Ten Years After: An Update on TG/2 (and Friends)

Stephan Busemann
DFKI GmbH
Stuhlsatzenhausweg 3
D-66123 Saarbrücken
busemann@dfki.de

Abstract

Since its first implementation in 1995, the shallow NLG system TG/2 has been used as a component in many NLG applications that range from very shallow template systems to in-depth realization engines. TG/2 has continuously been refined, the Java brother implementation XtraGen has become available, and the grammar development environment eGram today allows for designing grammars on a more abstract level. Besides a better understanding of the usability of shallow systems like TG/2 has emerged. Time has come to summarize the developments and look forward to new borders.

1 Introduction

Shallow NLG is known as "quick and dirty" on the one hand, and as a practical approach to implementing real-world applications on the other. Its legitimization stems from practical success rather than from theoretical advantages. As with shallow analysis, methods have become acceptable that had been rejected twentyfive years ago as linguistically unjustified. For instance, template-based NLG systems were known to be unscalable and unflexible. Besides they were quite trivial and did not contribute to solving any research questions in the field. However, it became evident that many practical applications involving NLG required limited linguistic coverage, used canned text and/or templates, and badly needed improvements to make the NLG systems more flexible. A revival of template-based systems followed, and subsequent scientific discussions clarified the relation to more advanced NLG research themes. The papers in [Becker and Busemann, 1999] nicely show a continuum between template- and "plan"-based systems.

Since its first implementation in 1995, the shallow NLG system TG/2 [Busemann, 1996] has been used as a component in several diverse applications involving NLG. Implemented in Common Lisp, TG/2 has continuously been refined over the years; a Java brother implementation, called XtraGen, has eventually become available, and the grammar development environment eGram eventually allows the grammar writer to design large-scale grammars.

Among the attractive properties of TG/2 is the quick development of new NLG applications with limited requirements on linguistic expressiveness. Numerous implementations show that TG/2 is well suited for simple dialogues, report generation (from database content), and even as a realizer for complex surface-semantic sentence representations. Besides a better understanding of the pros and cons of TG/2 has emerged.

Time has come to summarize these developments and, more generally, reassess the value of TG/2 as a framework to specify generation systems.

In the following section, TG/2 is localized on the NLG map, clarifying a few common misconceptions on what it can be used for. In Section 3 we sketch major use cases involving TG/2 that exhibit different degrees of "shallowness". Section 4 summarizes the major extensions and refinements that have been implemented over the last decade, taking into account some critical comments from the literature. We then describe in Section 5 the need for, and the benefits of, the dedicated grammar development environment eGram that supports the fast developments of large rule sets. The paper concludes with an outlook to upcoming work.

2 What TG/2 is and What it isn't

TG/2 has been described originally in [Busemann, 1996; Busemann and Horacek, 1998] as a template-based generator. To remind the reader of the main points, TG/2 is a flexible production system [Davis and King, 1977] that provides a generic interpreter to a separate set of user-defined condition-action rules representing the generation grammar. The generic task is to map a content representation, which must be encoded as a feature structure[1], onto a chain of terminal elements as defined by the rule set. The rules have a context-free categorial backbone used for standard top-down derivation guided by the input representation. The rules specify conditions on the input – the so-called test predicates – that determine their applicability. Due to the context-free backbone each subtree of depth 1 in a derivation tree corresponds to the application of one rule. TG/2 is equipped with a constraint propagation mechanism that supports the establishment of agreement relations across the derivation tree. Figure 1 shows a sample rule.

[1] A feature structure is either an atomic value, or a pair `[feature-name feature-value]`, where `feature-name` is a string and `feature-value` a feature structure.

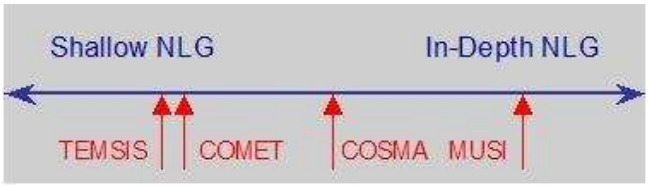

Figure 2: TG/2-based NLG applications Arranged on a Scale From Shallow to In-Depth Generation.

TG/2 production rules has a simple interpretation procedure that corresponds to the classical three-step evaluation cycle in production systems (matching, conflict resolution, firing) [Davis and King, 1977]. The algorithm starts from a (piece of the) input structure and a category.

1. **Matching:** Select all rules carrying the current category. Execute the tests for each of these rules on the input structure and add those passing their test to the conflict set.
2. **Conflict resolution:** Select an element from the conflict set, e.g. on the basis of some conflict resolution mechanism.
3. **Firing:** Evaluate its constraints (if any). For each right-hand side element, read the category, determine the substructure of the input, and goto step 1.

