A modular system for generating linguistic expressions from underlying clause structures

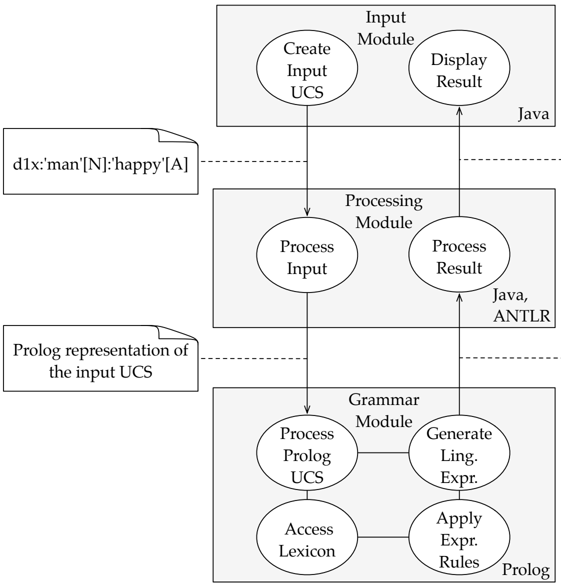
Fabian Steeg[1], Christoph Benden[1] & Paul O. Samuelsdorff[2]
Department of Linguistics[3], University of Cologne, Germany

[1]Sprachliche Informationsverarbeitung, {fsteeg | cbenden}@spinfo.uni-koeln.de, http://www.spinfo.uni-koeln.de
[2]Allgemeine Sprachwissenschaft, paul-o.samuelsdorff@uni-koeln.de, http://www.uni-koeln.de/phil-fak/ifl/asw [3]http://www.uni-koeln.de/ifl

1. Motivation

The idea of creating a computational implementation of Functional Grammar (FG) mechanisms, to "build a model of the natural language user" (Dik 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 system could be used to evaluate and improve the theory of FG with respect to theoretical issues in language generation. The system uses an underlying clause structure (UCS) representation based on Dik (1997) and can therefore be used to experiment with representational issues of FG. The expression component is based on a revised version of the implementation described in Samuelsdorff (1989).

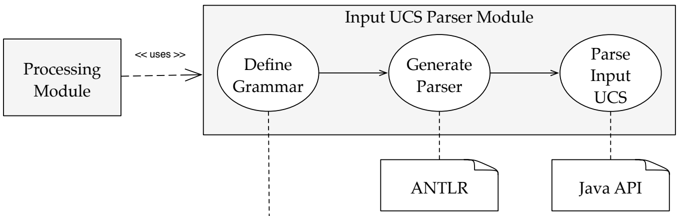
System Architecture



2. Well-defined, FG-conforming notation

In the original implementation the underlying structure is built up step by step via a user dialog, during which the expression to be generated is specified (see Samuelsdorff 1989:38ff.). To make the implementation work as a module in the described system, this user dialog is replaced by an immediate processing of the entire underlying clause structure (UCS) representing the linguistic expression to be generated. The user dialog is therefore replaced by the input UCS, which is created in the input module and converted into a Prolog representation by the processing module.

Parser Overview




```
// ANTLR GRAMMAR DEFINITION

[...]

// "d1x:" or "Past Pf e:"
((d:DEF (n:NUMBER)?) | ((tense:TENSE)? (aspect:ASPECT)?))? t:LAYER RESTRIKTOR
// man[N]
(w0:WORD p0:WORD_CLASS (RESTRIKTOR)? )?

[...]

LAYER
      : 'f'      // predicate
      | 'x'      // term
      | 'e'      // predication
      | 'E'      // clause
      | 'X'      // proposition
      ;
NUMBER
      : '1'      // sing
      | '2'      // dual
      | 'm'      // plural 'many'
      ;
[...]
```

```
(Past e:
    (d1x:'man'[N]:
        (Past Pf e:'give'[V]
            (d1x:'mary'[N])Ag
            (dmx:'book'[N]:'old'[A])Go
            (x:'man'[N])RecSubj
        )
    )
    (d1x:'john'[N])0
)
```

An example for a nested UCS that is parsable by the ANTLR-generated parser (would correspondent to the expression *John was the man that had given the old book to Mary).*

Java API

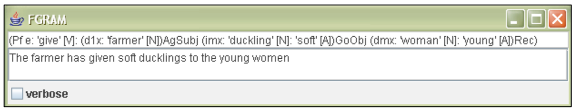
```
// Parser usage

String ucs = "(e:'love'[V]: (x: 'man' [N])AgSubj (dmx: 'woman' [N])GoObj)";
UcsParser parser = new UcsParser(new UcsLexer(new StringReader(ucs)));
// Parse the expression, the Predicate contains the entire UCS
Predicate p = parser.input();

// Full usage

InputProcessor processor = new InputProcessor(configFileLocation);
String ucs = "(e: 'love' [V]: (x: 'man' [N])AgSubj (dmx: 'woman' [N])GoObj)";
// Generate the expression: The man loves the woman
String expression = processor.process(ucs, true);
```

