrases. The form of semantic feature structures is influenced essentially by two factors: the underlying logical representation language and projection principles.

The basic ingredients of logical representation languages are *terms* designating objects, *predicates (relations)* and *formulas (wffs)*. The feature structure then takes the form of a meta-logical description of the logical representation, using the logical type of an expression as feature name (cf. Nerbonne(1991)). Projection principles govern the way semantic information is combined or passed on, such as the *head-feature-principle* of HPSG which states that the (syntactic) features of the head daughter in a tree structure are inherited by the mother node.

(1) V
$$\left[\begin{array}{l} \text{SYN}: \left[\text{SUBCAT}: \left[\begin{array}{l}\text{ARG1}: \left[\text{SEM}: \boxed{1}\right] \\ \text{ARG3}: \left[\text{SEM}: \boxed{2}\right]\end{array}\right]\right] \\ \text{SEM}: \left[\begin{array}{l}\text{CONTENT}: \left[\begin{array}{l}\text{PRED}: \text{have} \\ \text{INST}: \left[\text{SORT}: \text{state}\right]\boxed{3} \\ \text{AGENT}: \boxed{1} \\ \text{THEME}: \boxed{2}\end{array}\right] \\ \text{RESTR}: \left[\begin{array}{l}\text{VAR}: \boxed{3} \\ \text{COND}: \left[\begin{array}{l}\text{PRED}: \text{present} \\ \text{INST}: \boxed{3}\end{array}\right]\end{array}\right]\end{array}\right]\end{array}\right]$$

The above lexical entry for the transitive verb *has* contains under the SEM-feature the following information:

1. it specifies the mapping from the syntactic complements to thematic roles in the predicate-argument-structure under CONTENT by co-indexation with arguments in the verb's complement list SUBCAT.
2. the verb introduces an entity with a sortal aspectual restriction that the verb describes a state (in contrast to e.g. an event). This constraint can be exploited e.g. to exclude that the can be modified by a directional adverbial like *to the station*.

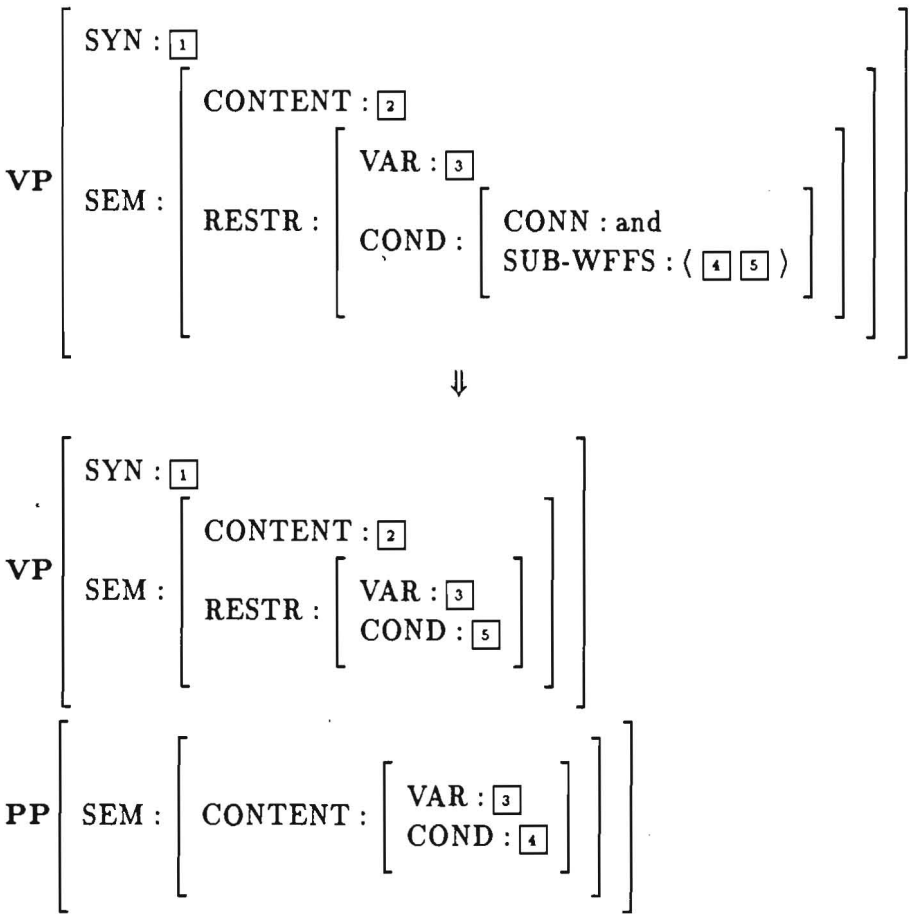
3. the RESTR specifies further restrictions on the stative entity introduced by the verb, in this case that the state is temporally located in the present.

Logically, the entry describes the following $\mathcal{NLL}$ formula: [3]

(2) have(inst:(?s|(present(inst:s))) agent:?x theme:?y)

In the feature structure the restriction imposed on the variable *s* is described in a separate structure RESTR due to considerations about the projection of information in analysis: the basic predicate-argument-structure is contained under CONTENT and passed unchanged to the highest projection of the category while RESTR collects the restrictions and modifiers of the denoted entities, as the following rule illustrates:

(3)



This rule for attaching a prepositional phrase (PP) to a verbal phrase (VP) has been taken from the ASL-grammar. It states that the CONTENT of the daughter VP which is the head phrase is passed to the mother VP, while the mother's RESTR collects the content of the PP and the other restrictions on the head phrase by building their logical

[3] $\mathcal{NLL}$ is an extended predicate logic language used in our system (Laubsch/Nerbonne(1991)).

conjunction.[4] Since PP and head phrase share variables sortal restrictions of the PP and the head must be compatible. In a HPSG-style grammar this could be generalized to a *head-modifier-principle* which applies not only to attaching a PP to a VP but to other categories such as NPs as well.

4 Outlook

As indicated, at present semantics can be integrated only partially into the grammar. Therefore, the semantics interface requires two steps though the mapping from grammar to logical representations has become much simpler by having the basic ingredients already represented in the feature structures. Current research aims at pushing the borderline between these two steps, and to include more and more into the semantic feature structures and leaving as little as possible to the second step.

References

Block, H. U./Schachtl, S. (1992): Trace & Unification Grammar. In: *Papers presented to the 14th International Conference on Computational Linguistics*, pp. 87–93, Nantes, 1992

Bresnan, J./Kaplan, R. (1982): Lexical Functional Grammar: A Formal System for Grammatical Representation. In: Bresnan, J. (ed.), *The Mental Representation of Grammatical Relations*. Cambridge/Mass., 1982

Kasper, W. (to appear): The Construction of Semantic Representations from F-STRUCTURES. In: Bes, G. (ed.), *The Construction of a Natural Language and Graphics Interface. Results and Perspectives from the ACORD Project*, Research Reports ESPRIT. Heidelberg: Springer, to appear

Laubsch, J./Nerbonne, J. (1991): An Overview of $\mathcal{NLL}$. Technical report, Hewlett-Packard Laboratories, Palo Alto, July 1991

Nerbonne, J. (1991): Constraint-Based Semantics. In: Dekker, P./van der Does, J. (eds.), *Proceedings of the 8th Amsterdam Colloquium*, 1991

Shieber, S./Uszkoreit, H./Pereira, F./Robinson, J./Tyson, M. (1983): The Formalism and Implementation of PATR-II. In: Grosz, B./Stickel, M. (eds.), *Research on Interactive Acquisition and Use of Knowledge*, pp. 39–79. Menlo Park: SRI International, 1983

Uszkoreit, H. (1986): Categorial Unification Grammars. In: *Proceedings of the 11th Conference on Computational Linguistics*, pp. 187–194, Bonn, 1986

[4] Strictly, this holds only for intersective modifiers.

Effects of Incremental Output on Incremental Natural Language Generation

Wolfgang Finkler and Anne Kilger (née Schauder)

Abstract

The benefits of incremental processing for natural language generation can only be fully utilized if the systems produce incremental output. Nevertheless, most of the known generators delay their output until it is complete. We show some advantages of incremental output and present its effects on the design and the processing of NL generation systems.
Keywords: incremental NL generation, speech repair.

1 Motivation

In order to use natural language in human–computer interfaces in an effective way, generation and analysis components have to be flexible and efficient. A promising approach consists in the use of an incremental processing mode as indicated by psycholinguistical studies. Incremental systems receive their input in a piecemeal way and immediately start with the computation of these parts. Therefore, the total time of computation can be shorter than for sequential systems, approaching real-time behavior (see Fig. 1). If NL systems additionally produce incremental output, they become more efficient. Producing incremental output is defined as uttering the first parts of a sentence before its computation or even the input is complete (see the uppermost scheme in Fig. 1). These systems are more flexible because they can be used in situations where it is not possible to delay the output until the input is complete, e.g., during simultaneous interpretation. Generally, incremental output is more natural for human recipients because it supports fluency in speech by shortening the initial delay of an utterance and distributing smaller pauses over the whole sentence.

Although there is an increasing number of systems for incremental NL generation, no approach is known in which the output of language production is treated adequately. Most of the systems are only *partially incremental.* Either their output is delayed until the surface structure is complete, or the effects of overtly spoken or written parts of an utterance are not examined (e.g., [Reithinger 91], [De Smedt 90]). In this paper[1] we will describe such

[1]Thanks to Wolfgang Wahlster for fruitful discussions.

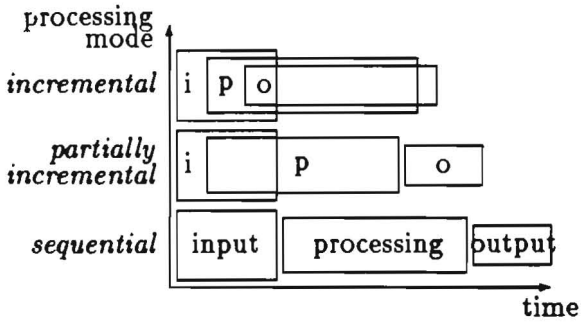


Figure 1: Processing Modes for Generation

effects and elaborate some consequences for the design of a syntactic generation component currently developed in the WIP project (cf. [Wahlster et al. 93]).

2 Types of Incremental Output

To produce fluent output, incremental generation systems must be able to begin an utterance and to continue it with respect to the uttered prefix. There are three ways to continue a sentence in an incremental way:

In the best case, the verbalization of every incoming part can be articulated following the uttered prefix. However, the generation process can be impeded, e.g., because parts of the input are given too late to be correctly integrated into the sentence or the input is modified. If the sentence cannot be continued correctly, 'overt repair' leads to utterances like "The child plays ... uh ... the tall child plays piano". Sometimes it is possible to hide the repair, e.g., if changes of syntactic constructions don't lead to changes in the already uttered portion. These 'hidden repairs' can sometimes be recognized by pauses, intonation or a bad style, as in "The goalkeeper has ... led the ball pass through". When starting the utterance the reporter might have thought the goalkeeper would stop the ball.

An alternative strategy consists in moving the verbalization of some fragments to an additional sentence if it seems to be too complicated or impossible to integrate it. This results in a sequence of sentences, as in "The child plays ... piano. The child is tall.".

3 Design of a System with Incremental Output

In this section we investigate the effects of incremental output on description–directed NL generation (see [McDonald 87]). For syntactic generation, several levels of description can be conceived that correspond to levels of processing, i.e., lexical choice, construction of the constituent structure and linearization. During incremental generation the represented data not only trigger their computation but also constrain further processing:

In order to achieve 'lexical guidance' (cf. [Kempen & Hoenkamp 82], [Neumann & Finkler 90]), each selected lemma directs the choice and processing of syntactic structures. The process of lexical choice not only depends on the given concepts but also on the *previously*

chosen lemmata and their created structures providing a syntactic context.

If grammatical encoding can lead to alternative structures, the decision between them must observe the *existing syntactic structure* as the basis for incremental expansions. In order to integrate new parts in a flexible way, the representation formalism must permit several kinds of expansions (cf. [Kempen 87], [De Smedt & Kempen 87]). E.g., by means of 'upward expansion' a structure can be embedded in a hierarchically higher one.

The *existing prefix* that has already been uttered should be continued in a syntactically correct way[2]. If the chosen syntactic structure does not fulfill this constraint, humans often try to retain the started utterance by means of hidden repair. This capability decreases the number of overt repairs that are unpleasant because they can be perceived by the hearer.

While the first and second constraint also hold for incremental generation systems with delayed output, the third one is new. If a system produces incremental output, it is no longer guaranteed that a correct position can be found for each syntactic structure that can be integrated on the constituent level. E.g., uttering the prefix "The bag was ..." makes it impossible to position "man's" in front of "bag". The third constraint additionally influences the design of the subcomponents of an incremental generator: Lemmata, constituent structure and linear order are often computed and represented at different levels realizing a modular approach with rather independent components. Producing incremental output causes a new interrelation between these levels. This task mainly affects the lowest module of a syntactic generator, the linearization component (LC), which is responsible for ordering parts of an utterance and for handing them over to an articulator in an incremental way. The input for the LC consists of constituent structures and realized words defining the set of possible linearizations. In the course of incremental output, each uttered fragment constrains the further linearization by reducing the set of linear precedence rules that describe possible continuations of the sentence. This additional constraint also influences the other components:

- Assume that a new element that has been structurally integrated at the constituent level is handed over to the LC. If no linearization rule remains that allows this element to be positioned to the right of the uttered prefix, it may be necessary to select another syntactic rule. An example of this dependency is an adjective that can be uttered even if the noun has already been said by integrating it in a relative clause ("The ball ... that is red ...").
- Another example describes the relation between LC and word choice. In German, complements realized as pronouns have to be positioned in front of complements realized as nouns if the finite verb has already been uttered. For a transitive verb like "geben" ("to give") the sentence starting "Ich gebe das Buch" ("I give the book") must not be continued with a pronoun. If a pronoun is nevertheless chosen, it must be rejected and replaced by the respective noun, as in "Ich gebe das Buch ... dem Mann" ("to the man") instead of "ihm" ("him").

In this way, the state of output and the linearization rules influence the selection of syntactic structures and lexical material. It is obvious that a modular approach can only

[2]This demand is comparable with the valid prefix property for parsers (cf. [Schabes 90]).

deal with these bidirectional dependencies indirectly, e.g., by means of generate and test cycles. Therefore, an effective model should attempt to interleave processing and integrate the related levels to a certain extent.

As discussed above, the uttered prefix strongly influences further computation at the different levels of an incremental generator. Therefore, the system's decision about *when to utter* plays a central part during generation. If the output is delayed, more information is available to compute a prefix that will not be revised later. But each unnecessary delay decreases efficiency and flexibility of the system. It is important to find the right balance between delaying the output and producing fluent utterances.

An incremental style of processing relies on the assumption that the decisions made on the basis of actual information need not be withdrawn later on. This assumption often turns out to be wrong. During incremental generation it may happen that the constraints described in the beginning of this section are violated. It is possible that a new part cannot be integrated into the existing syntactic structure or that the verbalization of an element would change the uttered prefix. In order to cope with these problems, they first of all have to be detected by the system (*'monitoring'*). The effectiveness of controlling the correction of such failures (*'supervising'*) depends on the amount of available information. Monitor and supervisor should have direct access to the internal state of the generation components. This allows the localization of the origin of the failure as well as the reuse of partial structures during repair. According to the computationally oriented approach to NL generation we suggest the utilization of this kind of direct influence on the incremental generator. In contrast to this approach, the so–called 'perceptual theory of monitoring' ([Levelt 89]) only describes monitoring as analyzing the inner or overt speech without direct access to the processes during grammatical encoding.