The processing strategy is top-down and depth-first. The set of actions is fired from left to right. Failure of executing some action causes the rule to be backtracked.

The right-hand side of a rule can consist of any mixture of terminal elements (canned text) and non-terminal categories, as in Figure 1. The presence of canned text is useful if the input does not express explicitly everything that should be generated. The grammar thus adds text to the output that does not have an explicit semantic basis in the input. With very detailed input and hence less "implicit" semantics, only little canned text will be needed in the grammar, and the terminal elements of the grammar usually are word stems.

Canned parts of the grammar are "invented". This gives rise to the notion of "shallow generation", as opposed to shallow analysis, where parts of the input text are ignored. TG/2 leaves complete freedom to using canned text, mixing it with context free rules, or sticking to the more traditional distinction between (context-free) rules and the lexicon. [Busemann and Horacek, 1998] refer to the former kind as shallow and to the latter as in-depth generation. One may thus identify TG/2 applications on a scale ranging from more shallow to more in-depth systems. Figure 2 attempts to compare some TG/2-based NLG applications along this dimension. They will be discussed in Section 3.

As mentioned above, there is no strict borderline between template-based and plan-based generation systems. While this insight resulted from comparing different systems, TG/2 implements this claim by forming a single framework that may host any approach ranging from pure canned text to completely lexicon-based. As Section 3 demonstrates, TG/2 can implement template-based systems and full-fledged realizers.

In an attempt to relate existing NLG systems to the RAGS framework [Mellish *et al.*, 2000], TG/2 was among the systems to look at. It turned out that TG/2 differs from the principles underlying RAGS in that it does not support any of the levels of conceptual, semantic, rhetoric, document or syntactic representation, which were abstractly defined to capture many (most) NLG approaches. Rather TG/2 entails a single mapping from input to output, and any tasks generally ascribed to components delivering the above intermediate representations must be encoded by one or several production rules. There is no pipeline of modules with intermediate representations, as ideally assumed in RAGS. Rather all tasks need to be encoded within the production rules. During this experiment it actually became evident that TG/2 isn't a classical generation system at all.

In non-trivial NLG applications, TG/2 is complemented by other components. On the output side it can be hooked up to morphological inflection components using a shared representation of word stems and morpho-syntactic features. On the input side TG/2 has been combined with a text structuring component in the TEMSIS application, with a context management system in COMET, and with a lexical choice component in the MUSI system.

3 Major Use Cases

3.1 Template generation in the appointment scheduling domain

Software agents communicated with human agents in order to schedule appointments on behalf of their owners. Communicative goals to be verbalized as dialogue steps in the COSMA system [Busemann *et al.*, 1994] include just a few speech acts for proposing, accepting, modifying, rejecting, or confirming dates or date intervals. See [Busemann, 1996] for a discussion and examples.

The event-oriented input is created by a non-linguistic component, the scheduling agent system, and converted into a surface-semantic representation, referred to as GIL in [Busemann, 1996], which is verbalized by TG/2.

GIL was defined since the necessary distinctions at the linguistic level are often based on information distributed all over the input structure. For instance, the choice of prepositions depends on the choice of verbs, which is based on the speech act. TG/2 can only access part of the input at a given moment. Yet generating directly from event-oriented input would have been possible at the cost of complex tests or constraints – thereby affecting the transparency of the grammar – or of more backtracking.

The necessary restructuring was implemented by simply exploiting the expressive power of the directed acyclic graphs used for feature structure input representation in TG/2. Using co-references, relations between the event-based `IN` and the language-oriented `OUT` feature of a set of semantic templates covering all possible inputs were defined. An input is unified with the `IN` feature, and TG/2 generates from the associated `OUT` value.

An additional, practical reason for adopting an "internal" `OUT` representation is to encapsulate the generation grammar, rendering it independent of external changes of the input

```
(defproduction "s2 top-subj.1"
    (:PRECOND (:CAT DECL
               :TEST ((sbp 's2) (top-deep-subj 'y) (vc-voice 'active)))
     :ACTIONS (:TEMPLATE (X1 :RULE ARG 'deep-subj)
                         (X2 :RULE FIN 'vc)
                         (X3 :RULE ARG 'deep-obj)
                         (X4 :OPTRULE INF 'verb-complex)
               :CONSTRAINTS ( X1.CASE := 'nom
                              X3.CASE := 'acc
                              X1.NUMBER = X2.NUMBER = X4.NUMBER
                              X1.PERSON = X2.PERSON = X4.PERSON ))))
```

Figure 1: A rule for the German transitive main clause in the MUSI grammar in the format processed by TG/2. Tests specify that this rule is applicable if the input suggests a certain syntactic structure called "s2", the subject should be in topic position, and active voice is called for. The context-free rule underlying the rule is DECL → ARG FIN ARG {INF}. Path expressions following the category such as deep-subj refer to substructures of the input, cf. Figure 3. Feature constraints assign nominative case to the first ARG and accusative case to the second; number and person are set to be equal on the first ARG and the verb complex, thus establishing subject verb agreement. Notation: Constraint variables refer to right-hand side elements by virtue of the indices X_i. The reserved index for the left-hand side is X_0.

language, which are accommodated by the feature structure mappings.

