A Fast and Portable Realizer for Text Generation Systems

Benoit Lavoie and **Owen Rambow**
CoGenTex, Inc.
840 Hanshaw Road, Ithaca, NY 14850, USA
benoit, owen@cogentex.com

1 Introduction

Systems that generate natural language output as part of their interaction with a user have become a major area of research and development. Typically, natural language generation is divided into several phases, namely *text planning* (determining output content and structure), *sentence planning* (determining abstract target language resources to express content, such as lexical items and syntactic constructions), and *realization* (producing the final text string) (Reiter, 1994). While text and sentence planning may sometimes be combined, a *realizer* is almost always included as a distinct module. It is in the realizer that knowledge about the target language resides (syntax, morphology, idiosyncratic properties of lexical items). Realization is fairly well understood both from a linguistic and from a computational point of view, and therefore most projects that use text generation do not include the realizer in the scope of their research. Instead, such projects use an off-the-shelf realizer, among which PENMAN (Bateman, 1996) and SURGE/FUF (Elhadad and Robin, 1996) are probably the most popular. In this technical note and demo we present a new off-the-shelf realizer, REALPRO. REALPRO is derived from previous systems (Iordanskaja et al., 1988; Iordanskaja et al., 1992; Rambow and Korelsky, 1992), but represents a new design and a completely new implementation. REALPRO has the following characteristics, which we believe are unique in this combination:

- REALPRO is implemented in C++. It is therefore both fast and portable cross-platform.
- REALPRO can be run as a standalone server, and has C++ and Java APIs.
- The input to REALPRO is based on syntactic dependency (roughly, predicate-argument and predicate-modifier structure).
- Syntactic and lexical knowledge about the target language is expressed in ASCII files which are interpreted at run-time. It can easily be updated.

We reserve a more detailed comparison with PENMAN and FUF, as well as with AlethGen/GL (Coch, 1996) (which is perhaps the system most similar to REALPRO, since they are based on the same linguistic theory and are both implemented with speed in mind), for a more extensive paper. This technical note presents REALPRO, concentrating on its structure, its coverage, its interfaces, and its performance.

2 Input Structure

The input to REALPRO is a syntactic dependency structure. It is called the Deep-Syntactic Structure or "DSyntS" for short, and is inspired in this form by I. Mel'čuk's Meaning-Text Theory (Mel'čuk, 1988). This representation has the following salient features:

- The DSyntS is an unordered *tree* with labeled nodes and labeled arcs.
- The DSyntS is *lexicalized*, meaning that the nodes are labeled with lexemes (uninflected words) from the target language.
- The DSyntS is a *dependency* structure and not a *phrase-structure* structure: there are no nonterminal nodes, and all nodes are labeled with lexemes.
- The DSyntS is a *syntactic* representation, meaning that the arcs of the tree are labeled with syntactic relations such as "subject" (represented in DSyntSs as I), rather than conceptual or semantic relations such as "agent".
- The DSyntS is a *deep* syntactic representation, meaning that only meaning-bearing lexemes are represented, and not function words.

First, consider the simple example in Figure 1, which corresponds to the sentence (1):

(1) This boy sees Mary.

Lexemes which are in the lexicon are in uppercase, those that are not are in lowercase. For lexemes not in the lexicon it is necessary to specify the word class

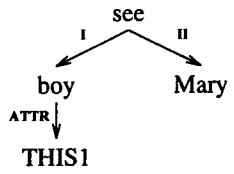

Figure 1: Input structure for sentence (1)

as a feature, e.g. word-class:verb. For readability, we omit these features in the tree diagrams. Subject and object are indicated by the arc labels I and II, respectively, and modification is represented by the arc label ATTR. If we add feature question:+ to the verb and feature number:pl to the node for *boy*, then we get (2):

(2) Do these boys see Mary?

This illustrates that function words (*do*) need not be included in the input DSyntS, and that syntactic issues such as subject-verb and noun-determiner agreement are handled automatically. The tree in Figure 2 yields (3):

(3) Mary winning this competition means she can study in Paris and can live with her aunt, whom she adores.