As discussed in Section 2, there are several strategies for repair which are candidates for the supervisor when it tries to direct the continuation of the sentence. To perform a repair, decisions must be withdrawn, i.e., structures have to be replaced. This can affect other structures which are connected with them. There is a need for a fast identification of and access to the affected structures. Storing the dependency data in a reason maintenance system supports such revisions. Another way to identify such structures and to find syntactic alternatives consists in parsing the actual output. This may result in several readings; one of them can be used as the basis for further expansions.

4 State of Realization

We have realized a prototype for an incremental generator (cf. [Schauder 92]) and integrated it into the WIP system ([Wahlster et al. 93], [André et al. 92]). The system is implemented in an object–oriented style in Common LISP and CLOS on MacIvory machines.

Our generator includes a hierarchical and a positional level where constituent structures and their linear order are computed. Word choice will be added in a next step. Both levels are based on the formalism of 'Lexicalized LD/LP Tree Adjoining Grammars'

(cf. [Harbusch et al. 91]). It associates lexical items with syntactic rules, permits flexible expansion operations and allows the separation of the description of local dominance from linear precedence rules.

We have implemented a distributed parallel model with active cooperating objects supporting the flexible and efficient processing of incrementally given structures. Each object is associated with one lemma and one grammar rule representing its syntactic context. On the hierachical level, the objects try to virtually combine their local structures via message passing. Linearization is realized as traversing the local structures and handing over the permission for output from one object to the other. Currently, only one simple revision strategy is realized. It helps to integrate elements that would change the prefix by revising the respective part of the string and repeating the whole utterance.

Conclusion

We have shown that the production of incremental output strongly influences the design of a generation system. The prefix of an utterance constrains the further construction of the sentence. Several types of repair can help to overcome the violation of this constraint. Contrary to the output of current systems, the production of incremental output is more natural for human recipients because it supports fluency in speech.

References

[André et al. 92] E. **André**, W. **Finkler**, W. **Graf**, T. **Rist**, A. **Schauder**, and W. **Wahlster**. *WIP: The Automatic Synthesis of Multimodal Presentations.* In: M. Maybury (ed.), Intelligent Multimedia Interfaces. AAAI Press, 1992. to appear.

[De Smedt & Kempen 87] K. **De Smedt** and G. **Kempen**. *Incremental Sentence Production, Self-Correction and Coordination.* In: Gerard Kempen (ed.), Natural Language Generation, pp. 365–376. Dordrecht: Martinus Nijhoff, 1987.

[De Smedt 90] K. **De Smedt**. *Incremental Sentence Generation, a Computer Model of Grammatical Encoding.* PhD thesis, NICI, Nijmegen, 1990.

[Harbusch et al. 91] K. **Harbusch**, W. **Finkler**, and A. **Schauder**. *Incremental Syntax Generation with Tree Adjoining Grammars.* In: 4th International GI Congress, pp. 363 – 374, Munich, 1991.

[Kempen & Hoenkamp 82] G. **Kempen** and E. **Hoenkamp**. *Incremental Sentence Generation: Implications for the Structure of a Syntactic Processor.* In: COLING'82. North-Holland, 1982.

[Kempen 87] G. **Kempen**. *A Framework for Incremental Syntactic Tree Formation.* In: 10th IJCAI, Milano, 1987.

[Levelt 89] W. J. M. **Levelt**. *Speaking: From Intention to Articulation.* Cambridge, MA: MIT Press, 1989.

[McDonald 87] D. D. **McDonald**. *Natural Language Generation: Complexities and Techniques.* In: S. Nirenburg (ed.), Machine Translation. Cambridge: Cambridge University Press, 1987.

[Neumann & Finkler 90] G. **Neumann** and W. **Finkler**. *A Head-Driven Approach to Incremental and Parallel Generation of Syntactic Structures.* In: 13th COLING, pp. 288–293, Helsinki, 1990.

[Reithinger 91] N. **Reithinger**. *Eine parallele Architektur zur inkrementellen Generierung multimodaler Dialogbeiträge.* PhD thesis, University of the Saarland, 1991.

[Schabes 90] Y. **Schabes**. *Mathematical and Computational Aspects of Lexicalized Grammars.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, Pennsylvania, 1990. MS-CIS-90-48, LINC LAB 179.

[Schauder 92] A. **Schauder**. *Incremental Syntactic Generation of Natural Language with Tree Adjoining Grammars.* Document D-92-21, DFKI, Saarbrücken, 1992.

[Wahlster et al. 93] W. **Wahlster**, E. **André**, W. **Finkler**, H.-J. **Profitlich**, and T. **Rist**. *Plan-based Integration of Natural Language and Graphics Generation.* Artificial Intelligence, to appear, 1993.

An Extended RST Planner for the Generation of Multimodal Presentations

Elisabeth André
German Research Center for Artificial Intelligence (DFKI)
W-6600 Saarbrücken 11, Stuhlsatzenhausweg 3, Germany
E-mail: andre@dfki.uni-sb.de

Abstract:*

The aim of our work is to develop a presentation system that is able to automatically generate illustrated documents. We start from the assumption that not only the generation of text, but also the generation of multimodal documents can be considered as a sequence of communicative acts which aim to achieve certain goals. The coordination of the different modes requires knowledge concerning the functions of textual and pictorial document parts and the relations between them. Based on textlinguistic work, the structure of an illustrated document is described by the hierarchical order of communicative acts and the relations between them. For the automated generation of illustrated documents, we propose an RST-like planner that supports data transfer between the content planner and the mode-specific generation components and allows for revising an initial document structure.

1 Introduction

Recently, there has been increasing interest in the design of systems generating multimodal output. Research in this area addresses the analysis and representation of presentation knowledge (cf. [Arens et al. 93]) as well as computational methods for the automatic synthesis of multimodal presentations (cf. [Badler et al. 91], [Feiner/McKeown 91], [Marks/Reiter 90], [Maybury 93], [Roth et al. 91] and [Wahlster et al. 93]). There seems to be general agreement that in these interfaces the presentation of information should incorporate different modes, particularly text and graphics. Such tailoring requires knowledge concerning the functions of textual and pictorial document parts and the relations between them. Furthermore, a presentation system must be able to handle the various dependencies between content planning, mode selection and content realization.

In the following, we will show that concepts applied in natural language generation, such as communicative acts and coherence relations, can be adapted in such a way that they become useful for the generation of text-picture combinations. We will present an approach that integrates content planning and mode selection and allows for interaction with mode-specific generators.

2 The Structure of Illustrated Documents

Our approach is based on the assumption that not only the generation of text, but also the generation of multimodal documents can be considered as an act sequence that aims to achieve certain goals (cf. [André/Rist 90]). We presume that there is at least one act that is central to the goal of the whole document. This act is referred to as the *main act*. Acts supporting the main act are called *subsidiary acts*.[1] Main and subsidiary acts can, in turn, be composed of main and subsidiary acts. The root of the resulting hierarchical structure generally corresponds to a complex communicative act such as describing a process, and its leaves are elementary acts, i.e., speech acts (cf. [Searle 69]) or pictorial acts (cf. [Kjorup 78]).

---

* The work presented here is supported by the BMFT under grant ITW8901 8.

[1] This distinction between main and subsidiary acts essentially corresponds to the distinction between *global* and *subsidiary speech acts* in [Searle 69], *main speech acts* and *subordinate speech acts* in [Van Dijk 80], *dominierenden Handlungen* and *subsidiären Handlungen* in [Brandt et al. 83] and between *nucleus* and *satellites* in [Mann/Thompson 87].

The structure of a document is, however, not only determined by its hierarchical act structure, but also by the role acts play in relation to other acts. In textlinguistic studies, a variety of coherence relations between text segments has been proposed (e.g., see [Grimes 75] and [Hobbs 78]). Perhaps the most elaborated set is presented in Rhetorical Structure Theory (RST, cf. [Mann/Thompson 87]). Examples of RST-relations are *Motivation*, *Elaboration*, *Enablement*, *Interpretation* and *Summary*. Text-picture researchers have investigated the role a particular picture plays in relation to accompanying text passages. E.g., Levin has found five primary functions (cf. [Levin et al. 87]): *Decoration*, *Representation*, *Organization*, *Interpretation* and *Transformation*. Hunter and colleagues distinguish between: *Embellish*, *Reinforce*, *Elaborate*, *Summarize* and *Compare* (cf. [Hunter et al. 87]). An attempt at a transfer of the relations proposed by Hobbs to pictures and text-picture combinations has been made in [Bandyopadhyay 90]. Unfortunately, text-picture researchers only consider the communicative functions of whole pictures, i.e., they do not address the question of how a picture is organized. To get an informative description of the whole document structure, one has to consider relations between picture parts or between picture parts and text passages too. E.g., a portion of a picture can serve as background for the rest of the picture or a text passage can elaborate on a particular section of a picture.

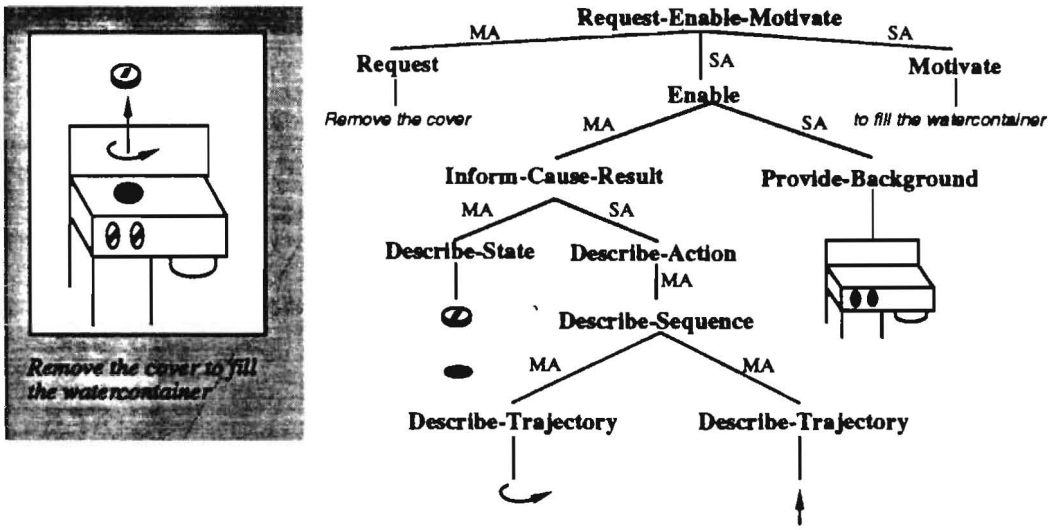


Fig. 1: A Document Fragment and its Structure

In Fig. 1, an example document fragment and its discourse structure are shown. The goal of this document fragment is to instruct the user in removing the cover of the water container of an espresso machine. The instruction can be considered as a composite goal comprising a request, a motivation and an enablement part. The request is conveyed through text (main act (MA)). To motivate that request, the author has referred to a superordinate goal, namely filling the water container (subsidiary act (SA)). The picture provides additional information which enables the addressee to carry out the request (subsidiary act). The generation of the picture is also subdivided into a main act, which describes the result and the actions to be performed, and a subsidiary act, which provides the background to facilitate orientation.

3 Realization of the Presentation Planning System

As argued in the preceding section, text-picture combinations follow similar structuring principles as text. In particular, a document is characterized by its intentional structure that is reflected by the presenter's intentions and by its rhetorical structure that is reflected by various coherence relations. Therefore, it is quite natural to extend methods for text planning in such a way that they become also useful for multimodal presentations.

In order to generate multimodal presentations, we have defined a set of presentation strategies that can be selected and combined according to a particular presentation task. To represent presentation strategies, we follow the approach proposed by Moore and Paris (cf. [Moore/Paris 89]) to operationalize RST for text planning. However, an additional slot for the presentation mode must be introduced. The strategies are represented by a name, a header, an effect, a set of applicability conditions and a specification of main and subsidiary acts. Whereas the header of a strategy is a complex communicative act (e.g., to enable an action), its effect[2] refers to an intentional goal (e.g., the user knows a particular object). The applicability conditions specify when a strategy may be used, and constrain the variables to be instantiated.

```
Name:
   Provide-Background
Header:
   (Background P A ?x ?p-x ?picture Graphics)
Effect:
   (BMB P A (Encodes ?p-x ?x ?picture))
Applicability Conditions:
   (AND        (Bel P (Perceptually-Accessible A ?x))
               (Bel P (Part-of ?x ?z)))
Main Acts:
   (S-Depict P A (Object ?z) ?p-z ?picture)
Subsidiary Acts:
   (Achieve P (BMB P A (Encodes ?p-z ?z ?picture)) ?mode)
```

E.g., the strategy above may be used to enable the identification of an object shown in a picture by depicting a landmark object (for more details see [André/Rist 93]). Whereas the strategy prescribes graphics for the main act, the mode for the subsidiary acts is still open.

For the automatic generation of illustrated documents, the presentation strategies are treated as operators of a planning system. During the planning process, the planner tries to find strategies that match the presentation goal and expands the nodes to generate a refinement-style plan in the form of a directed acyclic graph (DAG). The leaves of the planning DAG are specifications for elementary acts, such as S-Depict or S-Assert, that are forwarded to the mode-specific generators. The planning process terminates if all goals are expanded to elementary acts that can be realized by the text or graphics generator.

To ensure that document fragments in multiple modalities are smoothly tailored to each other in the document to be generated, one has, however, to consider various dependencies between content determination, mode selection and content realization. Previous work on natural language generation has shown that content selection and content realization should not be treated independently of each other (see also [Hovy 87] and [Reithinger 91]). A strictly sequential model in which data only flow from the "what to present" to the "how to present" part has proven inappropriate because the components responsible for selecting the contents would have to anticipate all decisions of the realization components. This problem is compounded if, as in WIP, content realization is done by separate components (currently a text and a graphics generator) of which the content planner has only limited knowledge.

It seems even inappropriate to sequentialize content planning and mode selection although mode selection is only a very rough decision about content realization. Selecting a mode of presentation depends to a large extent on the nature of the information to be conveyed. On the other hand, content planning is strongly influenced by previously selected mode combinations. E.g., to graphically refer to a physical object, we need visual information that may be irrelevant to textual references.

A better solution is to interleave content planning, mode selection and content realization. In the WIP system, we interleave content and mode selection using a uniform planning mechanism. In contrast to this, presentation planning and content realization are performed by separate components that access disparate knowledge sources. This modularization enables parallel processing, but makes interaction between the single components necessary.

---

[2] In [Moore/Paris 89], this distinction between header and effect is not made because the effect of their strategies may be an intentional goal as well as a rhetorical relation.