As was to be expected it turned out that GIL, as it stood, was never reused. Instead other internal encodings were required, which could, however, be implemented straightforwardly using the technique mentioned above.

3.2 Shallow multilingual generation from non-linguistic input

Later projects required the verbalization of non-linguistic domain-specific representations in multiple languages. In Mietta, database content is verbalized in German, Finnish, Italian and English as part of cross-language information retrieval [Xu *et al.*, 2000]. A useful input representation is created by applying a similar mechanism as in COSMA.

In COMET, TG/2 is used to generate personalized activity recommendations in a conference scenario that differ with the context consisting of interest and focus values, which form part of the input [Geldof, 1999]. This application uses the possibility of creating side-effects from applying a rule to update a discourse memory whenever a discourse referent is mentioned. It is indeed possible to create arbitrary side-effects by function calls, but care has to be taken that these functions can be called a second time during backtracking to undo the side effects. As side-effects are rarely required, its backtrack functionality is currently not supported and thus requires explicit Lisp (and Java) programming.

In TEMSIS, air quality reports reports are generated from a database containing measurement data [Busemann and Horacek, 1998]. The communicative goal is interactively specified by the user: the type of report (time series, threshold passing, etc.), the measuring station, the pollutant, a time interval of interest, and some further options relating to content selection. The generated texts can include confirmations about the user's choices as well as canned paragraphs informing about the measuring station or the pollutant in question. Corresponding views on previous periods are generated for comparison. If the report is composed of multiple elements, it is concluded by a summary that answers the key question again.

A separate, language-independent component for text structuring, accessing the database and calculating further values (e.g., average values) was implemented. It produces the actual inputs for TG/2, each corresponding to one paragraph. All language-specific issues remained within the TG/2 grammar.

The TEMSIS project was designed to be used in a border region of Germany and France. Thus the application initially generated German and French texts. The grammars comprised about 120 and 100 rules, respectively. In order to find out more details about the speed of grammar development for such an application, and in particular the time it takes to transport the application to a new language, native speakers of English, Chinese, Japanese and Portuguese were asked to produce a TG/2 grammar with the same coverage. Depending on programming skills, it took between two and four person-weeks to get acquainted with the system and to complete the final tests set up. While this result was very encouraging, the grammar writers stated that larger grammars would be less easily developed and maintained.[2]

As [Busemann and Horacek, 1998] show, the input structures are non-linguistic, i.e., they do not uniquely determine the content-bearing linguistic elements and the sentential structure to be used. These matters were defined in co-operation with the users (cf. [Reiter *et al.*, 1997]). The agreed-upon pieces of text were entered as canned parts into the grammar.

The grammars of Mietta, COMRIS and TEMSIS contain much more canned parts than the COSMA grammar. This is in direct correspondence to the nature of the respective input representations (see Figure 2).

3.3 In-depth realization of surface-semantic sentence representations

A much more in-depth use case for TG/2 is the generation of German sentences that form part of cross-lingual sum-

[2] A demonstrator of the resulting multilingual system is online at http://www.dfki.de/service/nlg-demo.

maries of scientific medical papers written in Italian or English (MUSI project, [Lenci *et al.*, 2002]). The sentences exhibit quite a complicated structure and much medical terminology. Their average length in a sample corpus is 22 words. The input structures (cf. Figure 3 for an example) are the results of a lexical and syntactic choice component [Busemann, 2002] that feeds TG/2. The structures contain specific references to syntactic "plans" (features SBP and NR). Test predicates in the rules check for these features, thus realizing the corresponding structure (cf. Figure 1). The morpho-syntacic features for inflecting the lexical stems are collected through the constraint mechanism and made available to the separate word inflection components MORPHIX-3 [Finkler and Neumann, 1988].

The input is a rather typical for linguistic realization, a task initially not deemed suitable for systems like TG/2. Previous applications show that TG/2 grammars are domain-dependent and must be replaced when a new task is at at stake. [Busemann and Horacek, 1998] consider this lack of reuse a disadvantage, but state that it is nevertheless acceptable since new grammars can be developed very quickly. For realization, however, a linguistically justified, domain-independent grammar is needed that is expensive to develop but can be reused across applications.

