A Development Environment for MTT-Based Sentence Generators

— Demonstration Note —

Bernd Bohnet, Andreas Langjahr and Leo Wanner
Computer Science Department, University of Stuttgart
Breitwiesenstr. 20-22, 70565 Stuttgart, Germany
{bohnet|langjahr|wanner}@informatik.uni-stuttgart.de

Introduction

With the rising standard of the state of the art in text generation and the increase of the number of practical generation applications, it becomes more and more important to provide means for the maintenance of the generator, i.e. its extension, modification, and monitoring by grammarians who are not familiar with its internals. However, only a few sentence and text generators developed to date actually provide these means. One of these generators is KPML (Bateman, 1997). KPML comes with a development environment and there is no doubt about the contribution of this environment to the popularity of the systemic approach in generation.

In this note, we describe a demonstration of a high quality development environment (henceforth DE) for *Meaning-Text Theory*-based sentence generators. The Meaning-Text Theory (MTT) (Mel'čuk, 1988) is a dependency-based linguistic theory, which becomes increasingly popular for generation; cf. e.g., (Iordanskaja *et al.*, 1992; Lavoie & Rambow, 1997; Coch, 1997). The introduction of MTT is beyond the scope of this note; the interested reader is asked to consult the above references.

The DE provides support to the user with respect to writing, modifying, testing, and debugging of (i) grammar rules, (ii) lexical information, and (iii) linguistic structures at different levels of abstraction. Furthermore, it automatically optimizes the organization of the lexica and the grammar by structuring them hierarchically.

DE is written in Java 1.2 and has been tested on a SUN workstation and on a PC pentium with 300 MHz and 128 MB of RAM.

Global View on the DE

In MTT, seven levels (or strata) of linguistic description are distinguished, of which five are relevant for generation: semantic (Sem), deep-syntactic (DSynt), surface-syntactic (SSynt), deep-morphological (DMorph) and surface-morphological (SMorph). The generation process consists of a series of structure mappings between adjacent strata until the SMorph stratum is reached. At the SMorph stratum, the structure is a string of linearized word forms.

The central module of the DE is a compiler that maps a structure specified at one of the five first of the above strata on a structure at the adjacent stratum. To support the user in the examination of the internal information gathered during the processing of a structure, a debugger and an inspector are available. The user can interact with the compiler either via a graphic interface or via a text command interface. For the maintenance of the grammar, of the lexica and of the linguistic structures, the DE possesses separate editors: a rule editor, a lexicon editor, and a structure editor.

The Rule Editor. Most of the grammatical rules in an MTT-based generator are two-level rules. A two-level rule establishes a correspondence between minimal structures of two adjacent strata. Given that in generation five of MTT's strata are used, four sets of two-level rules are available: (1) Sem⇒DSynt-rules, (2) DSynt⇒SSynt-rules, (3) SSynt⇒DMorph rules, and (4) DMorph⇒SMorph-rules.

The rule editor (RE) has two main functions: (i) to support the maintenance (i.e. editing and examination) of grammatical rules, and (ii) to optimize the organization of the grammar by automatic detection of common parts in several rules and their extraction into abstract 'class' rules. The theoretical background and the procedure of rule generalization is described in detail in (Wanner & Bohnet, submitted); a brief overview will also be given during the presentation.

Rule testing is usually a very time consuming procedure. This is so partly because the generator needs to be started as a whole again and again, partly because the resulting structure and the trace must be carefully inspected in order to find out whether the rule in question fired and if it did not fire why it did not. The DE attempts to minimize this effort. With 'drag and drop' the developer can select one or several rules and apply them onto an input structure (which can be presented either graphically or in a textual format; see below). When a rule dropped onto the structure fires, the affected parts of the input structure are made visually prominent, and the resulting output (sub)structure appears in the corresponding window of the structure editor. If a rule did not fire, the inspector indicates which conditions of the rule in question were not satisfied. See also below the description of the features of the inspector.

The Structure Editor. The structure editor manages two types of windows: windows in which the input structures are presented and edited, and windows in which the resulting structures are presented. Both types of windows can be run in a text and in a graphic mode. The input structures can be edited in both modes, i.e., new nodes and new relations can be introduced, attribute-value pairs associated with the nodes can be changed, etc.

In the same way as rules, structures can be checked with respect to their syntax and semantics. Each structure can be exported into postscript files and thus conveniently be printed.

The Lexicon Editor. The main function of the lexicon editor is to support the maintenance of the lexica. Several types of lexica are distinguished: conceptual lexica, semantic lexica and lexico-syntactic lexica.

Besides the standard editor functions, the lexicon editor provides the following options: (i) sorting of the entries (either alphabetically or according to such criteria as 'category'); (ii) syntax check; (iii) finding information that is common to several entries and extracting it into abstract entries (the result is a hierarchical organization of the resource). During the demonstration, each of these options will be shown in action.

The Inspector. The inspector fulfils mainly three functions. First, it presents information collected during the application of the rules selected by the developer to an input structure. This information is especially useful for generation experts who are familiar with the internal processing. It concerns (i) the correspondences established between nodes of the input structure and nodes of the resulting structure, (ii) the instantiation of the variables of those rules that are applied together to the input structure in question, and (iii) the trace of all operations performed by the compiler during the application of the rules.

Second, it indicates to which part of the input structure a specific rule is applicable and what its result at the destination side is. Third, it indicates which rules failed and why. The second and third kind of information is useful not only for generation experts, but also for grammarians with a pure linguistic background.

The Debugger. In the rule editor, break points within individual rules can be set. When the compiler reaches a break point it stops and enters the debugger. In the debugger, the developer can execute the rules statement by statement. As in the inspector, the execution trace, the variable instantiation and node correspondences can be examined. During the demonstration, the function of the debugger will be shown in action.

References

Bateman, J.A. 1997. Enabling technology for multilingual natural language generation: the KPML development environment. *Natural Language Engineering*. 3.2:15–55.

Coch, J. 1997. Quand l'ordinateur prend la plume : la génération de textes. *Document Numérique*. 1.3.

Iordanskaja, L.N., M. Kim, R. Kittredge, B. Lavoie & A. Polguère. 1992. Generation of Extended Bilingual Statistical Reports. *COLING-92*. 1019–1022. Nantes.

Lavoie, B. & O. Rambow. 1997. A fast and portable realizer for text generation systems. *Proceedings of the Fifth Conference on Applied Natural Language Processing*. Washington, DC.

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

Wanner, L. & B. Bohnet. submitted. Inheritance in the MTT-grammar.