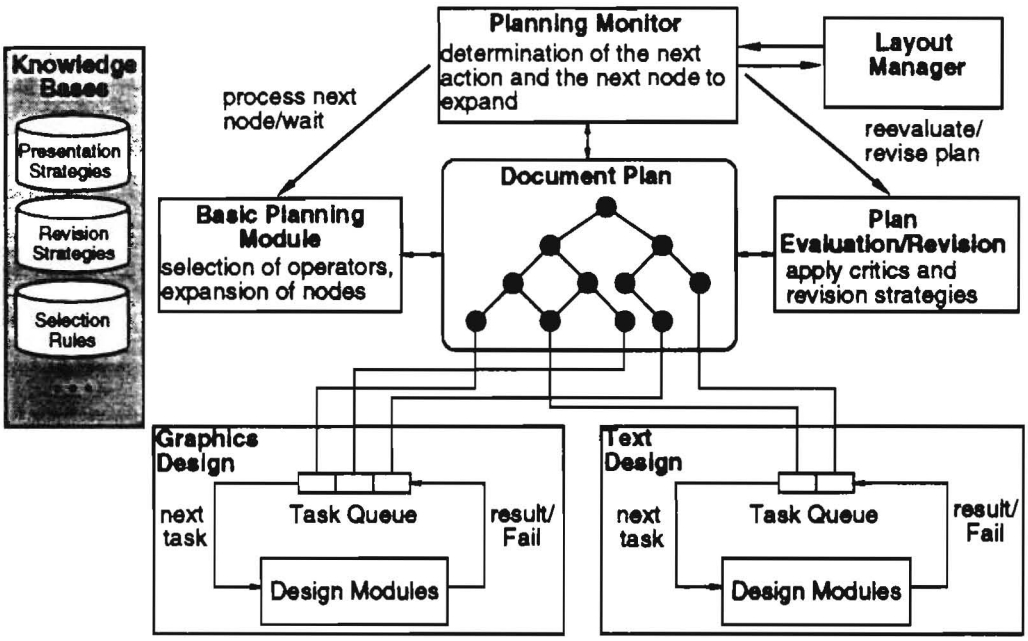


Fig. 2: The Architecture of the Presentation Planner

Interactions are, however, only useful if the realization components are able to process information in an incremental manner. As soon as the content planner has decided which generator should encode a certain piece of information, this piece should be passed on to the respective generator. Conversely, the content planner should incorporate the results of the realization components as soon as possible.

These considerations have led to the architecture shown in Fig. 2. The basic planning module selects operators that match the presentation goal and expands the nodes to generate a refinement-style plan in the form of a DAG. The plan evaluation/revision module evaluates plans and applies restructuring methods. To allow for alternating revision and expansion processes, WIP's presentation planner is controlled by a plan monitor that determines the next action and the next nodes to be expanded. All components of the presentation planner have read/write access to the document plan.

In the overall WIP system (cf. [Wahlster et al. 93]), the presentation planner collaborates with a text generator (cf. [Harbusch et al. 91]), a graphics generator (cf. [Rist/André 92]) and a layout manager (cf. [Graf 92]). As shown in Fig. 2, the leaves of the document plan are connected to entries in the task queues of the mode-specific generators. Thus, the document plan serves not only as an interface between the planner and the generators, but also enables a two-way exchange of information between the two generators.

4 Conclusion

In this paper, we have argued that not only the generation of text, but also the synthesis of multimodal documents can be considered as a communicative act which aims to achieve certain goals. We have shown that methods for planning text and discourse can be generalized in such a way that they become useful in the broader context of multimodal presentations. To represent knowledge about how to present information, we have proposed presentation strategies which relate to both text and picture production. For the realization of a system able to automatically generate illustrated documents, we have presented a plan-based approach that supports data transfer between the content planner and the mode-specific generators and allows for global plan evaluation after each plan step.

References

[André/Rist 90] E. **André** and T. **Rist**, Towards a Plan-Based Synthesis of Illustrated Documents, In: Proc. of ECAI-90, Stockholm, Sweden, pp. 25-30, 1990.

[André/Rist 93] E. André and T. Rist, The Design of Illustrated Documents as a Planning Task, to appear in: M. Maybury (ed.): Intelligent Multimedia Interfaces, AAAI Press, 1993.

[Arens et al. 93] Y. **Arens**, E. **Hovy** and M. **Vossers**. The Knowledge Underlying Multimedia Presentations, to appear in: M. Maybury (ed.): Intelligent Multimedia Interfaces, AAAI Press, 1993.

[Badler et al. 91] N.I. **Badler**, B.L. **Webber**, J. **Kalita** and J. **Esakov**. Animation from Instructions. In: N.I. Badler, B.A. Barsky and D. Zeltzer (eds.), Making them Move: Mechanics, Control, and Animation of Articulated Figures, Morgan Kaufmann Publishers: San Mateo, pp. 51-93, 1991.

[Bandyopadhyay 90] S. **Bandyopadhyay**. Towards an Understanding of Coherence in Multimodal Discourse. Technical Memo DFKI-TM-90-01, Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken, 1990.

[Brandt et al. 83] M. **Brandt**, W. **Koch**, W. **Motsch** and I. **Rosengren**. Der Einfluß der kommunikativen Strategie auf die Textstruktur - dargestellt am Beispiel des Geschäftsbriefes. In: I. Rosengren (ed.), Sprache und Pragmatik, Lunder Symposium 1982. Almqvist & Wiksell: Stockholm, pp. 105-135, 1983.

[Van Dijk 80] T. A. **van Dijk**. Textwissenschaft. dtv: München, 1980.

[Feiner/McKeown 91] S.K. **Feiner** and K.R. **McKeown**. Automating the Generation of Coordinated Multimedia Explanations. In: IEEE Computer, 24(10), pp. 33-41, 1991.

[Graf 92] W. **Graf**. Constraint-Based Graphical Layout of Multimodal Presentations, to appear in: Proc. of the International Workshop on 'Advanced Visual Interfaces', Rome, May 27 - 29 1992, World Scientific Press, Singapore, 1992.

[Grimes 75] J.E. **Grimes**. The Thread of Discourse. Mouton: The Hague, Paris, 1975.

[Harbusch et al. 91] K. **Harbusch**, W. **Finkler** and A. **Schauder**, Incremental Syntax Generation with Tree Adjoining Grammars, in: Proc. Fourth International GI Congress, Munich, Germany, pp. 363-374, 1991.

[Hobbs 78] J. **Hobbs**. Why is a discourse coherent? Technical Report 176, SRI International, Menlo Park, CA, 1978.

[Hovy 87] E. H. **Hovy**. Generating Natural Language under Pragmatic Constraints. PhD thesis, Department of Computer Science, Yale University, New Haven, CT., 1987.

[Hunter et al. 87] B. **Hunter**, A. **Crismore** and P.D. **Pearson**. Visual Displays in Basal Readers and Social Studies Textbooks. In: D.M. Willows and H. A. Houghton (eds.), The Psychology of Illustration, Basic Research, Vol. 2, Springer: New York, Berlin, Heidelberg, London, Paris, Tokyo, pp. 116-135, 1987.

[Kjorup 78] S. **Kjorup**. Pictorial Speech Acts. In: Erkenntnis 12, pp. 55-71, 1978.

[Levin et al. 87] J.R. **Levin**, G.J. **Anglin** and R. N. **Carney**. On Empirically Validating Functions of Pictures in Prose. In: D.M. Willows and H. A. Houghton (eds.), The Psychology of Illustration, Basic Research, Vol. 1, Springer: New York, Berlin, Heidelberg, London, Paris, Tokyo, pp. 51-85, 1987.

[Mann/Thompson 87] W.C. **Mann** and S.A. **Thompson**. Rhetorical Structure Theory: Description and Construction of Text Structures. In: G. Kempen (ed.), Natural Language Generation: New Results in Artificial Intelligence, Psychology, and Linguistics, Nijhoff: Dordrecht, Boston, Lancaster, pp. 85-95, 1987.

[Marks/Reiter 90] J. **Marks** and E. **Reiter**. Avoiding Unwanted Conversational Implicatures in Text and Graphics. In: 8th AAAI, pp. 450-455, 1990.

[Maybury 93] M. **Maybury**. Planning Multimedia Explanations Using Communicative Acts, to appear in: M. Maybury (ed.): Intelligent Multimedia Interfaces, AAAI Press, 1993.

[Moore/Paris 89] J.D. **Moore** and C.L. **Paris**. Planning Text for Advisory Dialogues. In: Proc. of the 27th Annual Meeting of the ACL, pp. 1504-1510,1989.

[Rist/André 92] T. **Rist** and E. **André**. From Presentation Tasks to Pictures: Towards a Computational Approach to Graphics Design, in: Proc. of ECAI-92, pp. 764-768, Vienna, Austria, 1992.

[Roth et al. 91] S.F. **Roth**, J. **Mattis** and X. **Mesnard**. Graphics and Natural Language as Components of Automatic Explanation. In: J. Sullivan and S. Tyler (eds.), Intelligent User Interfaces: Elements and Prototypes, Addison-Wesley, pp. 207-239, 1991.

[Reithinger 91] N. **Reithinger**. POPEL - An Incremental and Parallel Natural Language Generation System. In: C.L. Paris, W.R. Swartout, and W.C. Mann (eds.), Natural Language Generation in Artificial Intelligence and Computational Linguistics, pp. 179-200, Kluwer: Norwell, 1991.

[Searle 69] J.R. **Searle**. Speech Acts: An Essay in the Philosophy of Language. Cambridge University Press: Cambridge, MA, 1969.

[Wahlster et al. 93] W. **Wahlster**, E. **André**, W. **Finkler**, H.-J. **Profitlich** and T. **Rist**. Plan-Based Integration of Natural Language and Graphics Generation. to appear in: AI Journal, 1993.

$\mathcal{NLL}$ — Tools for Meaning Representation

John Nerbonne, Kader Diagne, Stephan Oepen,
Karsten Konrad, Ingo Neis
Deutsches Forschungszentrum für Künstliche Intelligenz
Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, Germany
{nerbonne | diagne | oe | konrad | neis}@dfki.uni-sb.de

Abstract

The paper discusses a concrete and implemented application of the meaning representation language $\mathcal{NLL}$ and the use of compiler technology both in semantics processing and in interfacing a computational linguistics core engine to a distributed AI application system. Various steps necessary in simplifying and resolving an initial (grammar-driven) meaning representation and mapping it into an appointment planner language, i.e. an application-specific logical form, are presented by virtue of detailed examples from the application domain (cooperative schedule management).

1 Introduction: Design Goals in Meaning Representation

The focus of this paper is the design and IMPLEMENTATION of semantic representation languages (SRLs). Given the need of such modules to represent natural language meanings, we assume that they should apply linguistic semantics, the specialized study of natural language meaning. But because of the focus on design and implementation, we examine quite generally the uses to which such modules may be put, abstracting away from details which distinguish such superficially distinct approaches as generalized quantifier theory (GQT), discourse representation theory (DRT), situation theory, or dynamic logic.

The appropriate design for any module can only be determined by close analysis of the uses to which it is to be put. We do not consider applications for semantics in providing test tools for linguistic hypotheses or in adding constraints to recognition tasks ([Young et al. 1989]), because these provide less clear design criteria. But the numerically most important group of applications, that of understanding and generation, give rise to fairly clear requirements. Independent semantics modules are used with these applications for semantic representation, inference, and in order to support meaning-related processing—disambiguation, resolution, and speech act management. The importance of these points confirms the good sense of current practice in the field—that of viewing the main task of the semantic module as the implementation of a linguistic semantic theory (with selected AI enhancements for resolution and disambiguation). But we suggest that insufficient attention is paid to the following four goals, which we focus on below:

Goals	PL	SRL
modularity	independent definition (BNF)	
tools	parser, printer	parser (%), printer
modifiability	parser-generator	
mappings in, out	compiler	
inference	program transformation	resolution, backward-chaining ...

Figure 1: Design goals common to PLs and SRLs plotted against "standard" solutions in the two areas. The analogy suggests filling the gaps for standard solutions for SRLs by using PL solutions: language specification tools for definition together with parser-generators to provide the SRL reader, and compiler technology for interfaces to semantics modules. Finally, program transformation techniques suggest a simple implementation for at least some inference rules.

- modularity—independence from syntax and application
- modifiability—for experimentation
- interface support—for mapping into and out of module
- support for independent use (reader, printer, tracer)

These prompt us to a comparison to programming language technology and compiler construction.

2 Analogy to Programming Languages

It is axiomatic that modern PLs should meet the last four goals listed in the design goals for SRLs. Standard introductions ([Aho et al. 1986]) detail how a programming language syntax is specified in definitions independent of specific machines and environments (modularity) which are, moreover, easily modifiable given tools for parsing (parser-generators) and printing. The parsers automatically created from language specifications take well-formed strings as input and produce abstract syntax trees (like linguistic parse trees) as output. Modern tools also provide printers (unparsers) which reverse the process: given an abstract syntax tree, they produce a print form ([Friedmann et al. 1992], 85ff).

Just as a modular SRL must interface to more than one application, a programming language needs to be able to run on different machines. In the latter case, this is accomplished by compiling: the abstract syntax trees produced by the PL's parser are transformed into (the abstract syntax trees representing) expressions of another lower-level language (often a machine-specific assembler language). The SRL correspondence is the use of compiler technology to translate into SRLs, viz., in syntax/semantics interfaces (in NLU) or application/semantics interfaces (in generation). Further details below.

Some of the transformations performed by compilers are not simple translations into target languages, but rather transformations to alternative structures in the source language (cf. [Aho et al. 1986], 592ff), or immediate evaluation of parse structures (cf. "translation during parsing" [Aho et al. 1986], 293–301). The use of these techniques suggests an implementation for some inference facilities for SRLs—those arising from equivalence rules.

3 $\mathcal{NLL}$: Data Structures and Interfaces

$\mathcal{NLL}$ is an SRL which borrows heavily from linguistic semantics in order to provide representational adequacy, using, e.g., on the one hand work from generalized quantifier theory and on the other from the logic of plurals. [Laubsch et al. 1991] and [Nerbonne 1992] present an overview of $\mathcal{NLL}$ and the background linguistic and model-theoretic ideas, which will not be repeated here. For the sake of understanding examples below, we note that atomic formulas in $\mathcal{NLL}$ are composed of a predicate together with a set of role-argument pairs, e.g.

'Anterist ships to Hamburg' **ship(agent:'anterist' goal:'hamburg')**

The $\mathcal{NLL}$ formula may also be read: **'anterist'** plays the **role** of agent and **'hamburg'** that of goal in some shipping situation. An advantage of identifying arguments via roles rather than positions (as is customary in predicate logic) is that one can sensibly use the same predicate, e.g., **ship()**, with various numbers of arguments; thus even though something must also play a **theme** role in this situation (what is shipped), it need not be expressed in the role-coded set of arguments. Cf. [Nerbonne 1992] for formal development.

Following the PL lead, we begin with a formal syntactic specification of $\mathcal{NLL}$ in a form usable by a parser-generator.[1] We use **Zebu** ([Laubsch 1992a]), a public-domain tool in Common LISP.[2] **Zebu** grammar specifications consist of a set of RULES, each of which specifies a SYNTAX for a grammatical category and an ACTION to be taken by the parser when the category is found. These specifications are easily modified in case extensions, variations or even substantial modifications of the language become interesting. In addition to syntax rules, **Zebu** grammars may also contain lexical restrictions ([Laubsch 1992a], 15) needed for generating a lexical analyzer (which, however, is not used in $\mathcal{NLL}$). From the $\mathcal{NLL}$ grammar, **Zebu** generates an LALR(1) parser ([Aho et al. 1986], § 4), which is the $\mathcal{NLL}$ reader. The reader immediately supports experiments with the semantics module by easing the creation of semantic data structures. The **Zebu** grammar compilation process detects any inconsistencies or ambiguities in the grammar definition.