The parts of a grammar rule depending on input elements can be isolated and treated as an interface between the grammar and any input language. If an input language changes, the test predicates and the access to input substructures need to be recoded.[3] This interface allows us to develop generic grammar knowledge that abstracts from specific semantics of test predicates and access details. We call such a generic grammar a *protogrammar*, as it is supposed to form the reusable basis for different instances geared towards different applications. Technically, a protogrammar can be instantiated by defining the test predicates and access methods needed for the input language in question.

The protogrammar developed covers the main types of sentential structures, as specified by the Duden grammar [Dudenredaktion, 1998]. The NP syntax comprises prenominal APs (on the basis of adjective subcategorization frames), generic possessive constructions, a temporal, a locative and an adverbial modifier and a relative clause. In addition, nouns and adjectives can subcategorize for specific arguments.

How could a protogrammar be developed independently of a particular input language, as it needs testing? An intuitive, minimal input representation would need to determine the depth of the nesting of constituents (as to avoid endless recursion), specify morpho-syntactic features such as case, number, tense etc., indicate the prepositions and distinguish the syntactic adjuncts at the sentence and NP level. After defining a corresponding language, grammar development could proceed in a way independent of the MUSI application. When the other parts of the MUSI system became stable and well-defined, the necessary adaptation of the input language and the grammar were made. It goes without saying that the grammar had to be extended to cover linguistic structures not foreseen explicitly in [Dudenredaktion, 1998], but the additional effort was surprisingly small.[4]

[3] Changes may include restructuring and recoding of information. Obviously if the input language encodes different kinds of information, the grammar possibly cannot be reused.

```
(defproduction "parser-grammar"
  (:PRECOND (:CAT ANALYSIS
             :TEST ((always-true))
   :ACTIONS (
      :TEMPLATE (X1 :RULE PARSER 'self)
                (X2 :RULE GRAMMAR 'self)
      :CONSTRAINTS ( X0.LANG = X2.LANG
                     X1.API = X2.API ))))
```

Figure 4: A rule linking a parser and a grammar with compatible interfaces. The parser is language-independent, the grammar is not. The LANG feature is specified in the input, whereas the API feature is specified in the rules representing the individual grammars.

The MUSI grammar comprises about 950 rules with 135 categories, and 14 features for constraints. A sample rule is shown in Figure 1. During the development of this large grammar the use of standard text editors became a nuisance. A grammar development environment was designed and implemented that supports multi-format development of large grammars (see Section 5). With this system, all practical needs arising from using TG/2 as a syntactic realizer could be fulfilled.

3.4 Other usage

TG/2 is general enough to be usable for other tasks than NLG. A sample grammar for software configuration has been written using the constraint mechanism to define API properties of software components (the "lexicon") and matching conditions for components to be integrated into a larger piece of software (the "grammar"). The input is a specification of the desired system (e.g., machine translation from Spanish to English), and the system would enumerate the possible specifications it can derive. A sample rule is shown in Figure 4.

4 Modifications and Extensions

The experience gained from the various applications suggested some modifications and extensions to the system. Also a closer look at comparable systems, most importantly YAG [McRoy *et al.*, 2003], revealed opportunities for improvement. YAG differs from TG/2 in that it is deterministic and thus does not search. Every next rule to be chosen is depicted in the input or identified by a preceding table lookup. Therefore YAG is probably faster than TG/2 since TG/2 lets the interpreter select the next rule. On the other hand, as we will see in Section 4.2, this gives TG/2 some flexibility YAG does not exhibit: TG/2 output can vary according to non-linguistic parameters.

The need for backtracking associated with search can be kept small in practice. Moreover the costs are small since TG/2 reuses previously generated substrings during backtracking, as described in [Busemann, 1996]. In practice an-

[4] The author's guess is that adaptation work required about 20% of the overall effort; unfortunately no reliable figures are available.

```
[(SENTENCE DECL)
 (VC [(SBP S2)                                                ;;name of sentence plan
      (G AKTIV)                                                    ;;active voice
      (STEM "verursach")])
 (DEEP-OBJ [(DET DEMONST) (STEM "wirkung")])
 (DEEP-SUBJ [(TOP Y)                                ;;this constituent to the fore-field
             (DET INDEF)                                        ;;indefinite article
             (NR V2)                                          ;;name of nominal plan
             (STEM "antagonismus")
             (PP-ATR [(MODALITY
                          [(PP-OBJ
                             [(TERM [(DET DEF)                      ;;definite article
                                     (STEM "bindungsstelle")
                                     (ADJ [(STEM "muskarinisch") (DEG POS)])
                                     (TERM [(DET DEMONST1)             ;;demonstrative
                                            (STEM "substanz")])])
                              (STEM "Niveau") (DET DEF)
                              (PREP AUF-DAT)])])          ;;P governs dative NP here
                      (STEM "acetylcholin")
                      (DET WITHOUT)                                    ;;no article
                      (PREP ZU)])                ;;this P always governs dative NP
                (ADJ [(STEM "kompetitiv") (DEG POS)])])]
```