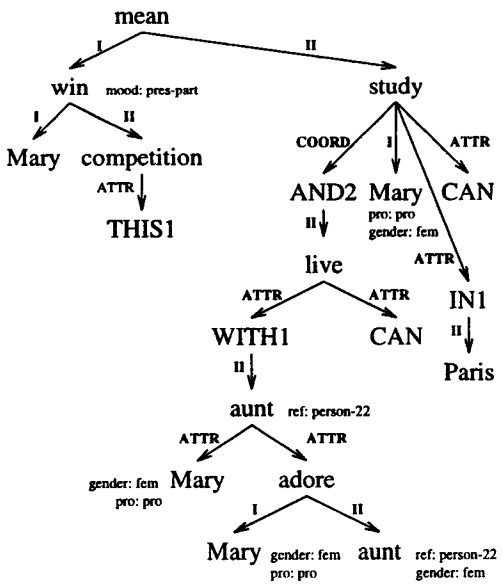

Figure 2: Input structure for sentence (3)

Note that REALPRO does not perform the task of lexical choice: the input to REALPRO must specify all meaning-bearing lexemes, including features for free pronominalization. Also, REALPRO does not map any sort of semantic labels to syntactic categories. These tasks, we assume, are handled by a separate component (such as a sentence planner). This has the advantage that the sentence planner can be unabashedly domain-specific, which is necessary in today's applications, since a broad-coverage implementation of a domain-independent theory of conceptual representations and their mapping to linguistic representations is still far from being realistic. Furthermore, there is no non-determinism in REALPRO: the input to REALPRO fully determines the output, though the input is a very abstract linguistic representation which is well suited for interfacing with knowledge-based applications. This means that REALPRO gives the developer control over the output, while taking care of the linguistic details.

3 System Architecture

The architecture of REALPRO is based on Meaning-Text Theory, which posits a sequence of correspondences between different levels of representation. In REALPRO, each transformation is handled by a separate module. REALPRO is really a realizer shell, which allows for a (run-time) configuration using specially formatted Linguistic Knowledge Bases (LKBs) which state grammar rules, lexical entries, and feature defaults. Each module draws on one or several LKBs. The lexicon is an LKB which is used by all components. Figure 3 shows the architecture.

- First, the input DSyntS is checked for syntactic validity and default features from the Default Feature Specification are added.
- The Deep-Syntactic Component takes as input a DSyntS. Using the DSynt grammar and the lexicon, it inserts function words (such as auxiliaries and governed prepositions), and produces a second dependency tree, the surface-syntactic structure or SSyntS, with more specialized arc labels.
- The Surface-Syntactic Component linearizes the nodes of the SSyntS, which yields the deep-morphological structure, or DMorphS. It draws on the SSynt grammar, which states rules of linear precedence according to arc labels.
- The Deep-Morphological Component inflects the items of the DMorphS, yielding the Surface-Morphological Structure (SMorphS). It draws on information from the lexicon, as well as on a default inflection mechanism (currently hard-coded in C++).
- The Graphical Component adds abstract punctuation and formatting instructions to the SMorphS (including "point absorption" – see (White, 1995)), yielding the Deep-Graphical Structure (DGraphS).
- *Ad-hoc* formatters transform the DGraphS into formatting instructions for the targeted output medium. Currently, REALPRO supports ASCII, HTML, and RTF output.