Zebu goes beyond the capabilities of parser-generators such as UNIX yacc in further optionally generating (automatically) the definition of a DOMAIN, a hierarchy of data structures (LISP structures) for abstract syntax. If this option is chosen, then **Zebu** defines a structure type for each expression type; the structure for a given expression has as many fields as the expression has subexpressions (e.g., Predicate and Role-Argument-Pairs). On the basis of the domain **Zebu** then also generates an "unparser", in this case the $\mathcal{NLL}$ printer (which in turn may be called by the LISP printer).

The dual access provided by the PL approach already allows an interesting degree of freedom. For example, the opportunity to create $\mathcal{NLL}$ via constructor functions allows the implementation of a syntax-semantics interface of the sort suggested by [Johnson et al. 1990]

[1] We are concentrating here on the more recent $\mathcal{NLL}$ implementation; an earlier implementation in REFINE ([Laubsch et al. 1991]) is no longer the focus of our efforts, even though we continue to maintain it for its usefulness in rapid prototyping. REFINE is a trademark of Reasoning Systems, Palo Alto.

[2] **Zebu** was originally developed in Scheme by William Wells.

in which the syntax/semantics interface is constituted by a set of generic constructor functions attached to syntactic rules (and therefore nonterminal nodes). $\mathcal{NLL}$ has been employed this way in a syntax/semantics interface in an extensive NLP system, viz. the hp-nl system ([Nerbonne et al. 1987]). This is appropriate when relatively complex structures are created in a series of simple increments. Alternatively, one may invoke the $\mathcal{NLL}$ reader to create $\mathcal{NLL}$, and an interface from the COSMA appointment manager (cf. below) to $\mathcal{NLL}$ (for generation) invokes the reader extensively. This made the single-step creation of complex structures much simpler.

But given the relatively easy access to abstract syntax trees provided by the Zebu reader, the construction of interfaces through genuine compilation (transformation based on abstract syntax) is also feasible. $\mathcal{NLL}$'s basic scheme of compilation is TREE REWRITING ([Aho et al. 1986], 572ff). An abstract syntax tree is traversed, and at each node, each of a sequence of REWRITE RULES is applied. A rewrite rule abstractly takes the form:

$$\textit{meta-syntactic-pattern} \rightarrow \textit{replacement-node}$$

A rewrite rule checks whether a *meta-syntactic-pattern* is satisfied at the current node, and returns the replacement node together with a boolean flag indicating whether the rule has fired: (*replacement-node*, δ?:bool). In case we are translating from one abstract syntax to another, then the meta-syntactic-pattern describes a node in the source language, while the replacement node belongs to the translation target language. The traversal routine replaces the current node with the replacement-node in case the rule has fired.

The top-down tree-rewriting algorithm PREORDER-TRANSFORM inputs a tree t and a sequence $(r_1 \ldots r_n)$ of rewrite rules. It then traverses the tree in preorder ([Aho et al. 1983], 78–9), and at each node, attempts to rewrite using each of the rules r_i. If any rule in the sequence fires, then the entire sequence is tried again, until no rules fire. Then the traversal continues, until the leaf nodes of the tree. The algorithm is attractive because it reduces the tree-transformation problem to the specification of transformations on local subtrees. An analogous POSTORDER-TRANSFORM invokes sequences of rewrite rules in a bottom-up traversal of the abstract syntax tree.

In addition to optimized control routines for tree-transformation $\mathcal{NLL}$ provides a library of access and manipulation functions (for substitution, construction, simplification) to support the transformation process. [Laubsch 1992] reports on the required transformations for $\mathcal{NLL}$ compilers to SQL and to the New Wave task language the following sections present transformations implemented in the COSMA system, a distributed cooperate appointment and schedule manager.

Compilation is normally an effective translation technique because it abstracts away from irrelevant details of the concrete syntaxes of target and source languages. It is especially appropriate: (i) when communication between modules is limited (e.g., when modules run on separate machines or in separate processes, so that communication is limited to strings); (ii) when the nature of target data structures is unknown or unspecified; or (iii) when there is minor variability in targets (e.g., different versions of the same programming language or query language).

4 $\mathcal{NLL}$ in the Scope of the COSMA System

In building the COSMA (Cooperative Schedule Management Agent) system we connected a natural language core engine (viz. the DISCO system) to an appointment planner that was provided by another in-house project. When interfacing the DISCO meaning processing modules to the planner internal representation language (henceforth IR) the compiler approach as taken in $\mathcal{NLL}$ design has been proven especially useful because the target language has undergone syntactic change several times.

The $\mathcal{NLL}$ transformations implemented so far in the COSMA prototype can be classified according to theory vs. domain dependence and target language ($\mathcal{NLL}$ vs. IR) as follows:

(I) **core $\mathcal{NLL}$ inference**: purely logical (theory-independent) equivalence transformation, e.g. flattening of nested conjunctions (example below)

(II) **simplification**: meaning preserving transformation generating a 'canonical' form, e.g. grouping of multiple restrictions on the same variable (example below)

(III) **disambiguation and domain-specific inference**: transformation based on domain knowledge, e.g. scope resolution, anchoring of underspecified temporal locations

(IV) **translation**: mapping into *different* abstract syntax — mediation of difference in expressive power, e.g. translation to SQL or the COSMA planner internal representation (example below)

5 Initial Sentence Semantics ($\mathcal{NLL}$ Representation of Parse)

Based on a concrete example from the appointment domain, viz. the noun phrase

(1) Ein Termin am 23. Oktober um 13:30 Uhr. (*A meeting on October 23 at 1:30 pm.*)

the following sections will present in some detail the major transformation steps that are taken in mapping the initial sentence semantics into an appropriate application directive. As in the current DISCO architecture the disambiguation and domain-specific inference components are still under development we focus on steps (I), (II) and (IV).

The initial semantics as input to the semantics module in feature structures is heavily determined by purely syntactic conditions, i.e. the structure derived from the underlying grammar. Because (1) is rich in prepositional phrases (free adjuncts in the HPSG-style DISCO grammar), the attachment problem shows up in the grouping of restrictions on the *restricted parameter* ?x in Figure 2. Basically, the variable ?x is constrained to be an instance of an appointment (termin()) and to stand in various spatial relations to expressions originating from the prepositional phrases *on October 23* and *at 1:30 pm*.

6 Equivalence Transformation and Simplification

An instance of logical equivalence transformation relevant to the given example is the following:

- **flattening of nested conjunctions:**

  $\mathsf{and}\{p, \mathsf{and}\{q, r\}\} \rightarrow \mathsf{and}\{p, q, r\}$
  $\mathsf{and}\{p\} \rightarrow p$

```
(?x | and{and{termin(instance:?x)
              temp-um(theme:?x goal:(?t2 | time(instance:?t2 hour:13 min:30)))}
        temp-am(theme:?x
                goal:(?t1 | time(instance:?t1 mon:(?m | oktober(instance:?m)) day:23)))})
```

Figure 2: (Part of the) Parsing result of the phrase 'Ein Termin am 23. Oktober um 13:30 Uhr.' after simple translation into $\mathcal{NLL}$. The nesting and order of conjuncts in the restriction of the variable ?x is semantically arbitrary, reflecting the grammatically convenient treatment of adjunct semantics in the DISCO grammar by introduction of additional conjunctions.

Applied to the $\mathcal{NLL}$ expression in Figure 1 the nested binary conjunctions restricting the variable ?x are 'flattened' to a single conjunction with three order-independent conjuncts.

While a set of core logical rewrite rules comes built-in with the $\mathcal{NLL}$ semantics module, the second class of transformations listed above, viz. the generation of a 'canonical' form by simplification, depends on the concrete linguistic theory of semantics assumed. With respect to example (1) we exclusively look at the representation of temporal constraints in the DISCO framework.

The basic format of temporal expressions is given by the atomic predicate time() with roles instance, year, month, day etc. down to second — we think of formulae of this type as representing either a moment or interval of time (much like a temporal location in situation theory) with varying granularity (cf. [Kasper 1993]). However, as Figure 2 shows, the actual distribution of temporal data from the example phrase is determined by the number of prepositional phrases and the syntactical nesting of adjuncts. Moreover, the two chunks of temporal restrictions on the appointment stand in the scope of two distinct predicates (viz. temp-am() and temp-um())[3] as they are lexically introduced by the corresponding German prepositions.

Two similar examples from the DISCO grammar yield structurally quite different results (with temporal units abbreviated by the corresponding English phrases):

- 'am Freitag um 13:30 h' (*on Friday at 1:30 pm*)

  temp-am(... goal:(?t1 | and{time(instance:?t1 *Friday*)
  temp-um(theme:?t1
  goal:(?t2 | time(instance:?t2 *1:30 pm*)))

- 'am Nachmittag des 23.10.' (*on the afternoon of October 23* [genitive case])

  temp-am(... goal:(?t1 | and{time(instance:?t1 *afternoon*)
  poss(possessed:?t1
  possessor:(?t2 | time(instance:?t2 *23.10.*)))

The first case exhibits typical nesting of prepositional phrases while the second example reflects a (grammatically) comfortable treatment of genitive attributes. All example configurations show the need for simplification into a canonical representation, thus abstracting

[3]The temp- prefix results from sortal disambiguation based on the type of complements (both temporal in this case) as it is integrated into the feature structure (meta-) representation of $\mathcal{NLL}$ expressions and evaluated at parse time. Cf. [Kasper 1993].

```
(?x | and{termin(instance:?x)
        temp-in(theme:?x goal:(?t | time(instance:?t
                                         mon:(?m | oktober(instance:?m)) day:23
                                         hour:13 min:30))))})
→ ((type . :appointment)
   (time . ((month . :october) (day-of-month . 23) (hour . 13) (minute . 30))
```

Figure 3: $\mathcal{NLL}$ representation of example phrase after equivalence transformation and simplification (top) and corresponding structure in COSMA IR (bottom). Nested conjunctions and multiple restrictions on the variable ?x have been simplified in the $\mathcal{NLL}$ formula according to the rewrite rules given in section 3 and the generic **temp-in()** relation has been substituted for the lexically determined predicates.

from syntactic structure and allowing uniform further processing. The DISCO general time semantics legitimates the grouping of multiple (and possibly nested) temporal restrictions on the same variable into the scope of a single **time()** predicate by 'unification' of all existing information. For all the given examples $\mathcal{NLL}$ transformations have been implemented[4] that fire on appropriate meta-syntactic configurations and rewrite to a unified structure.

7 Translation into Target Language

The internal representation of the appointment planner as the interface language to the semantics module in the COSMA system (besides the interface to feature structure descriptions) is far more restricted in expressive power than $\mathcal{NLL}$. The IR language is syntactically close to Common-Lisp (all data structures are represented as nested association lists) — it basically provides a fixed set of negotiation primitives that take one or more appointment descriptions (of the type given in Figure 3) as arguments.

Despite the obvious simplicity of the target language in the COSMA prototype we have chosen to follow the general $\mathcal{NLL}$ compiler approach in mapping into IR. Just as inside the $\mathcal{NLL}$ module expressions of the target language are represented as (hierarchically organized) Common-Lisp structures thus allowing us to view the process of translation as a series of tree transformations again and to make use of the optimized $\mathcal{NLL}$ tree traversal drivers. The Common-Lisp surface syntax of IR expressions is generated from a (yet hand-coded) pretty printer (unparser) that makes IR abstract syntax trees print as Lisp association lists.

Adapting the compiler approach to the $\mathcal{NLL}$ to IR interface, the translation module is construed as a set of rewrite rules that are applied in post-oder (bottom-up) tree traversal[5] just like the other transformations and simplifications we have seen so far. Though the

[4] In the current state of $\mathcal{NLL}$ implementation rewrite rules are realized as hand-coded Common-Lisp functions that are applied to abstract syntax trees by use of an optimized tree traversal driver. Ongoing $\mathcal{NLL}$ research and implementation is devoted to the realization of a rule compiler that compiles appropriate transformation routines from *meta-syntactic* descriptions of rewrite rules.

[5] Bottom-up processing is the control regime of choice because several high-level rewrite rules depend on the argument type of embedded structures. Generally a mixed control strategy might be achieved by splitting transformations into (not necessarily disjoint) sets and applying these sequentially.

example phrase has been chosen because it maps nicely into an appropriate IR expression (see Figure 3) by strictly local rewrite rules the tree transformation mechanism has turned out to be suitable in handling larger and more complex structures as well and to effectively support the mediation of different expressive power needed in mapping $\mathcal{NLL}$ into COSMA IR.

References

[Aho et al. 1983] Alfred Aho, John Hopcroft and Jeffrey Ullman. *Data Structures and Algorithms.* Addison Wesley, 1983.

[Aho et al. 1986] Alfred Aho, Ravi Sethi and Jeffrey Ullman. *Compilers: Principles, Techniques and Tools.* Addison Wesley, 1986.

[Friedmann et al. 1992] Daniel Friedman, Mitchell Wand and Christopher Haynes. *Essentials of Programming Languages.* McGraw-Hill, New York, 1992.

[Johnson et al. 1990] Mark Johnson and Martin Kay. Semantic abstraction and anaphora. In Hans Karlgren, editor, *Proceedings of COLING-90*, pages 17–27, Helsinki, 1990. COLING.

[Kasper 1993] Walter Kasper. Integration of Syntax and Semantics in Feature Structures. In Stephan Busemann and Karin Harbusch, editors, *DFKI Workshop on Natural Language Systems: Modularity and Reusability*, DFKI Document, DFKI, Saarbrücken, 1993.

[Laubsch et al. 1991] Joachim Laubsch and John Nerbonne. An Overview of $\mathcal{NLL}$. Technical Report, Hewlett-Packard Laboratories, Palo Alto, 1991.

[Laubsch 1992] Joachim Laubsch. The semantics application interface. In Hans Haugeneder, editor, *Applied Natural Language Processing*, 1992.

[Laubsch 1992a] Joachim Laubsch. **Zebu**: A tool for specifying reversible lalr(1) parsers. Technical report, Hewlett-Packard Laboratories, Palo Alto, CA, July 1992.

[Nerbonne et al. 1987] John Nerbonne and Derek Proudian. The hp-nl system. Technical report, Hewlett-Packard Labs, 1987.

[Nerbonne 1992] John Nerbonne. $\mathcal{NLL}$ Models. Research Report, DFKI, Saarbrücken, 1992.

[Nerbonne et al. 1993] John Nerbonne, Joachim Laubsch, Kader Diagne and Stephan Oepen. Natural Language Semantics and Compiler Technology. Research Report, DFKI, Saarbrücken, 1993.

[Neumann et al. 1993] Günter Neumann, Stephan Oepen and Stephen P. Spackman. Design and Implementation of the COSMA system. Technical Report, DFKI Saarbrücken, 1993

[Young et al. 1989] Sheryl Young, Alexander Hauptmann, Wayne Ward, Edward Smith, and Philip Werner. High level knowledge sources in usable speech recognition systems. *Communications of the ACM*, 32(2):183–194, 1989.

Incremental Lexical Choice Constrained by Generation Parameters

Karin Harbusch

phone: (+49 681) 302 5271
e-mail: harbusch@dfki.uni-sb.de

Abstract

In the following paper, principal ideas for an incremental lexical choice process guided by generation constraints during the incremental natural language generation in the WIP[1] system are presented. In this approach, a most adequate expression for an arbitraryly complex conceptual description is processed by examining a list of constraining generation parameters like time to produce the result, style of the produced text, habituality, etc. The component works incrementaly in the sense that knowledge arriving later leads to further constraints on the selection process. In the following paper, basic concepts of our first prototype for the incremental lexical choice — e.g., an anytime approach — are described.

