Abstract of a Poster for ICFG12

# A Modular System for Generating Linguistic Expressions from Underlying Clause Structures

Fabian Steeg, Christoph Benden, Paul-Otto Samuelsdorff
Department of Linguistics, University of Cologne

December, 14 2005

**Abstract**

This paper describes a modular system for generating linguistic expressions from underlying clause structures (UCS), implemented in Java and Prolog. The system uses a UCS representation based on Dik & Hengeveld (1997), the expression component is based on a revised version of the implementation described in Samuelsdorff (1989). The system can be used to evaluate and improve the theory of Functional Grammar (FG) with respect to theoretical and representational issues in language generation. By means of its modular architecture it could act as the language generation component in a larger FG-based NLP system.

## Contents

## List of Figures

# 1 Motivation and Overview

This paper describes a modular implementation of a system for generating linguistic expressions applying the theory of Functional Grammar (FG). The idea of creating a computational implementation of FG mechanisms, to "build a model of the natural language user" (Dik & Hengeveld 1997:1) is central to the theory of FG and a valuable evaluation tool for linguistic theories in general, since "linguistics may learn from being applied" (Bakker 1994:4). Therefore our implementation could be used to evaluate and improve the theory of FG with respect to theoretical issues in language generation.

The system consists of individual, exchangeable modules for creating an underlying clause structure (UCS), processing that input and generating a linguistic expression from the input UCS. By means of its modular architecture our program could act as the language generation component in a larger FG-based NLP system.

The UCS format is based on the representation of underlying structures given in Dik & Hengeveld (1997), our implementation could therefore be used to evaluate and improve representational aspects of the theory of FG. The expression rules and the lexicon are based on a revised and extended version of the implementation described in Samuelsdorff (1989).

# 2 System Architecture

## 2.1 System Architecture Overview

The system consists of three modules. The graphical user interface (GUI) assists the user in entering a valid UCS (*input module*). When a valid UCS is entered, the input is converted into a representation of the UCS in Prolog (*processing module*). This representation is then used by the *grammar module* to generate the linguistic expression. The result is passed back[1] by the *processing module* to the *input module* to display the linguistic expression generated from the original UCS entered by the user. For an overview of the described interaction of the system modules see figure 1.

Such a modular approach has two main advantages: First, modules can be exchanged, for instance the *input module* could be a web-based user interface and the actual processing could happen on a server. Second, by using a defined input UCS format, our system could be combined with other FG-based NLP components which would have to generate the input UCS for the *processing module*.

The reason for using Java for the user interface and processing of the UCS on the one hand and Prolog for the expression rules and the lexicon on the other hand stems from

[1]For calling Prolog from Java we use Interprolog (http://www.declarativa.com/interprolog/). The Prolog implementation we use is SWI-Prolog (http://www.swi-prolog.org/)