Figure 3: A TG/2 MUSI input for “Ein kompetitiver Antagonismus zu Acetylcholin auf dem Niveau der muskarinischen Bindungsstellen dieser Substanzen verursacht diese Wirkungen.” [“These effects are caused by a competitive antagonism with acetylcholine on the level of the muscarinic sights of these substances.”]. Comments are separated by semicolons. The structure is simplified by omitting gender, number, mood and phrase type information.

other cost factor turned out to be sensible, namely the number of rules to be checked in each cycle. In experiments with automatic rule generation using meta-rules in eGram, [Rinck, 2003] showed a linear increase of TG/2 runtime with the number of rules per left-hand side category.

With large grammars such as in MUSI, which has about 950 rules, it is important to reduce the number of alternative rules. This can be achieved by using optional right-hand side elements, thus covering many possible verbalizations by a single rule. The semantics of optional right-hand side elements has been refined to capture this idea fully; they must be verbalized if and only if there is input for them. A right-hand side element failing on non-empty input causes the parent rule to fail.

4.1 Reducing the need for programming

In comparing YAG and TG/2, [McRoy *et al.*, 2003, p. 417] observe that “YAGs template language is also more declarative, yielding higher maintainability and comprehensibility”. While they do not point out details, it is true that defining TG/2 rules requires some Lisp programming. The test predicates have to be defined and the defined ones have to be called properly, and, in the version reviewed, the access functions to relevant parts of the input have to be specified. Calling an access function such as `(theme)` should return some part of the input structure that is accessible at the current state of processing. The function would encapsulate the way this is achieved. Access functions may fail, in which case the parent rule may fail.

A number of frequently used general test predicates such as testing the presence of a feature at a certain location in the input structure, equality of some feature value with a given object, or a list element being the but last one in a list, can be held on offer for grammar developers. Usually most tests can be carried out using one of these, but new demands need programming. Since the structure of the Boolean test predicates is simple, such tasks are not difficult to solve. eGram offers support as for Java all the embedding code such as exception handling is provided and only the core condition must be written.

Access functions were met with some disgust by grammar writers as new ones are required with the change of the input language, i.e. with any new generation task. It turned out that the advantage of having the access to input material encapsulated did not pay off. The implied possibility of reorganizing or renaming input was never used, as other ways to do this were preferred (cf. [Busemann, 1999]). Instead just relative path descriptions were implemented. The need to provide access functions in both Lisp and Java eventually gave rise to a uniform solution: now a single format for relative path descriptions is used in eGram as a source code that is compiled into Lisp and Java expressions to serve the runtime systems. Hence the rule in Figure 1 now has a feature path on each right-hand side element instead of function calls, as in [Busemann, 1996].

4.2 Generating Personalized Text

Given a certain input, different outputs may well desirable for different users. Some examples:

- A user may be an expert or a novice in the topic at stake. Expert users will read terminology whereas novices need explanations or circumscriptions.
- Depending on whether a user is interested in receiving

background information, relevant hyperlinks may be inserted into the text.

- When text is generated for display on a hand-held device, it must be organized and presented differently.

Sometimes the components producing input for the generator are not capable of accounting for the respective linguistic differences since they don't have a model of the grammar at their disposal. A mechanism is needed to feed the system with parameters corresponding to such distinctions and to translate the parameter settings into appropriate decisions in the generation process. For this purpose the approach to parameterization introduced in [Busemann, 1996] has been refined.

First and foremost, all variations that could be generated for a given input must be covered by the grammar. Then the system would produce the complete set, one by one. The grammar writer defines, in cooperation with the application developer, parameters such as *expertise*, *background*, and *device* with appropriate values. She tags all rules that exhibit properties of some parameter value. Then the system can select a rule according to a single parameter. But parameters may be in conflict as well. TG/2 offers a linear preference scheme for the defined parameters implemented in step 2 of the basic algorithm. The grammar writer defines, in cooperation with the application developer, a partial order describing the relative importance of the parameters. With this scheme the system can produce a text that conforms best to the user's preferences.