1 General Suggestions on the Lexical Choice Process in Natural Language Generation

The acceptance of an AI system depends directly on its user interface facilities. For the direction of output this means that situationally adequate text — instead of canned text prepared for a general situation — should be produced (e.g., the generation of a summarizing statement to avoid repetitions of the same canned text).

The process of *natural language generation* can be divided into three major tasks (see, e.g., [Levelt 89]):

- *Macro planning (MAP)* means the elaboration of the communicative intention as a sequence of subgoals, and the selection of information to be expressed in order to realize these communicative goals.

[1]WIP is the acronym for "Wissenbasierte Informationspräsentation". The WIP project is supported by the German Ministry of Research and Technology under grant ITW8901 8.

- *Micro plannung (MIP)* consists of the conceptual planning which includes decisions like determination of the information perspective, its topic, its focus and the way in which it could attract the addressee's attention.

- *Sentence formulaton (SF)* involves the tasks of accessing lemmas, inspecting the grammatical relations, and mapping these onto inflectional and phrasal structure to form the surface structure given to the phonological encoder.

It has been shown that the boundaries between the individual tasks cannot be drawn separately because for many decisions, knowledge of another component is required. This can lead to dead ends as a result of local decisions in MAP or MIP or SF, respectively. Therefore the architecture of a generation realizing MAP, MIP, and SF in individual components should provide features for backtracking or feedback (see, e.g., the system TEXT by [McKeown 85] for a sequential model or POPEL by [Reithinger 91] for a cascaded model with feedback). In an integrated approach this problem is avoided by allowing access to all knowledge sources for each task where the order of processing is not predetermined (see, e.g., the KAMP system by Appelt [Appelt 85]). The approach concerned here in the natural language generation in WIP follows the paradigm of Reithinger for a cascaded model with feedback [Wahlster et al. 93] because it provides the flexibility necessary for the incremental processing without the drawbacks of an integrated system.

Lexical choice (LC) is the process of selecting a lemma or a number of lemmas, respectively, describing a net of concepts and roles adequately in the current situation. With this characterization, the decision where to locate a component for lexical choice in a non-integrated generation system influences the performance of the system. If the lexical choice is postponed during sentence formulation, a propositional structure can be selected where the obligatory finite verb position cannot be filled by the lexical choice. Conversely, e.g., the lexical choice in the micro planning can determine a valency description for the verb where the obligatory positions cannot be filled later on in the sentence formulation.

In the incremental natural language generator in WIP where each component must be able to run with partial information and defaults — which can be revised by contradictory input — processed from a piece-meal wise input of a higher component the lexical choice is located in MIP. There it plays the role of a default generating routine where in a feedback step SF can add further constraints for a new search process in the lexical choice[2].

But we do not want to go as far as Levelt [Levelt 89] who proposes that the whole formulation process is lexically driven, i.e. the grammatical and phonological encoding are mediated by lexical entries. The preverbal message triggers lexical items into activity. The syntactic, morphological, and phonological properties of an activated lexical item trigger, in turn, the grammatical, morphological, and phonological encoding procedures underlying the generation of an utterance (lexical hypothesis). In his framework, e.g., the decision for active or passive constructions is triggered by mediating lexical items instead of a direct encoding in the speaker's message.

[2]See the syntactic constraints in the input description of the lexical choice component where these constraints are formulated.

Our decision against the lexical hypothesis results from the idea of building an LC component fitting into different generators which can or cannot follow the lexical hypothesis. Thus, in the input specification of the lexical choice component in WIP, a direct encoding is provided. We hope to subsume the lexical hypothesis by this approach in the sense that an empty specification in this position and an early call of the LC component in MIP produces the mediating lexical items, although this hypothesis must be discussed in more detail in future investigations.

In the following section, the current state of our incremental lexical choice component is described in more detail.

2 The Input Specification for the Lexical Choice Component in WIP

The input consists of two kinds of information. On the one hand a net of concepts and roles of arbitrary size is described. On the other hand a list of generation parameters is specified for each concept node. This list can be empty.

The net of concepts and roles is given descriptively by a uniquely interpretable bracket structure. A description of generation parameters can be associated with each concept node in the net.

The list of generation parameters which constrain all processes in the WIP system are the following:

- *processing time* for the task (here 5 different degrees ("*0*"-"*4*") are given as an abstract description instead of an absolute cpu time or an interrupt of an external control process),
- *space* for the utterance (same as for processing time an abstract description — at the moment simply a binary choice between "'*short*" and "*long*"),
- *style* of the utterance ("*written text*" or "*spoken language*" or "*slang*" etc.),
- *frequency* (on the latice between "*1*" and "*10*"), which is a specific parameter of lexical choice and not of the overall generation system, to select less common words – in order to allow for more flexible texts[3],
- another constraint most intensively interpreted[4] in the lexical choice process are the *syntactic constraints* which are specified, e.g., by a previous attempt in the sentence formulation process. They are written in the terminology of the lexicon, e.g., VAL=4 for a valency description with direct and indirect object.

[3]For the next prototype it should be discussed how this parameter should be extended — especially towards a well funded decision between hypernyms.

[4]In the case of a feedback from SF, MIP can also react, e.g., by deciding for another propositional structure.

Two further specific cases are encoded in the lexical choice process:

- The selection of the lexical entries describing the *relation between propositions* is initiated by the specification of the kind of relation (anteriority, reason, condition, etc.) and the specification of the kind of sentence in the particular role (main clause or subordinate clause or nominalization of a clause).
- The determination of *prepositions for semantic roles* will be produced by specifying the role and the selected lexical item for the corresponding concept as input.

Each input can be associated with the above mentioned generation parameters.

For easy processing in the LC component, all three kinds of input can be identified separately by reading the first sign (by convention "!" introduces all concepts and "$" all roles).

3 Description of the Lexica for our LC

Another input resource of the Lexical Choice process are the lexica where the correspondence between concepts and roles with lexical entries is specified.

The lexical choice component in WIP uses four different lexica:

- RELLEX contains the specifications of relations between sentences. For example, the relation *condition* between a subordinate and a main clause is "wenn - dann" in spoken language and "bevor - ." in written text.
- PREPLEX contains the prepositional items for semantic roles. For example, *locational_goal_on_top_of* is expressed as "auf+accusative_obj".
- PREPSPEC imposes further constraints on prepositions in combination with specific lexical items in the object position. For example, *location_goal_in* in combination with a "*Dessertteller*" becomes "auf" instead of "in".
- CONLEX contains a word or a list of words corresponding to a net of concepts and roles in order to express the net. For example , the net concept *motion_by_means* + role *instrument* + concept *plane* is related to "fliegen" for a verb and "Flug" for a noun in written text, with "düsen" for a verb in slang[5].

In the preliminary implementation to test the algorithm for the selection of lexical items all lexica are realized as assoc lists containing structures [Steele 84]. The key is a concept net, relation or preposition. In the individual structure the lexical item and the best characterization of constraints is described (e.g., *motion_by_means* with *(V (zischen_1) SYNTRESTR = (VAL = 1 MODE = ACTIVE) FREQ = (3) LENGTH = SHORT STYLE*

[5]This entry is annotated with a reduced frequency so that "fliegen" can also be a valid choice during the evaluation of generation constraints where "*slang* is required.

= SLANG) (N (heißer_Ritt_1) SYNTRESTR = (IDIOM GEN = MAS NUM = SING) FREQ = (1) LENGTH = LONG STYLE = SLANG) etc.

For large lexica more sophisticated methods of storing and searching should be adopted in the next prototype.

4 The Processing in our Incremental LC Component

By the input specification the three different kinds of processing in the lexical choice process can be identified separately by reading the first input sign. The first element (net of concepts and roles or relation or semantic role) becomes the search key for the lexica. In all three cases the generation parameters are associated with individual variables. If a generation parameter is not specified in the input it is initialized with the following default: *synrestr := nil, freq := 10, length := short, time:= 4, style := written_text.*

For all three types of processing the most important parameter during processing is the time constraint. If it is "*0*" the most primitive filter selects the lexical item of the first entry under the search key in the corresponding lexicon and returns it as an answer[6]. If the time restriction is greater than zero a number of nested filters is applied to the list of entries found under the search key in the specific lexicon. Each test produces a new list where the currently tested constraint is satisfied. If a test comes up with the empty list the resulting list of the last test becomes valid again. Therefore a result is always produced except when the key is not in the lexicon.

For the input of a net of concepts and roles the filtering is the following. The next filter checks the category restriction for a concept. Only entries with the required category remain in the resulting list. If time = "*1*" the first entry in the resulting lists becomes the answer. For time greater than *1* the further syntactic restrictions are checked. For time greater than *2* the style is tested. For time greater than *3* the length is considered. The most extended test selects entries with a frequency best fitting the input specification in this constraint position.

The processing of prepositions is carried out in the same fashion. For the relation between sentences no syntactic restriction beside the nominalization of a clause can be specified.

As a basic requirement of the WIP system the input can be incrementally extended. This can mean that either the concept–role net or the generation parameters are extended or modified. Since in the case of lexical choice an extension of the net can lead to different lexical items — e.g., if the instrument "*plane*" occurs at a later stage the choice for "fahren" must be revised – at the moment the whole processing in the LC component is repeated. In the next version, intermediate results should be considered and reused as far as possible. Especially, returning to the finally processed set of results should be possible if the time constraint has diminished i.e. the number is increased. This is an important step towards fulfilling the requirements of an anytime approach.

[6]This can be seen as a very basic *anytime algorithm* [Dean, Boddy 88] where the interrupt is simulated by an explicit specification in terms of a duration time.

5 Conclusions

Here a straight-forward realization of an incremental lexical choice under generation constraints is presented. The main emphasis placed on the consideration of generation constraints especially of the time restriction which always leads to a result which is more or less adequate.

The next steps go towards investigations in a more well funded theory of representation of relations between lexical items. As suggested in [Pustejovsky 91], techniques known from knowledge representation can be adapted for that purpose (e.g., the lexical choice can be interpreted as a classifying problem in this framework).

There are ideas of taking a more general and more flexible approach in running the filters according to the current specification of constraints. As realized in the sentence generator of the WIP system for the selection of grammar rules, according to the generation parameter a geometric representation of all constraints is computed [Kilger 93]. For this measurement a more or less restricted approximation is required — in our case in the lexicon where entries are represented accordingly by this global measurement.

References

[Appelt 85] D. E. Appelt. *Planning English Sentences.* Cambridge University Press, Cambridge, MA, 1985.

[Dean, Boddy 88] T. Dean and M. Boddy. An analysis of time-dependent planning. In *AAAI-88*, pages 49–54, 1988.

[Kilger 93] A. Kilger. Incremental generation with tree adjoining grammars. *Computational Intelligence*, 1993. to appear.

[Levelt 89] W. J. M. Levelt. *Speaking: From Intention to Articulation.* MIT Press, Cambridge, MA, 1989.

[McKeown 85] K.R. McKeown. *Text generation.* University Press, Cambridge, MA, 1985.

[Pustejovsky 91] J. Pustejovsky. The generative lexicon. *Computional Linguistics*, 17(4):409–441, 1991.

[Reithinger 91] N. Reithinger. *Eine parallele Architektur zur inkrementellen Generierung multimodaler Dialogbeiträge.* PhD thesis, University of the Saarland, 1991.

[Steele 84] G. L. Steele. *Common Lisp: The Language.* Digital Press, Hanover, MA, 1984.

[Wahlster et al. 93] W. Wahlster, E. André, W. Finkler, H.-J. Profitlich, and T. Rist. Plan-based Integration of Natural Language and Graphics Generation. *Artificial Intelligence*, to appear, 1993.

Towards the Configuration of Generation Systems: Some Initial Ideas

Stephan Busemann*
busemann@dfki.uni-sb.de

1 Introduction

Many systems for NL generation that serve as NL front end to some application system (e.g. a database, or an expert system) are designed in such a way that general, linguistic parts are not carefully distinguished from application-dependent parts. Often decisions made in the course of generation implicitly rely on properties of the application system, or they are based on knowledge that combines domain-specific aspects with more general ones. For instance, a generator used in dialogues about appointment scheduling might deterministically generate the speech act REQUEST if it encounters the domain concept **arrange** since, in this application situation, **arrange** always represents the intention of arranging a meeting.

If such domain dependencies are placed in a generation front end, it is difficult, or even impossible, to transport the system to other applications. There is an obvious need for reusable generation components. What is called for is a better way of modularization.

The problem is well known, and partial solutions have been suggested. For instance, transportable surface generators do exist and are used for various application classes. Beyond the surface-oriented levels, however, there is as yet no overall design strategy for generators that would help in achieving better modularity and reusability.

There is an obvious reason for this deficiency. The large variety of possible generation tasks, including discourse generation, dialog contributions and machine translation[1], obviously cannot be dealt with by one single system. Moreover, the kind of input given to a generator by an application system is in no way standardized. For instance, focus assignment may or may not be reflected; word choice may already be done or may be left to the generation system; segmentation (into clause-sized portions) may be an issue at stake, etc.

*This work was supported by the German Ministry for Research and Technology (FKZ ITW 9002 0).

[1] We exclude the problems of generating spoken output in this paper.

Under these circumstances, a modular and reusable generation system cannot be monolithic. Rather generation front ends should be configurable depending on the generation task at hand and on the respective communicative competence of the application system. The basic idea is that, based on a shared formalism, different combinations of modules can be configured. The application-dependent parts reside in other modules than the generalizable parts, thus allowing for better transportability. To be clear, this should be understood as a *design strategy* that allows various system architectures.

This paper gives some preliminary ideas about configurable generators. After discussing a generic application situation, the idea of configurable system architectures is presented. We then show how both static and dynamic (run-time) configuration can be achieved in the generator of DFKI's COSMA system, which is involved in multi-agent e-mail dialogues about appointment scheduling.

2 Adaptation to different application situations

When a generation front end system is coupled with an application system to make the results of the latter available in NL, the distribution of tasks is important. A generator can legitimally expect an explicit representation of a communicative goal as its input. However, many existing application systems (e.g. databases) are overburdened with this requirement as they do not have a notion of communicative goals. They do, for instance, not know the difference between what they know and what they say. Other application systems are more intelligent in this respect (e.g. some explanation components of expert systems). Obviously an application-dependent adapter is needed that provides the generator with a communicative goal (in the case of a database, this may be the goal of informing the hearer by default).[2]

On the basis of a communicative goal the content of an utterance can be determined. Usually this involves concept selection from a knowledge base containing the domain model. As the generator must refer to such knowledge at several stages of the generation process, another kind of adaptation problem arises. We suggest relying on an Upper Model approach (cf. [Bateman, 1990]) for transportability. It would be the task of the adapter (or a requirement for the application system) to guarantee that the domain knowledge is available in such a way that it can be classified according to an Upper Model.

In general, adaptation is the task of mediating between the application system and the generator. It includes all application-dependent decisions (it is here where the above-mentioned decision about REQUEST should be made), and all application-dependent generation parameters should be set here. Adaptation does, however, not include application-independent aspects of generation. These are left to the transportable generator.

Finally the adapter serves also solving the technical problem of translating between application-specific and generator-specific formal devices.