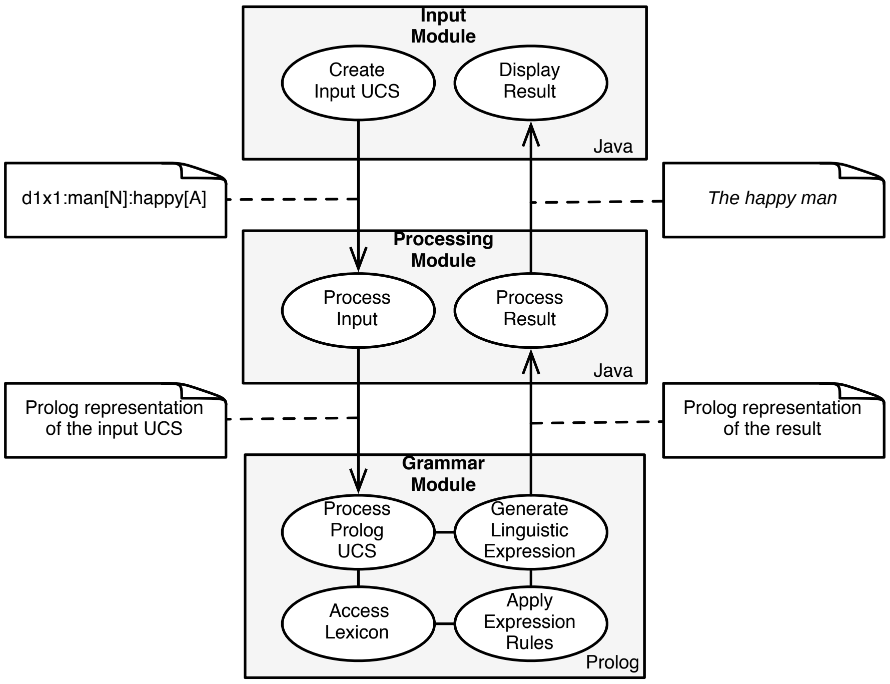

Figure 1: System Architecture

the idea of using implementation languages well suited for a particular task.[2] Java is a widespread multi-purpose programming language with abundant supply of libraries, while Prolog offers convenient notation and processing mechanisms,[3] is familiar to many linguists and has a particular strong standing as an implementation language for FG (e.g. Dik 1992, Samuelsdorff 1989).

## 2.2 Input Module

The *input module* provides a graphical user interface (GUI) to our system. Here a UCS is created, edited and evaluated. Upon evaluation the UCS is sent to the *processing module*, which communicates with the *grammar module*. When the generation is done, the GUI displays either the result of the evaluation, namely the linguistic expression generated from the UCS, or an error message.

[2]An example of combining Java and Prolog in a different NLP system in a similar fashion is described in Macks (2002).

[3]For instance, lexical entries can be stored directly as Prolog facts, e.g. in the following format: verb(believe,state,[regular, regular],[[experiencer,human,X1],[goal,proposition,X2]],Satellites).

## 2.3 Processing Module

The *processing module's* input format is a representation of the linguistic expression to be generated; its form is based on the representation of underlying structures given in Dik & Hengeveld (1997). Figure 2 shows a sample input UCS representing the linguistic expression *John is the man who was given the book by Mary.* Indentation is used to visualize the structure of the UCS, terms are separated by commas.

```
(pres e1:
  (d1x1:man[N]:
    (past e2:give[V]
      (d1x2:mary[N])Ag,
      (d1x3:book[N])Go,
      (x1)RecSubj
    )
  ),
  (d1x4:john[N])0
)
```

Figure 2: Sample input UCS representing the linguistic expression *John is the man who was given the book by Mary*

The *processing module* parses the input UCS entered by the user (or potentially coming from a different source) and creates an internal representation which is then converted into the output format of the *processing module*, a Prolog representation of the input UCS.

## 2.4 Grammar Module

In the *grammar module* the Prolog representation of the UCS generated by the *processing module* is used to generate a linguistic expression. The expression rules and the lexicon are implemented in Prolog, based on a revised and extended version of the implementation described in Samuelsdorff (1989).

In the original implementation the underlying structure is subsequently built via a user dialog, during which the expression to be generated is specified. To make the implementation work as a module in the described system, this user dialog is replaced by an immediate processing of the entire UCS representing the linguistic expression to be generated. The user dialog is therefore functionally replaced by the input UCS, which is created in the *input module* and converted into a Prolog representation by the *processing module.*

## 3 Conclusion

We described a modular implementation of a language generation system based on the theory of FG. It makes use of a UCS format based on Dik & Hengeveld (1997) and consists of modules implemented in Java and Prolog. The system can be used to evaluate and improve the FG grammar model with respect to theoretical and representational issues in language generation. By means of its modular architecture it could act as the language generation component in a larger FG-based NLP system.

## References

Bakker, Dik. 1994 "Formal and Computational Aspects of Functional Grammar and Language Typology". Amsterdam: IFOTT.

Dik, Simon C. 1992 "Functional Grammar in Prolog; an integrated implementation for English, French and Dutch". Berlin, New York: Mouton de Gruyter.

Dik, Simon C. & Kees Hengeveld. 1997 "The Theory of Functional Grammar, Part 1: The Structure of the Clause". Berlin, New York: Mouton de Gruyter.

Macks, Aaron. 2002 "Parsing Akkadian Verbs with Prolog". 27. November 2005 <www.cs.um.edu.mt/~mros/WSL/papers/macks.pdf >

Samuelsdorff, Paul-Otto. 1989 "Simulation of a Functional Grammar in Prolog" In: Connolly, John H. & Simon C. Dik (eds.) *Functional Grammar and the Computer.* 29-44. Utrecht, Providence: Foris Publications.