The scheme is best explained using an example of two parameters. Let us assume that the user chooses "non-expert" text with background information. Assume that a conflict set contains the following tagged rules: {`R1-[expertise: expert, background: -]`, `R2-[expertise: non-expert, background: -]`, `R3-[expertise: expert, background: +]`}. None of the tags matches exactly the specifications. If the parameter *expertise* is defined to be more important than *background*, `R2` will be selected. If, however, *background* is preferred over *expertise*, `R3` is applied. [Busemann, 1998] has a more detailed description of this idea.

While subsequent experiments seem to show the viability of this simple approach to let other components (or the user, via a task interface) influence the system behavior[5], a real test will probably consist in its envisaged usage for answer presentation in Semantic Web contexts.

5 Grammar Development

The development of small grammars with 100 to 200 rules such as the ones underlying COSMA, TEMSIS, Mietta or COMET could safely be developed with standard text editors using the syntax exemplified in Figure 1. However even in this work, the difficulty of maintenance and a considerable error-proneness were observed. With the MUSI grammar, a dimension was reached that made a dedicated grammar development environment necessary. While some abstraction from the Lisp-like rule format was desirable, the Java implementation XtraGen [Stenzhorn, 2002] required a different format anyway, as it is consistently using XML to encode all objects.

eGram [Busemann, 2004] was hence designed to developing grammars without bothering about their syntax or size or interpreting NLG system. Major benefits of eGram include

- a developer-friendly grammar format,
- syntactic and semantic checks of grammar knowledge,
- the option to derive additional grammar rules by meta-rules, and
- integration with grammar testing in generation systems.

A major difficulty in the course of developing the MUSI grammar was to maintain consistency. Features used are sometimes not defined, values are not sufficiently restricted, or certain categories do not occur in any other rule. When such grammars are interpreted, errors occur that can be difficult and time-consuming to trace. eGram verifies that every new piece of grammar knowledge is fully consistent with what already exists, thus eliminating many obvious sources of mistake.

eGram allows the definition of complex objects only after all their elements are defined. Before a rule may be entered, the categories, test predicates, access paths and constraints used must be defined. The eGram GUI offers dynamically generated menus for more complex elements in addition to textual input windows, where these remain necessary. For the definition of e.g. a constraint, a menu would offer all defined features, and for the selected feature, all defined values.

Different working styles are supported: either the grammar writer pro-actively plans her work by first defining all low-level elements and then proceeding to higher-level ones, or she prefers to add missing elements "on the fly", i.e. when eGram complains.

eGram's main pane contains a set of tabs corresponding to the different elements. Clicking on a tab opens a new screen with all the tabs remaining available at any moment (see Figure 5). A set of tabs opens separate sub-panes allowing for the definition of the tests, RHS elements, and constraints of rules.

In MUSI the major disadvantage of context-free grammars posed a problem. The rules cannot easily express certain linguistic phenomena, such as word order variation, pronominalization, voice, the relation between sentential structures and relative clauses, or verb positions. To cover these phenomena, several hundreds, if not thousands, of different rules must be defined. Every-day practice involved copy-and-paste approaches that are error-prone. Moreover such phenomena are often captured only partially, leaving unknown gaps in the coverage of the grammar.

eGram is equipped with a meta-rule mechanism that is technically similar to that of Generalized Phrase Structure Grammars [Gazdar *et al.*, 1985]. Meta-rule expansion starts with a set of base rules and then applies to the set of base rules and derived rules. Meta-rules serve as an abbreviation technique and do not affect the expressive power of the system. The basic meta-rule mechanisms and their integration into eGram are described in detail in [Rinck, 2003]. A redesign of the MUSI grammar led to a reduction to 452 base

[5] Parameters should not depend on each other to guarantee that the best version is generated first, cf. the discussion in [Busemann, 1996, Section 5].