[2]Note that this does not necessarily suggests a single component; adaptation could as well be realized by providing various components with special extensions.

The adapter tasks are subject to modification or even redesign for every new application. This will allow us to keep the generation system by and large unchanged and will thus serve as an important prerequisite for reusability.

3 Towards configuration in generation systems

In this section we restrict ourselves to the *determination of linguistic form.* We have nothing to say about content determination. We do not specify how close the input representation should be to the surface; we want to be able to deal with input specifications of varying depth.

In addition to the adaptation of input structures it is necessary to accustom the generation system to the tasks corresponding to various application situations. We suggest, as a guideline for the architecture of generation systems, to make the systems *configurable* in the sense that different tasks can be achieved by activating different subsets of components.

We distinguish static and dynamic configuration. The linguistic and communicative demands of the application system determine which generation components are required. For instance, the application system might use discourse knowledge to represent the focus of attention in the generator's input structure. If this is not the case, the generator should activate a focus determination component. The equipment of a generation system in view of such tasks is called *static configuration.*

What amount of flexibility should the generation interface allow for? The generator must apparently assume certain properties of its input. A given input may have these properties; it may have less or even more than required. If there is, for instance, no representation of a communicative goal, the generator may assume a default goal according to the application situation. We leave it to future research to identify the demands placed on an application system.

Properties of a particular input structure can also activate a component, or exclude it from being activated. For instance, a large excerpt of a database is often better readable for the user if it is presented in a tabular form rather than by a complex natural language text. Moreover, the generation of idiomatic or often needed expressions is best achieved by using some form of canned text or templates rather than the full power of linguistic knowledge. The decision about which method should be adopted can only be made at run time. We subsume these decisions under the notion of *dynamic configuration.*

The configuration can be achieved by parameterizing a complex generation system that is equiped with a large set of integrated modules. For static configuration the system developer specifies the parameters. Dynamic configuration is performed by the generation system and based on criteria for the adapted input structure.

Some prerequisites are necessary for a configurable system to work. First of all, the components need to be usable as modules in different generation contexts. Second, all components need an interface to a shared formalism, which allows for the communication

and flow of control between the components. Possible candidates are logical representation languages such as NLL (see [Nerbonne *et al.*, this volume]) or typed feature descriptions, as proposed in e.g. [Krieger *et al.*, 1993]. Finally, a clear distinction between declaratively represented knowledge and processing is necessary, as is convincingly argued in e.g. [Paris and Maier, 1991].

4 Dynamic configuration: The COSMA generator

In one of DISCO's application scenarios, several people try to find a meeting time. They communicate via e-mail, and most use a system called COSMA (Cooperative Schedule Management Agent), which can be authorized to negotiate and in the end commit to a meeting time. A COSMA has a built-in planner that can initiate the request for a meeting and determine with help of an electronic calendar database whether a request can be met. In addition, a COSMA contains a natural language dialogue system for e-mail analysis and generation.[3] From the point of view of the dialogue system, the planner works as an application system. For details on the COSMA architecture see [Neumann, this volume].

The representations at the interface between the planner and the dialogue system specify sender and addressee of the letter as well as a set of actions. Here is an excerpt: **arrange** corresponds to the goal of the sender to arrange a meeting; **confirm** and **reject** represent the expected intentions; **refine** accepts and concretizes some data (e.g. from *tomorrow afternoon* to *14:30 h*); **modify** rejects but suggests an alternative. A sample structure for **arrange** is shown in Figure 1.

Starting from such an expression decisions regarding linguistic form must be made and afterwards realized in natural language. For the determination of form, the first order semantic representation language NLL (see [Nerbonne *et al.*, this volume]) can be used as the basic formalism. Several modules transform NLL expressions until most decisions about the surface form are made. During surface generation an HPSG-type constraint-based grammar of German is used that is encoded by typed feature structures. This grammar combines semantic structures (see [Kasper, this volume]) with syntactic ones (see [Netter, this volume]). Surface generation is based on a Lisp variant of the semantic-head-driven algorithm described in [Shieber *et al.*, 1990].

NLL is conceived as a standard semantics module with a fixed core and various extensions for diverse language understanding systems. NLL offers a formal basis for disambiguation — this cannot be completely done with syntax and compositional semantics — and supports access to diverse application systems by, among other things, allowing to encode and apply domain-specific inferences. This design fits well with the idea of having modules working in different contexts. However, the use of NLL and typed feature

[3]Natural language is required for those participants that still read their e-mail themselves and use their old fashioned, leather bound calendar. Besides the natural language e-mail text, an internal COSMA representation is sent that can be directly processed by COSMA recipients.

```
((from . "schulz@dfki.de")
 (to . ("smith@research.de"))
 (message-id . 4711)
 (actions . ((arrange ((type . :meeting)
                       (topic . "COSMA-Architecture")
                       (participants . ("schulz@dfki.de"
                                        "smith@research.de"
                                        "brown@development.de"))
                       (time . ((this-year)
                                (add (this-week) (week . 8))
                                (day-of-week . :tuesday)
                                (hour . 14)
                                (minute . 30)))
                       (place . "DFKI 1.01"))))))
```

Figure 1: Input Structure for the Generation System

structures involves two distinct formalisms. We are presently investigating whether the representation of NLL expressions as feature structures might ease the communication between the modules.

How is linguistic form determined in COSMA? There is an adapter component for translating expressions provided by the planner (cf. the one in Figure 1) into NLL. The planner has no linguistic knowledge available to it. As a consequence, the adapter must produce suitable speech acts from the actions (e.g., **arrange** is interpreted as a REQUEST). Static configuration accounts for the following tasks to be fulfilled by different interacting modules: access to the domain model, access and maintenance of the dialog model and the discourse memories (including anaphora generation), choice of content words, translation of NLL expressions into feature structures, surface generation with help of the grammar.[4]

Besides the free generation of text, which involves full linguistic processing, the use of prefabricated text is reasonable in certain generation contexts. The exclusive use of canned text has been criticized correctly for being too inflexible. A deliberate integration of canned parts and freely generated parts, however, can be extremely useful expescially in standard situations of the respective application situation. In COSMA, the linguistic form can either be generated freely, as in (1), or by using a tabular form (Figure 2). Dynamic configuration of the generation method depends on the complexity of the underlying NLL expression. A plausible criterion is the number of arguments of **arrange** (in addition to **type**). If there is just one or two, free generation is preferred; if there are more than two, a tabular form is preferred.

[4]At present, static configuration does not allow the treatment of textual relations, but we expect that an appropriate extension can be made without having to redesign the architecture.

```
Date: Tue, 7 Jul 92 16:17:14 +0200
Message-Id: <9207071417.AA03280@sol.dfki.de>
Received: by sol.dfki.de; Tue, 7 Jul 92 16:17:14 +0200
Organization: DFKI Saarbruecken GmbH, D-W 6600 Saarbruecken
From: schulz@dfki.de (Peter Schulz)
To: smith@research.de
Subject: Meeting
Reply-To: schulz@dfki.de

Dear Mr Smith,

I would like to arrange the following meeting:

Time:         September 1, 1992, 14:30 h
Place:        DFKI Saarbruecken, room 1.01
Participants: Mr Brown, Schulz, Smith
Topic:        COSMA Architecture

Sincerely yours,
Peter Schulz
```

Figure 2: Letter in Tabular Form

The system decides to use a table for the input structure in Figure 1. The speech act is verbalized by an introductory sentence that is generated by virtue of a template. A tabular description of the meeting's parameters follows. The message is surrounded by suitable forms of salutation and greeting. This text structure is very simple, and again, this is due to properties of the domain and the planner's communicative competence.

(1) I would like to meet you, Mr Brown and Mr Smith on September 1st, 1992, 14:30 h at DFKI Saarbrücken, room 1.01. We would like to talk about the topic "COSMA Architecture".

If the system decides to produce a table, the introductory sentence must be generated. Since its content depends solely on the action (in this case, **arrange**), the adapter component provides an underspecified NLL representation of the sentences. The generator has available to it a set of templates associating NLL expressions with surface sentences. The NLL representation of the **arrange** request is matched against the templates. If the match doesn't succeed, free generation is chosen as the escape case. Which templates are available should depend on the frequency of their use. Note that templates are generated by using the analysis component of the dialogue system, thus ensuring that template-

```
Date: Tue, 8 Jul 92 10:10:18 +0200
Message-Id: <9207081018.AA03391@mac.research.de>
Received: by mac.research.de; Wed, 8 Jul 92 10:10:18 +0200
Organization: Research Inc., D-W-6000 Frankfurt/M.
From: smith@research.de
To: schulz@dfki.de
Reply-To: smith@research.de

Dear Mr Schulz,

I can't attend the appointment as scheduled since I have to travel
to Bonn for negotiations. Could we postpone the meeting to Sept. 9 ?

Smith
```

Figure 3: Partially Freely Generated Letter

based utterances can be understood by other COSMA systems. An example for a freely generated text as a possible answer to the above letter is shown in Figure 3. It is based on the domain action **modify**. Free generation occurs when non-standard reactions or elaborations are produced. In the present scenario argumentations for rejections are the most typical examples. One task of the generator is to determine sentence borders and what should go into a sentence. The problem is not as difficult as in the general case since the domain often provides good heuristics for what should be said next.

5 Conclusion

The design sketched so far has been implemented to a small extent. Many details of the interactions between the different components are not defined yet. Future plans include the use of canned text and template mechanisms not only as an alternative to but also as part of free generation.

As long as only one application area is investigated, static configuration only serves as a guideline for design; the adaptation of other kinds of input structures is, however, envisaged.

Configuration of generation systems certainly can solve many problems with input specifications of varying depth. In this paper, we have assumed an ideal world in which a superset of a system performance possibly required is available. This is acceptable for more surface-oriented modules, e.g. the grammar or the realization component, where we know fairly well how to extend them in an appropriate fashion. For "deeper" modules such as the domain model and anaphora generation suggestions for modularization are

much more experimental.

This is partly due to a lack of scientific agreement about knowledge modelling for NL processing (cf. e.g. the discussion in [Klose *et al.*, 1992]) and to the fact that only a part of the naturally occurring anaphoric phenomena can be modelled appropriately, given the current state of linguistic knowledge. In practice, only approximations to the ideal picture will be achievable. Fortunately, this will already be beneficial and increase the chances of reusability.

An important question is how much effort it will be to redesign the adapter. The ideas presented here are based on the plausible assumption that the redesign of larger parts of the generator will usually cost more.

Important advantages of configuration that will justify the considerable software-technical effort over the medium-term include:

- Generation front-ends can be adapted more easily to diverse application systems.
- Existing modules can be reused.
- Comparison of generation front-ends is easier.

References

[Bateman, 1990] J. A. Bateman. Upper modeling: organizing knowledge for natural language processing. In *Proceedings of the 5th International Workshop on Language Generation*, Pittsburgh, PA., 1990.

[Klose *et al.*, 1992] Gudrun Klose, Ewald Lang, and Thomas Pirlein, editors. *Ontologie und Axiomatik der Wissensbasis von LILOG.* Springer, Berlin, Heidelberg, 1992. IFB Bd. 307.

[Krieger *et al.*, 1993] Hans-Ulrich Krieger, Ulrich Schäfer, Stephan Diehl, and Karsten Konrad. TDL, A Type Description Language for HPSG. Part 1: Overview. 1993.

[Paris and Maier, 1991] Cecile L. Paris and Elisabeth Maier. Knowledge resources or decisions? In Marie Meteer and Ingrid Zukerman, editors, *Proc. IJCAI-91 Workshop on Decision Making Throughout the Generation Process*, pages 11–17, Sydney, 1991.

[Shieber *et al.*, 1990] Stuart M. Shieber, Gertjan van Noord, Robert C. Moore, and Fernando C. N. Pereira. A semantic-head-driven generation algorithm for unification-based formalisms. *Computational Linguistics*, 16(1):30–42, 1990.

The DISCO DEVELOPMENT SHELL and its Application in the COSMA System

Günter Neumann*

Deutsches Forschungszentrum für Künstliche Intelligenz
Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, Germany
neumann@dfki.uni-sb.de

Abstract

This paper describes the DISCO DEVELOPMENT SHELL, which serves as a basic tool for the integration of natural language components in the DISCO project, and its application in the COSMA system, a Cooperative Schedule Management Agent. Following an *object oriented* architectural model we introduce a *two-step approach*, where in the first phase the architecture is developed independently of specific components to be used and of a particular flow of control. In the second phase the "frame system" is instantiated by the integration of existing components as well as by defining the particular flow of control between these components. Because of the object-oriented paradigm it is easy to augment the frame system, which increases the flexibility of the whole system with respect to new applications. The development of the COSMA system will serve as an example of this claim.

1 Introduction

Today's natural language systems are large software products. They consist of serveral mutually connected components of different kinds, each developed by different researchers often placed on different location. The integration of these components has therefore become a software engineering and mangement problem. We will consider the project DISCO (DIalogue Systems for COoperating agents) from this perspective. DISCO's primary goal is processing of multiagent natural language dialogue. Multiagent capabilities make it an appropriate front end for autonomous cooperative agents, exemplified by the COSMA system described in section 3. The primary task of DISCO is to serve as a kernel linguistic system in order to support distributed cooperative dialogues.

*The research underlying this paper was supported by a research grant, FKZ ITW 9002 0, from the German Bundesministerium für Forschung und Technologie to the DFKI project DISCO.

The use of modern programming techniques in system integration is crucial to support the following desiderata:

- modularity of NLP components
- experimentation with flow of control
- incorporation of new modules
- building of subsystems and standalone applications
- accommodation of alternative modules with similar functionality

The architecture of the DISCO system and COSMA have both been realized using the DISCO DEVELOPMENT SHELL, which we are introducing in the following section. Because of the lack of space we give only a short description of the basic ideas.

2 Overview of the DISCO development shell

In order to perform the tasks mentioned above we have chosen a *two-step approach* to realize DISCO's architecture:

1. In a first step the architecture is described and developed independently of the components to be used and of the particular flow of control. Possible components are viewed as black boxes and the flow of control is described independently of specific components. In such an abstract view the architecture realizes only a 'contentless' schema called the *frame-system.*
2. Next the frame-system has to be 'instantiated' by the integration of existing modules and by defining the particular flow of control between these modules.

It is useful to divide the system components into different types according to their specific tasks. Currently, we distinguish:[1]

- tool components (e.g., graphic devices, printer, debugger, errror handler)
- natural language components (e.g., morphology, parser, generator, speech act recognition)
- control component

In order to obtain a high degree of *flexibility* and *robustness* (especially during the development phase of a system) the control unit directs and monitors the flow of information between the other components. The important tasks of the control unit are:

[1] We do not assume that this list is complete. For example, it is also possible to realize knowledge sources as components of the frame system.

- to direct the data flow between the individual components
- to define which components should run together to realize a 'subsystem'
- to check the data received from one component before they are sent to another one
- to manage global memory and call specific tools

There is a command level for direct communication with the kernel. The purpose of the command level is to provide commands that allow users to run subsystems, to activate or inactivate tracing of modules and to specify printing devices. Users may also specify values for global variables interactively or with configuration files for each module.

Object Oriented Design If a new component must be integrated, one would like to concentrate only on those parts that are of specific interest for these new components. Algorithms or data which are common to all components (or components of a specific type) should be defined only once and then be added automatically for each new component without side–effects to other already integrated components.