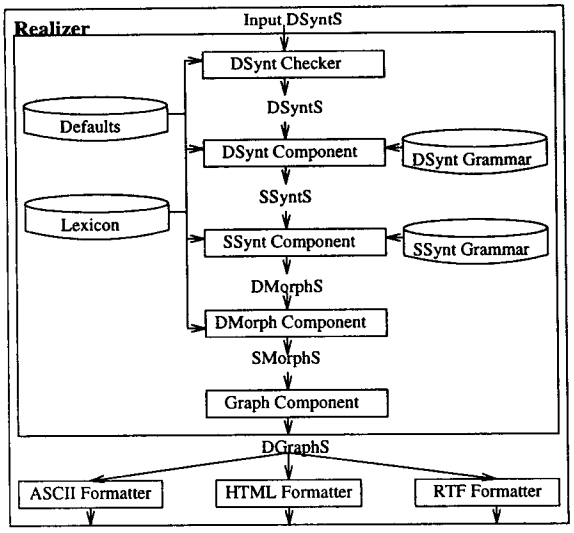

Figure 3: System architecture

4 Linguistic Knowledge Bases

As mentioned in Section 3, REALPRO is configured by specifying several LKBs. The system comes with LKBs for English; French is currently under development. Normally, the user need not change the two grammar LKBs (the DSynt and SSynt grammars), unless the grammar of the target sublanguage is not a subset of English (or French). However, the user may want to extend the lexicon if a lexeme with irregular morphology is not in it yet. (Recall that not all words in the input representation need be in the lexicon.) For example, in order to generate *saw* (rather than the default *seed*) for the past tense of *to see*, the following entry would be added to the lexicon.

```
LEXEME:      SEE
CATEGORY:    verb
MORPHOLOGY:  [([mood:past-part] seen [inv] )
              ([tense:past]     saw  [inv] )]
```

The user may also want to change the defaults. For example if in his/her application all sentences must be in past tense, the user can set the default tense to be past rather than present as follows:

```
DEFAULT:  verb [ tense:past mood:ind ]
```

5 Coverage of the English Grammar

The English grammar currently covers a wide range of syntactic phenomena:

- Full range of verbal forms (such as compound tenses, aspects, passive voice, and so on), including negation and questions. Also subject-verb agreement.
- Coordination of both nouns and clauses.
- Relative clauses (both on subject and object).
- Default word order; certain word order variations (including so-called "topicalization", i.e. fronting of adjuncts or non-subject complements) are controled through features.
- Full English morphology, including a full range of pronominal forms (personal pronouns, possessive pronouns, relative pronouns).
- Full range of punctuation, such as commas around descriptive relative clauses.

Most of these points are illustrated by the input in Figure 2. Phenomena currently not handled automatically include certain types of "fancy syntax" such as clefts and it-clefts (though these can be generated by specifying the surface structure in the input), as well as long-distance dependencies such as *These are books which I think you should buy* (where *which* is an argument of *buy*).

6 Interfaces

REALPRO is currently distributed with a socket interface which allows it to be run as a standalone server. It has an application programming interface (API), available in C++ and Java, which can be used to integrate REALPRO in applications. For training, debugging, and demonstration purposes, REALPRO can also be used in interactive mode to realize sentences from ASCII files containing syntactic specifications. The following ASCII-based specification corresponds to the DSyntS of sentence (2):

```
SEE [ question:+ ]
( I boy [ number:pl ]
     ( ATTR  THIS1  )
  II Mary [ class:proper_noun ] )
```

In this definition, parentheses () are used to specify the scope of dependency while square brackets [] are used to specify features associated with a lexeme.

REALPRO can output text formatted as ASCII, HTML, or RTF. In addition, REALPRO can also output an ASCII representation of the DGraphS that a user application can format in application-specific ways.

7 System Performance

The following table shows the runtime for sentences of different lengths. These sentences are all of the form *This small girl often claims that that boy often claims that Mary likes red wine*, where the middle clause *that that boy often claims* is iterated for the longer sentences. The row labeled "Length" refers to the length of the output string in words. Note that the number of output words is equal to the number of nodes in the SSyntS (because it is a dependency tree), and furthermore the number of nodes in the

SSyntS is greater than or equal to the number of nodes in the DSyntS. (In our case, the number of nodes in the input DSyntS is equal to the number of words in the output string.) The row labeled "Sec" represents average execution time (over several test runs) for the sentence of the given input length, in seconds, on a PC with a 150MHz Pentium processor and 32 Megs of RAM.