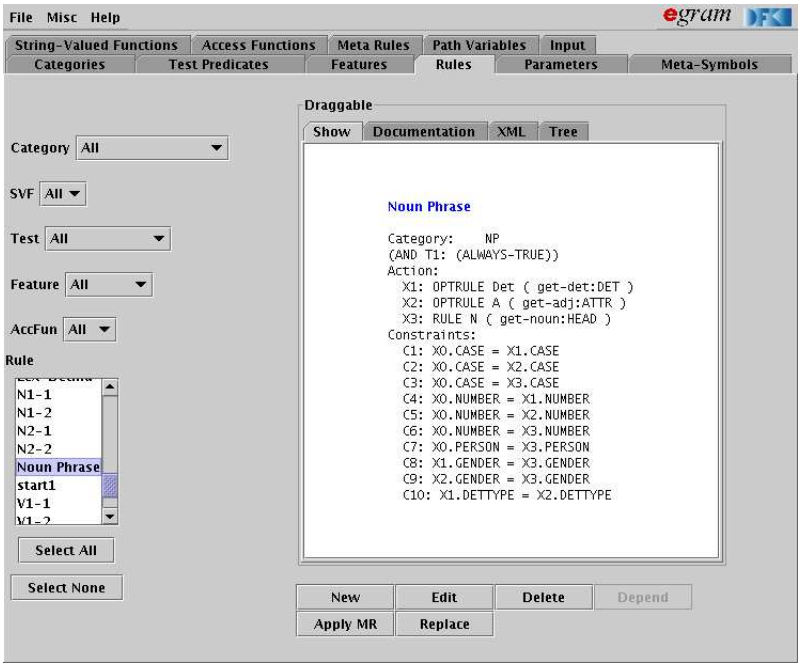

Figure 5: A Screenshot of eGram with the Rule Pane Active. It displays a simple NP rule for German. The feature constraints express various agreement relations. The rule window can be dragged to some other location on the screen, allowing to view multiple objects at the same time. The rules names shown on the left-hand side can be filtered by the elements contained in the rules. For instance by selecting category NP, only the rules with NP as their LHS category are shown.

rules. Applying to these base rules 19 meta-rules modeling the above phenomena resulted in 2.488 derived rules, demonstrating that the original grammar did in fact not systematically cover all the phenomena represented by the meta-rules.

Integrating grammar development and grammar testing is crucial to verify the effects of modifying a grammar. eGram is implemented in Java and integrated with TG/2 via a client-server interface. The integration with XtraGen is achieved via a Java API. eGram provides suitable export formats for both. Calls to the generators can be issued from within eGram. A call to a running generation system consists of an input structure that can be defined within eGram, and the modifications of the grammar since the last call. The generator either returns the generated string or an error message.

6 Conclusions and Outlook

TG/2 has been used continuously for more than ten years. From its first appearance as a "template generator" it has evolved into a framework that accommodates both shallow template-based generation and in-depth realization tasks. In combination with the grammar development environment eGram, large grammars can be developed and maintained. They can be used by both TG/2 in Lisp and XtraGen in Java.

To take up the comparison with YAG again, the most important difference is perhaps the way the rules are defined. YAG uses complex nested conditionals covering alternative verbalizations, whereas TG/2 sticks to production rules based on a context-free backbone that license local trees in a derivation.

The RAGS experiment showed that comparing TG/2 with in-depth NLG systems proves difficult. TG/2 remains shallow in that it does not support complex interrelated NLG tasks such as lexical choice, aggregation, or the generation of referring expressions.

Future applications of TG/2 are geared towards presenting personalized summaries about multilingual results of question answering, generating meaningful and consistent annotations of objects in the process of modeling software, and producing user manuals for technical devices in multiple languages.

Though (or because) it has matured for more than a decade, TG/2 is alive and kicking.[6]

Acknowledgments

This work was partially supported by a research grant from the German *Bundesministerium für Bildung und Forschung* to the project COLLATE-II (contract 01 IN C02). I am indebted to many people who have contributed to the different projects. Space limitations permit only to list those who implemented and/or documented major parts of the TG/2 and eGram systems, or of the grammars: Ana Água, Tim vor der Brück, Matthias Großkloß, Eelco Mossel, Matthias Rinck, Joachim

[6] TG/2 has been licensed to more than 30 sites for commercial, research and educational purposes. TG/2 and eGram are available from DFKI GmbH. The licensed software includes a user manual for TG/2 and a guidebook for writing grammars in eGram. XtraGen can be licensed from XtraMind GmbH.

Sauer, Holger Stenzhorn and Michael Wein. Special thanks go to Sabine Geldof, who was a patient and inspiring user of TG/2 during the COMRIS project. Her feedback helped making TG/2 usable.

References

[Becker and Busemann, 1999] Tilman Becker and Stephan Busemann, editors. *May I Speak Freely? Between Templates and Free Choice in Natural Language Generation. Workshop at the 23rd German Annual Conference for Artificial Intelligence (KI '99). Proceedings*, Document D-99-01, 1999.

[Busemann and Horacek, 1998] Stephan Busemann and Helmut Horacek. A flexible shallow approach to text generation. In Eduard Hovy, editor, *Nineth International Natural Language Generation Workshop. Proceedings*, pages 238–247, Niagara-on-the-Lake, Canada, 1998.

[Busemann *et al.*, 1994] Stephan Busemann, Stephan Oepen, Elizabeth Hinkelman, Günter Neumann, and Hans Uszkoreit. COSMA–multi-participant NL interaction for appointment scheduling. Technical Report RR-94-34, DFKI, Saarbrücken, 1994.