We have choosen an *object–oriented programming style* (OOP style) using the Common Lisp Object System (CLOS) in order to realize the two–step approach described above. In the object–oriented paradigm a program is viewed as a set of objects that are manipulated by actions. The state of each object and the actions that manipulate the state are defined once and for all when the object is created. The essential ingredients of object–oriented programming are *objects*, *classes* and *inheritance*. *Objects* are modules that encapsulate data and operations on that data. Every object is an instance of a specific class which determines its structure and behaviour. *Inheritance* allows new classes to specialize already defined classes. The result is a hierarchy of classes where classes inherit the behaviour (data and operations) from superclasses. The advantage for the programmer is that she need only specify to what extend the new class is different from the class(es) it inherits from. This supports the design of modular and robust systems that are easy to use and extend.[2]

CLOS The main programming language for the DISCO project is Common Lisp. Because CLOS is defined to be a *standard language extension* to Common Lisp it is easy to combine 'ordinary' Lisp code with OOP style. CLOS allows us to define an hierarchical organization of classes that models the relationship among the various kinds of objects. Furthermore, because CLOS supports multiple inheritance it is possible to define methods that are defined for particular combinations of classes. Therefore a large amount of control flow is automatically realized by CLOS. This helps us to concentrate on the individual properties of new components, which simplifies and speeds up their integration extremly. Of course, CLOS itself does not enforce modularity or makes it possible to organize programs poorly; it is just a tool that helps us to achieve such modular systems.

[2]The reader should consult e.g., (Keene, 1988) for an excellent introduction into CLOS if more detailed information on object oriented programming is of interest.

DISCO's Class Hierachy The DISCO DEVELOPMENT SHELL consists of the *class hierarchy* and the specification of class specific *methods*. Every type of component and its specializations are defined as CLOS classes. Figure 1 shows a portion of the current hierarchy.

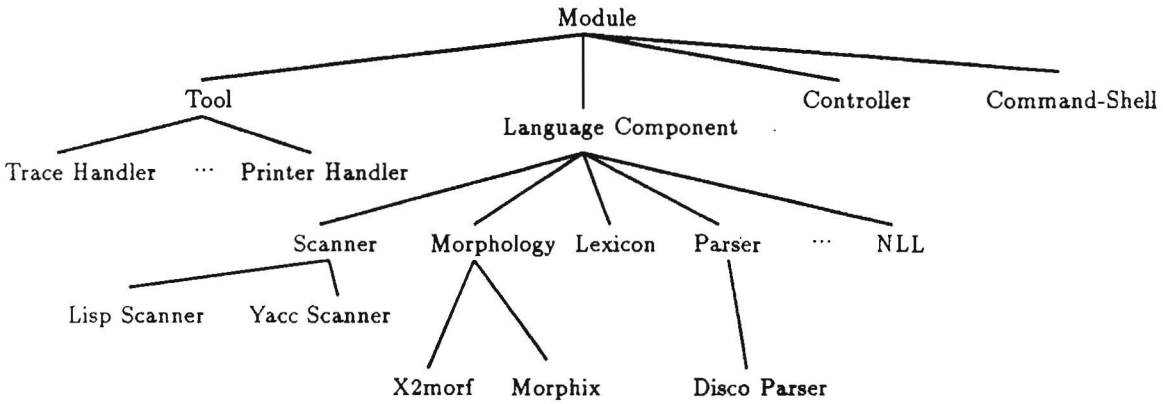


Figure 1: A portion of the current class hierarchie in the DISCO system.

MODULE is the most general class. All other classes inherit its data structure and associated methods. The class LANGUAGE COMPONENT subsumes all modules of the current system which are responsible for language processing. A module that is actually used in the system is an *instance* of one of the classes.

New modules are added to the system by associating a class with them. CLOS supports dynamic extension of the class hierarchie so that new types can be added even at run-time. For example, if we wanted to add a new parser module we would either use the already existing class PARSER or define a new class, say ALTERNATIVE-PARSER. In the first case we assume that we only need the methods that have already been defined for the PARSER class. In the second case we would have to add new methods or could specialize some of the methods that ALTERNATIVE-PARSER inherits from PARSER. In principle it could also happen that the new parser shares many properties with DISCO-PARSER. This would mean that we have to refine the parser subnet in order to avoid redundancy.

Protocols The flow of control between a set of components is mediated by means of *protocols*. Protocols are methods defined for the class *controller*. They specify the set of language components to be used and the input/output relation between the language components. All current protocols are defined using the schema:

```
(call-component controller component-1)
(check-and-transform controller component-1 component-2)
(call-component controller component-2)
(check-and-transform controller component-2 component-3)
(call-component controller component-3)
```

```
.
.
(check-and-transform controller component-(n-1) component-n)
(call-component controller component-n)
```

The controller uses the generic function CALL-COMPONENT to activate an individual language component instance, specialized to the appropriate subclass. Control flow is determined by the sequence of CALL-COMPONENT invocations. Between calls, output is verified and converted to the following component's input format by calling the generic function CHECK-AND-TRANSFORM. This mechanism is very important to support *robustness* especially during the development phase of the system. Specific methods are defined for each module that indicate what to do if a module does not come up with a correct result. These methods are activated by the controller during the call of CHECK-AND-TRANSFORM. In the current version of the system further processing is then interrupted and the user is informed about the problem that occured.

For example, the output of MORPHOLOGY defines the input to PARSER and so on. By calling (CHECK-AND-TRANSFORM CONTROLLER MORPHOLOGY PARSER) the controller checks whether the morphology yielded a valid output and eventually transforms the output for the parser. If MORPHOLOGY detected an unknown word X further processing would be interrupted and the user receives a message notifying him that X is unknown to the morphological component.

Some Remarks If two adjacent components have been proven to work together without problems CHECK-AND-TRANSFORM need not be called for them as it is the case in the following example:

```
(call-component controller component-1)
(check-and-transform controller component-1 component-2)
(call-component controller component-2)
(call-component controller component-3)
.
.
.
(check-and-transform controller component-(n-1) component-n)
(call-component controller component-n)
```

In this example, COMPONENT-2 and COMPONENT-3 interact directly, as shown in Figure 2.

Input and output for the whole system is specifed using general communication channels. In the normal case this is the standard terminal input/output stream of Common Lisp. In the COSMA system an e–mail interface for standard e–mail is used as the principle communication channel. Besides the general input/output device the controller also manages *working* and *long-term memory*. These memories are used to process a sequence of sentences. In this case the controller stores each analysed sentence in long–term memory.

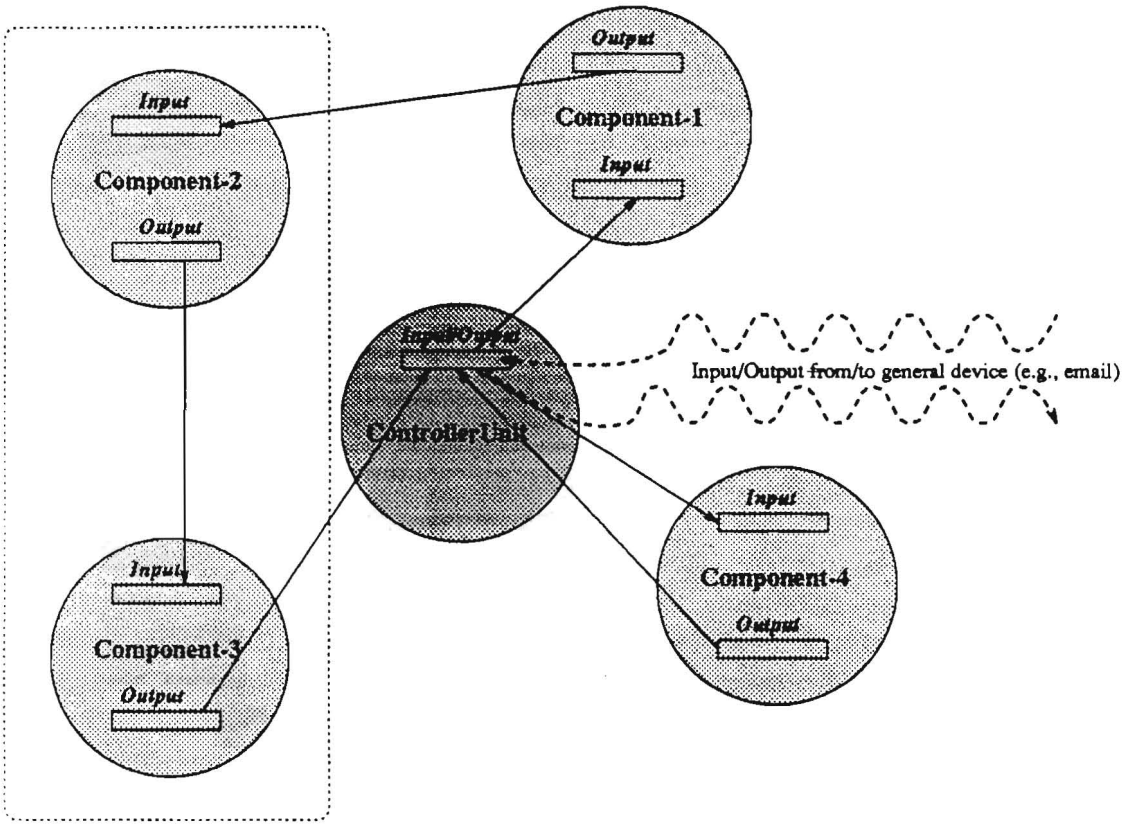


Figure 2: Flow of control between four components. In this protocol component 2 and 3 interact directly. The controller views them as being one component (indicated by the dotted lines around them).

The architecture by itself is not restricted to pipeline processing but would be used in modeling *cascade* or *blackboard* architectures as well. In the latter case the working memory can be used to realize the (possibly structured) blackboard. This has already been partially realized in the current version. In principle, the architecture appeels to be general enough to realize *negotiation*–based architectures.

Status of the DISCO kernel We will give a very brief overview of the current status of the DISCO kernel. We are using an HPSG style of linguistic description (see Netter this volume). The grammar is formalized using the type description language TDL. The basic machinery for linguistic processing is UDINE, a powerful feature structure unifier. The unifier is used in TDL and in almost all linguistic processing units (like parser, generator). The grammar, TDL and UDINE (see Netter this volume for references) constitute the basic linguistic resources. They are not part of the class hierarchy (but accessible from the command level). The basic tool we are using for tracing, debugging and editing feature structures is FEGRAMED, developed by Bernd Kiefer in the DISCO project. FEGRAMED also serves as generic printing device in the kernel machinery as well as in the COSMA system. For grammar debugging it is possible to run serveral subsystems (called standalone

applications), which are activated via the command level. For example, one might want to run the parser and generator without morphology or only the set of components necessary during analysis aso. In each case the same functionality is available as well as the same set of tools. In principle it would be possible for a user to define protocols himself e.g., to test self–written modules because the integration of modules and the definition of protocols takes place in a standardized fashion.

3 Overview of the COSMA system

In this final section we will give a brief overview of the COSMA system. Today, there already exists appointment and resource scheduling tools that allow to display day, week, month, year views or schedule single or repeating events and set beeping, flashing, or pop-up reminders. The principle idea behind the COSMA system is to support scheduling of appointments between several human participants by means of *distributed intelligent calendar assistents*. Instead of using a centralized solution where only one planner mantains a global calendar data–base we have choosen a distributed solution. We assume that each person has its own (therefore local) calendar data–base available on hers computer where each calendar is managed by an individual plannning component. Scheduling of appointments between several participants is viewed as a cooperative negotiation dialogue between the different agents.

It is assumed that electronic mail will be used as a basic means of communication. Information concerning the schedule of particular appointments (e.g., request to arrange a meeting, cancelation of a previously setup appointment or other information relevant in performing some negotiation) is sent around the set of relevant participants via e–mail. Using standard e–mail software has the advantage that scheduling of appointments can be done in a distributed and asychronous way.

Natural language (NL) comes into play because we allow humans to participate who have no calendar assisent access. The only restriction is that they have electronic mail available. Such a (poor) person is responsible for mantaining an old–fashioned calendar but is allowed to use ***natural language*** during appoinment negotiation. Consequently, each COSMA system needs to be able to process natural language, either to understand a NL dialogue contribution or to produce one. To sum up, each COSMA consists of three basic components

- An intelligent assistent that keeps and manages the calendar database
- A graphical user interface to the calendar data–base application planner
- The natural language system DISCO

Each user of a COSMA system has access to the calendar data–base by means of a graphical user interface. The graphical user interface — developed by Stephan Spackman who named the tool DUI — is used to display and update existing items and enter new items into the data–base. The intelligent calendar manager mantains the calendar database.

The current version (developed by the AKA-MOD group) consists of time processing functions, a finite-state protocol for arranging appointments, and an action memory storing the protocol state and original e–mail for each arrangement in progress. The natural language system is used to analyse natural language dialogue contributions, which is the normal case for non–COSMA participants. The natural language system has been developed in the DISCO project, and hence is what we refer as DISCO. The principle task of the DISCO system is to extract that information from an natural language expression that can be used by the calendar manager. Netter (this volume) and Nerbonne et al. (this volume) describe in detail the several knowledge sources and processes that are currently in use to solve this task. DISCO is also responsible for the production of natural language text from the internal representation of scheduling information computed by the calendar manager. The produced text is sent in addition to the internal structure of scheduling expressions to the participants via e–mail. Buseman (this volume) describes the current approach for generating natural language expressions in DISCO in more detail.

Short Example Figure 3 gives an overview of a configuration where three participants, a human (Tick) and two COSMAs (Trick, Track) are involved.

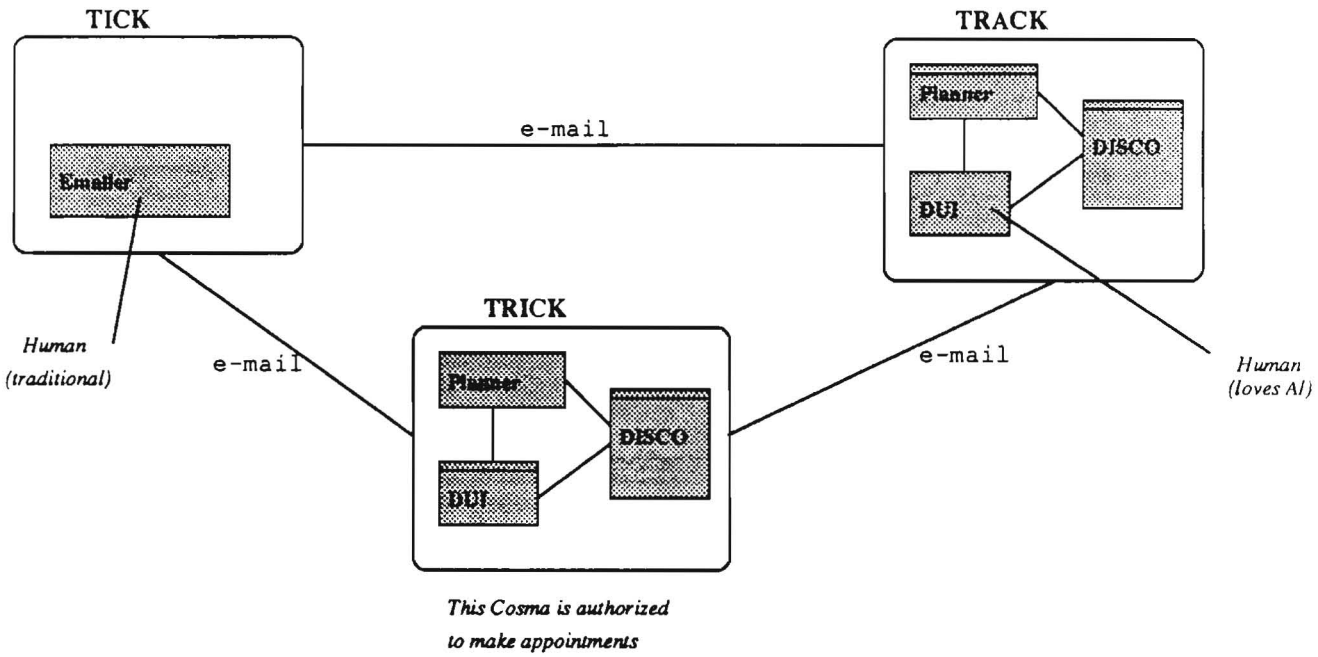


Figure 3: General Overview of the Sample Scenarios

A possible appointment scheduling is as follows (abstracting away from details):

Track to Trick and Tick:
arrange(meeting, 21.10.1992,1p.m.)
Trick to Track:
accept(meeting)
Tick to Track:

Ich bin mit dem Termin einverstanden. (*I accept the appointment*).
Track to Trick and Tick:
confirm(meeting)

In words: Track wanted to arrange a meeting and sends this request to Trick and Tick. Trick automatically sends an acception. Because there are no conflicting entries in his calendar data–base, Tick sends an acception using NL. Track will update its calendar while sending a confirmation to the two participants notifying them that all participants accepted the appointment.