Length	5	10	15	20	30	40	50
Sec	.11	.17	.20	.28	.44	.58	.72

We also tested the system on the syntactically rather varied and complex input of Figure 2 (which is made up of 20 words). The average runtime for this input is 0.31 seconds, which is comparable to the runtime reported above for the 20 word sentence. We conclude that the uniformity of the syntactic constructions found in the sentences used in the above test sequence does not influence the results.

The complexity of the generation algorithm derives primarily from the tree traversals which must be performed twice, when passing from DSyntS to SSyntS, and from SSyntS to the DMorphS. Let n be the length of the output string (and hence an upper bound on the size of both DSyntS and SSyntS). At each node, each rule in the appropriate grammar (deep- or surface-syntactic) must be checked against the subtree rooted at that node. This tree matching is in the general case exponential in n. However, in fact it is dependent on two variables, the maximal size of grammar rules in the grammar (or n, whichever is greater), and the branching factor (maximum number of daughter nodes for a node) of the input representation. Presumably because of deeper facts about language, the grammar rules are quite small. The current grammar does not have any rules with more than three nodes. This reduces the tree matching algorithm to polynomial in n. Furthermore, while the branching factor of the input tree can in theory be $n - 1$, in practice it will be much smaller. For example, all the input trees used in the tests discussed above have branching factors of no more than 5. We thus obtain de-facto linear performance, which is reflected in the numbers given above.

8 Status

The system is fully operational, runs on PC as well as on UNIX work stations, and is currently used in an application we have developed (Lavoie et al., 1997) as well as in several on-going projects (weather report generation, machine translation, project report generation). REALPRO is licensed free of charge to qualified academic institutions, and is licensed for a fee to commercial sites.

Acknowledgments

The development of REALPRO was partially supported by USAF Rome Laboratory under contracts F30602-93-C-0015, F30602-94-C-0124, and F30602-92-C-0163, and by DARPA under contracts F30602-95-2-0005 and F30602-96-C-0220. We are grateful to R. Kittredge, T. Korelsky, D. McCullough, A. Nasr, E. Reiter, and M. White as well as to three anonymous reviewers for helpful comments about earlier drafts of this technical note and/or about REALPRO.

References

Bateman, J. A. (1996). KPML development environment. Technical report, Institut für Integrierte Publikations- und Informationssysteme (IPSI), GMD.

Coch, J. (1996). Overview of AlethGen. In *Proceedings of the Eighth International Natural Language Generation Workshop (INLG'96) (Demonstrations Volume)*, Herstmonceux Castle, Sussex.

Elhadad, M. and Robin, J. (1996). An overview of SURGE: a reusable comprehensive syntactic realization component. In *Proceedings of the Eighth International Natural Language Generation Workshop (INLG'96) (Demonstrations Volume)*, Herstmonceux Castle, Sussex.

Iordanskaja, L., Kim, M., Kittredge, R., Lavoie, B., and Polguère, A. (1992). Generation of extended bilingual statistical reports. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING'92)*.

Iordanskaja, L., Kittredge, R., and Polguère, A. (1988). Implementing the Meaning-Text Model for language generation. Paper presented at COLING-88.

Lavoie, B., Rambow, O., and Reiter, E. (1997). Customizable descriptions of object-oriented models. In *Proceedings of the Conference on Applied Natural Language Processing (ANLP'97)*, Washington, DC.

Mel'čuk, I. A. (1988). *Dependency Syntax: Theory and Practice*. State University of New York Press, New York.

Rambow, O. and Korelsky, T. (1992). Applied text generation. In *Third Conference on Applied Natural Language Processing*, pages 40–47, Trento, Italy.

Reiter, E. (1994). Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? In *Proceedings of the 7th International Workshop on Natural Language Generation*, pages 163–170, Maine.

White, M. (1995). Presenting punctuation. In *Proceedings of the Fifth European Workshop on Natural Language Generation (EWNLG5)*.