Prolog Lexicon

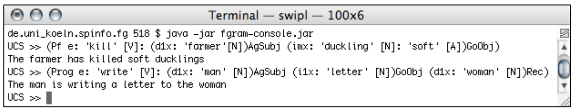
```
% LEXICON
verb(believe,state,[regular, regular],[[experiencer,human,X1],[goal,proposition,X2]],Satellites).
verb(give,action,[gave,given],[[agent,animate,X1],[goal,any,X2],[recipient,animate,X3]],Satellites).
[...]

noun(axe,instrument,[regular,neuter],[[argument,instrument,X]],Sat).
noun(book,readable,[regular,neuter],[[argument,readable,X]],Sat).
[...]

adj(big,size,[[],big],[[argument,any,X]],Sat).
adj(eager,quality,[[],eager],[[first_argument,animate,X1],[second_argument,infinitive,X2]],Sat).
[...]

% GRAMMATICON
be([[was,past,sing], [were,past,plural], [is,present,sing],[are,present,plural]]).
have([[had,past,N],[has,present,sing],[have,present,plural]]).
do([[did,past,N],[does,present,sing],[do,present,plural]]).
determiner([["the",def,N,G],["a",indef,sing,G],["every",total,sing,G],[...]
pronouns([[he,pers,masc,sing,subj],[him,pers,masc,sing,ob],[she,pers,fem,sing,subj],[...]
```

Different Implementations of the Input Module

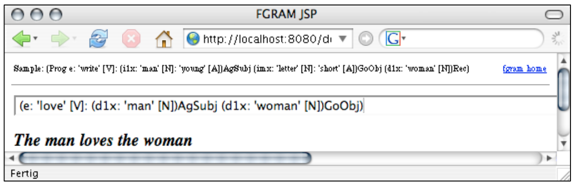
These experimental implementations all use Java. In general any implementation in any language could act as the Input Module, if it fomulates (or assists in formulating, as here) the input UCS for the Processing Module.



Graphical user interface using Java Swing (here on Windows XP)



Java console application (here on Mac OS X)



Web-based interface using Java Server Pages (here in Firefox on Mac OS X)

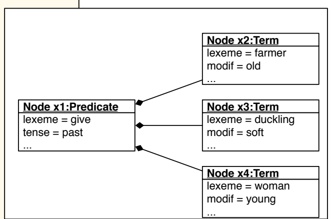
3. Modularity

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. The system architecture can therefore be characterized as a Model-View-Controller (MVC) or three-tier architecture. Such a modular approach has two main advantages: First, modules can be exchanged, for instance the input module can be a web-based user interface and the actual processing can happen on a server. Second, by using a defined input UCS format, our system could be combined with other FG-based natural language processing (NLP) components which could formulate the input UCS for our system or use the Java or the Prolog representation of the parsed input UCS.

Prolog UCS

```
% PROLOG REPRESENTATION OF THE INPUT UCS

node(x1, 0).
node(x2, 1).
node(x3, 1).
node(x4, 1).

prop(clause, illocution, decl).
prop(clause, type, mainclause).

prop(x1, type, pred).
prop(x1, tense, past).
prop(x1, perfective, true).
prop(x1, progressive, false).
prop(x1, mode, ind).
prop(x1, voice, active).
prop(x1, subnodes, [x2, x3, x4]).
prop(x1, lex, 'give').
prop(x1, nav, [V]).
prop(x1, det, def).

prop(x2, type, term).
prop(x2, role, agent).
prop(x2, relation, subject).
prop(x2, proper, false).
prop(x2, pragmatic, null).
prop(x2, num, plural).
prop(x2, modifs, [old]).
prop(x2, lex, 'farmer').
prop(x2, nav, [N]).
prop(x2, det, def).

prop(x3, type, term).
prop(x3, role, goal).
prop(x3, relation, object).
prop(x3, proper, false).
prop(x3, pragmatic, null).
prop(x3, num, plural).
prop(x3, modifs, [soft]).
prop(x3, lex, 'duckling').
prop(x3, nav, [N]).
prop(x3, det, indef).

prop(x4, type, term).
prop(x4, role, recipient).
prop(x4, relation, restarg).
prop(x4, proper, false).
prop(x4, pragmatic, null).
prop(x4, num, plural).
prop(x4, modifs, [young]).
prop(x4, lex, 'woman').
prop(x4, nav, [N]).
prop(x4, det, def).
```

```
(Past Pf e:'give'[V]:
    (dmx:'farmer'[N]:'old'[A])AgSubj
    (imx:'duckling'[N]:'soft'[A])GoObj
    (dmx:'woman'[N]:'young'[A])Rec
)
```



The old farmers had given soft ducklings to the young women

4. Programming Languages

The System uses Java, Prolog and the ANTLR Grammar description language. The reason for using Java for the user interface and processing of the underlying clause structure (UCS), ANTLR for the Grammar definition and Prolog for the expression rules and the lexicon stems from the idea of using implementation languages well suited for a particular task. Java is a widespread multi-purpose programming language with abundant supply of libraries, ANTLR a specialized grammar description language and parser generator and Prolog offers convenient notation and processing mechanisms, is familiar to many linguists and has a particular strong standing as an implementation language for FG (e.g. Samuelsdorff 1989, Dik 1992).

Participation

Infrastructure for participation (in particular a Subversion repository, a website and a forum) is available at Sourceforge:

http://fgram.sourceforge.net

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. 1997. The Theory of Functional Grammar, Part 1: The Structure of the Clause. Kees Hengeveld (ed.) 2nd rev. ed. Berlin, New York: Mouton de Gruyter.

Samuelsdorff, Paul O. 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.