The current version of the system is able to handle more complex dialogs, e.g., appointment scheduling initiated by a non-COSMA user, cancellation and modification of already set up appointments.

To be able to integrate the DISCO system into this domain the following modules have been integrated or modified:

- Exchange of a Lisp–based scanner for a more powerful one implemented using the Unix tools YACC and LEX.
- Integration of a surface–based speech act system (SAR which has been placed between the parser and NLL, cf. (Hinkelman and Spackman, 1992)).
- Application interface for mapping NLl expressions into an internal representation language for the planner (see Nerbonne et al. (this volume)).
- exchange of the semantic head–driven generator with a 'canned text' generator
- A set of communication interfaces to e–mail, the graphical user interface and the planner

4 Conclusion

The DISCO DEVELOPMENT SHELL has been proven very useful in setting up the COSMA system. It was possible to intergrate the new modules independently from the rest of the system. Existing modules have been exchanged by new ones without the need of adding new methods. Because different researchers were able to run subsystems the development of the whole system could be done in a distributed way. Therefore eight very different modules have been intergrated in less than three weeks including test phases.

Based on these experiences we believe that the oject–oriented architectual model of our approach is a fruitful basis for managing large–scale projects. It makes it possible to develop the basis of a whole system in parallel to the development of the individual components. Therefore it is possible to take into account restrictions and modifications of each component as early as possible.

References

Elizabeth A. Hinkelman and Stephen P. Spackman. Abductive speech act recognition, corporate agents and the cosma system. In W. J. Black, G. Sabah, and T. J. Wachtel, editors, *Abduction, Beliefs and Context: Proceedings of the second ESPRIT PLUS workshop in computational pragmatics*, 1992.

Sonya E. Keene. *Object-Oriented Programming in Common Lisp. A Programmer's Guide to CLOS.* Addision-Wesley, 1988.

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
D-6750 Kaiserslautern
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse bezogen werden.
Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far can be ordered from the above address.
The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-92-14
Intelligent User Support in Graphical User Interfaces:
1. InCome: A System to Navigate through Interactions and Plans
   Thomas Fehrle, Markus A. Thies
2. Plan-Based Graphical Help in Object-Oriented User Interfaces
   Markus A. Thies, Frank Berger

22 pages

RR-92-15
Winfried Graf: Constraint-Based Graphical Layout of Multimodal Presentations
23 pages

RR-92-16
Jochen Heinsohn, Daniel Kudenko, Berhard Nebel, Hans-Jürgen Profitlich: An Empirical Analysis of Terminological Representation Systems
38 pages

RR-92-17
Hassan Aït-Kaci, Andreas Podelski, Gert Smolka: A Feature-based Constraint System for Logic Programming with Entailment
23 pages

RR-92-18
John Nerbonne: Constraint-Based Semantics
21 pages

RR-92-19
Ralf Legleitner, Ansgar Bernardi, Christoph Klauck: PIM: Planning In Manufacturing using Skeletal Plans and Features
17 pages

RR-92-20
John Nerbonne: Representing Grammar, Meaning and Knowledge
18 pages

RR-92-21
Jörg-Peter Mohren, Jürgen Müller
Representing Spatial Relations (Part II) -The Geometrical Approach
25 pages

RR-92-22
Jörg Würtz: Unifying Cycles
24 pages

RR-92-23
Gert Smolka, Ralf Treinen:
Records for Logic Programming
38 pages

RR-92-24
Gabriele Schmidt: Knowledge Acquisition from Text in a Complex Domain
20 pages

RR-92-25
Franz Schmalhofer, Ralf Bergmann, Otto Kühn, Gabriele Schmidt: Using integrated knowledge acquisition to prepare sophisticated expert plans for their re-use in novel situations
12 pages

RR-92-26
Franz Schmalhofer, Thomas Reinartz, Bidjan Tschaitschian: Intelligent documentation as a catalyst for developing cooperative knowledge-based systems
16 pages

RR-92-27
Franz Schmalhofer, Jörg Thoben: The model-based construction of a case-oriented expert system
18 pages

RR-92-29
Zhaohui Wu, Ansgar Bernardi, Christoph Klauck: Skeletel Plans Reuse: A Restricted Conceptual Graph Classification Approach
13 pages

RR-92-30
Rolf Backofen, Gert Smolka
A Complete and Recursive Feature Theory
32 pages

RR-92-31
Wolfgang Wahlster
Automatic Design of Multimodal Presentations
17 pages

RR-92-33
Franz Baader: Unification Theory
22 pages

RR-92-34
Philipp Hanschke: Terminological Reasoning and Partial Inductive Definitions
23 pages

RR-92-35
Manfred Meyer:
Using Hierarchical Constraint Satisfaction for Lathe-Tool Selection in a CIM Environment
18 pages

RR-92-36
Franz Baader, Philipp Hanschke:
Extensions of Concept Languages for a Mechanical Engineering Application
15 pages

RR-92-37
Philipp Hanschke: Specifying Role Interaction in Concept Languages
26 pages

RR-92-38
Philipp Hanschke, Manfred Meyer:
An Alternative to Θ-Subsumption Based on Terminological Reasoning
9 pages

RR-92-40
Philipp Hanschke, Knut Hinkelmann: Combining Terminological and Rule-based Reasoning for Abstraction Processes
17 pages

RR-92-41
Andreas Lux: A Multi-Agent Approach towards Group Scheduling
32 pages

RR-92-42
John Nerbonne:
A Feature-Based Syntax/Semantics Interface
19 pages

RR-92-43
Christoph Klauck, Jakob Mauss: A Heuristic driven Parser for Attributed Node Labeled Graph Grammars and its Application to Feature Recognition in CIM
17 pages

RR-92-44
Thomas Rist, Elisabeth André: Incorporating Graphics Design and Realization into the Multimodal Presentation System WIP
15 pages

RR-92-45
Elisabeth André, Thomas Rist: The Design of Illustrated Documents as a Planning Task
21 pages

RR-92-46
Elisabeth André, Wolfgang Finkler, Winfried Graf, Thomas Rist, Anne Schauder, Wolfgang Wahlster: WIP: The Automatic Synthesis of Multimodal Presentations
19 pages

RR-92-47
Frank Bomarius: A Multi-Agent Approach towards Modeling Urban Traffic Scenarios
24 pages

RR-92-48
Bernhard Nebel, Jana Koehler:
Plan Modifications versus Plan Generation: A Complexity-Theoretic Perspective
15 pages

RR-92-49
Christoph Klauck, Ralf Legleitner, Ansgar Bernardi:
Heuristic Classification for Automated CAPP
15 pages

RR-92-50
Stephan Busemann:
Generierung natürlicher Sprache
61 Seiten

RR-92-51
Hans-Jürgen Bürckert, Werner Nutt:
On Abduction and Answer Generation through Constrained Resolution
20 pages

RR-92-52
Mathias Bauer, Susanne Biundo, Dietmar Dengler, Jana Koehler, Gabriele Paul: PHI - A Logic-Based Tool for Intelligent Help Systems
14 pages

RR-92-54
Harold Boley: A Direkt Semantic Characterization of RELFUN
30 pages

RR-92-55
John Nerbonne, Joachim Laubsch, Abdel Kader Diagne, Stephan Oepen: Natural Language Semantics and Compiler Technology
17 pages

RR-92-56
Armin Laux: Integrating a Modal Logic of Knowledge into Terminological Logics
34 pages

RR-92-58
Franz Baader, Bernhard Hollunder:
How to Prefer More Specific Defaults in Terminological Default Logic
31 pages

RR-92-59
Karl Schlechta and David Makinson: On Principles and Problems of Defeasible Inheritance
13 pages

RR-92-60
Karl Schlechta: Defaults, Preorder Semantics and Circumscription
19 pages

RR-93-02
Wolfgang Wahlster, Elisabeth André, Wolfgang Finkler, Hans-Jürgen Profitlich, Thomas Rist:
Plan-based Integration of Natural Language and Graphics Generation
50 pages

RR-93-03
Franz Baader, Berhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, Enrico Franconi:
An Empirical Analysis of Optimization Techniques for Terminological Representation Systems
28 pages

RR-93-04
Christoph Klauck, Johannes Schwagereit:
GGD: Graph Grammar Developer for features in CAD/CAM
13 pages

RR-93-05
Franz Baader, Klaus Schulz: Combination Techniques and Decision Problems for Disunification
29 pages

RR-93-08
Harold Boley, Philipp Hanschke, Knut Hinkelmann, Manfred Meyer: COLAB: A Hybrid Knowledge Representation and Compilation Laboratory
64 pages

RR-93-09
Philipp Hanschke, Jörg Würtz:
Satisfiability of the Smallest Binary Program
8 Seiten

DFKI Technical Memos

TM-91-12
Klaus Becker, Christoph Klauck, Johannes Schwagereit: FEAT-PATR: Eine Erweiterung des D-PATR zur Feature-Erkennung in CAD/CAM
33 Seiten

TM-91-13
Knut Hinkelmann: Forward Logic Evaluation: Developing a Compiler from a Partially Evaluated Meta Interpreter
16 pages

TM-91-14
Rainer Bleisinger, Rainer Hoch, Andreas Dengel:
ODA-based modeling for document analysis
14 pages

TM-91-15
Stefan Busemann: Prototypical Concept Formation
An Alternative Approach to Knowledge Representation
28 pages

TM-92-01
Lijuan Zhang: Entwurf und Implementierung eines Compilers zur Transformation von Werkstückrepräsentationen
34 Seiten

TM-92-02
Achim Schupeta: Organizing Communication and Introspection in a Multi-Agent Blocksworld
32 pages

TM-92-03
Mona Singh:
A Cognitiv Analysis of Event Structure
21 pages

TM-92-04
Jürgen Müller, Jörg Müller, Markus Pischel, Ralf Scheidhauer:
On the Representation of Temporal Knowledge
61 pages

TM-92-05
Franz Schmalhofer, Christoph Globig, Jörg Thoben:
The refitting of plans by a human expert
10 pages

TM-92-06
Otto Kühn, Franz Schmalhofer: Hierarchical skeletal plan refinement: Task- and inference structures
14 pages

TM-92-08
Anne Kilger: Realization of Tree Adjoining Grammars with Unification
27 pages

DFKI Documents

D-92-08
Jochen Heinsohn, Bernhard Hollunder (Eds.):
DFKI Workshop on Taxonomic Reasoning
Proceedings
56 pages

D-92-09
Gernod P. Laufkötter: Implementierungsmöglichkeiten der integrativen Wissensakquisitionsmethode des ARC-TEC-Projektes
86 Seiten

D-92-10
Jakob Mauss: Ein heuristisch gesteuerter Chart-Parser für attributierte Graph-Grammatiken
87 Seiten

D-92-11
Kerstin Becker: Möglichkeiten der Wissensmodellierung für technische Diagnose-Expertensysteme
92 Seiten

D-92-12
Otto Kühn, Franz Schmalhofer, Gabriele Schmidt: Integrated Knowledge Acquisition for Lathe Production Planning: a Picture Gallery (Integrierte Wissensakquisition zur Fertigungsplanung für Drehteile: eine Bildergalerie)
27 pages

D-92-13
Holger Peine: An Investigation of the Applicability of Terminological Reasoning to Application-Independent Software-Analysis
55 pages

D-92-14
Johannes Schwagereit: Integration von Graph-Grammatiken und Taxonomien zur Repräsentation von Features in CIM
98 Seiten

D-92-15
DFKI Wissenschaftlich-Technischer Jahresbericht 1991
130 Seiten

D-92-16
Judith Engelkamp (Hrsg.): Verzeichnis von Softwarekomponenten für natürlichsprachliche Systeme
189 Seiten

D-92-17
Elisabeth André, Robin Cohen, Winfried Graf, Bob Kass, Cécile Paris, Wolfgang Wahlster (Eds.): UM92: Third International Workshop on User Modeling, Proceedings
254 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-92-18
Klaus Becker: Verfahren der automatisierten Diagnose technischer Systeme
109 Seiten

D-92-19
Stefan Dittrich, Rainer Hoch: Automatische, Deskriptor-basierte Unterstützung der Dokumentanalyse zur Fokussierung und Klassifizierung von Geschäftsbriefen
107 Seiten

D-92-21
Anne Schauder: Incremental Syntactic Generation of Natural Language with Tree Adjoining Grammars
57 pages

D-92-22
Werner Stein: Indexing Principles for Relational Languages Applied to PROLOG Code Generation
80 pages

D-92-23
Michael Herfert: Parsen und Generieren der Prolog-artigen Syntax von RELFUN
51 Seiten

D-92-24
Jürgen Müller, Donald Steiner (Hrsg.): Kooperierende Agenten
78 Seiten

D-92-25
Martin Buchheit: Klassische Kommunikations- und Koordinationsmodelle
31 Seiten

D-92-26
Enno Tolzmann: Realisierung eines Werkzeugauswahlmoduls mit Hilfe des Constraint-Systems CONTAX
28 Seiten

D-92-27
Martin Harm, Knut Hinkelmann, Thomas Labisch: Integrating Top-down and Bottom-up Reasoning in COLAB
40 pages

D-92-28
Klaus-Peter Gores, Rainer Bleisinger: Ein Modell zur Repräsentation von Nachrichtentypen
56 Seiten

D-93-01
Philipp Hanschke, Thom Frühwirth: Terminological Reasoning with Constraint Handling Rules
12 pages

D-93-02
Gabriele Schmidt, Frank Peters, Gernod Laufkötter: User Manual of COKAM+
23 pages

D-93-03
Stephan Busemann, Karin Harbusch(Eds.): DFKI Workshop on Natural Language Systems: Reusability and Modularity - Proceedings
74 pages

DFKI Workshop on Natural Language Systems Proceedings

Stephan Busemann, Karin Harbusch (Eds.)

D-93-03
Document