[Busemann, 1996] Stephan Busemann. Best-first surface realization. In Donia Scott, editor, *Eighth International Natural Language Generation Workshop. Proceedings*, pages 101–110, Herstmonceux, Univ. of Brighton, England, 1996.

[Busemann, 1998] Stephan Busemann. A shallow formalism for defining personalized text. In *Proceedings of Workshop Professionelle Erstellung von Papier- und Online-Dokumenten: Perspektiven für die automatische Textgenerierung. 22nd Annual German Conference on Artificial Intelligence (KI '98)*, Bremen, Germany, 1998.

[Busemann, 1999] Stephan Busemann. Constraint-based techniques for interfacing software modules. In Chris Mellish and Donia Scott, editors, *Proc. of the AISB'99 Workshop on Reference Architectures and Data Standards for NLP*, pages 48–54, University of Edinburgh, Scotland, April 1999. The Society for the Study of Artificial Intelligence and Simulation of Behgaviour.

[Busemann, 2002] Stephan Busemann. Language generation for cross-lingual document summarisation. In Huanye Sheng, editor, *International Workshop on Innovative Language Technology and Chinese Information Processing (ILT&CIP-2001), April 6-7, 2001, Shanghai, China*, Beijing, China, 2002. Science Press, Chinese Academy of Sciences.

[Busemann, 2004] Stephan Busemann. eGram – a grammar development environment and its usage for language generation. In *Proc. Fourth International Conference on Language Resources and Evaluation (LREC)*, Lisbon, Portugal, May 2004.

[Davis and King, 1977] Randall Davis and Jonathan King. An overview of production systems. In E. W. Elcock and D. Michie, editors, *Machine Intelligence 8*, pages 300–332. Ellis Horwood, Chichester, 1977.

[Dudenredaktion, 1998] Die Dudenredaktion. *Duden. Die Grammatik. Grammatik der deutschen Gegenwartssprache*, volume 4 of *Duden - Das Standardwerk zur deutschen Sprache*. Dudenverlag, Mannheim - Wien - Zürich, 6. edition, 1998.

[Finkler and Neumann, 1988] Wolfgang Finkler and Günter Neumann. Morphix: A fast realization of a classification–based approach to morphology. In H. Trost, editor, *Proceedings der 4. Österreichischen Artificial–Intelligence Tagung, Wiener Workshop Wissensbasierte Sprachverarbeitung*, pages 11–19, Berlin, August 1988. Springer.

[Gazdar *et al.*, 1985] Gerald Gazdar, Ewan Klein, Geoffrey Pullum, and Ivan Sag. *Generalized Phrase Structure Grammar*. Basil Blackwell, London, 1985.

[Geldof, 1999] Sabine Geldof. Templates for wearables in context. In Becker and Busemann [1999], pages 48–51.

[Lenci *et al.*, 2002] Alessandro Lenci, Ana Água, Roberto Bartolini, Stephan Busemann, Nicoletta Calzolari, Emmanuel Cartier, Karine Chevreau, and José Coch. Multilingual summarization by integrating linguistic resources in the MLIS-MUSI project. In *Proc. Third International Conference on Language Resources and Evaluation (LREC)*, pages 1464–1471, Las Palmas, Canary Islands, Spain, May 2002.

[McRoy *et al.*, 2003] Susan W. McRoy, Songsak Channarukul, and Syed S. Ali. An augmented template-based approach to text realization. *Natural Language Engineering*, 9(4):381–420, 2003.

[Mellish *et al.*, 2000] Chris Mellish, Roger Evans, Lynne Cahill, Christy Doran, Daniel Paiva, Mike Reape, Donia Scott, and Neil Tipper. A representation for complex and evolving data dependencies in generation. In *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP-NAACL)*, pages 119–126, Seattle, Washington, USA, 2000.

[Reiter *et al.*, 1997] Ehud Reiter, Alison Cawsey, Liesl Osman, and Yvonne Roff. Knowledge acquisition for content selection. In *Proceedings of the 6th European Workshop on Natural Language Generation (ENLGWS-97)*, pages 117–126, Duisburg, 1997.

[Rinck, 2003] Matthias Rinck. Ein Metaregelformalismus für TG/2. Master's thesis, Department for Computational Linguistics, University of the Saarland, 2003.

[Stenzhorn, 2002] Holger Stenzhorn. XtraGen. A natural language generation system using Java and XML technologies. In *Proceedings of the Second Workshop on NLP and XML*, Taipeh, Taiwan, 2002.

[Xu *et al.*, 2000] Feiyu Xu, Klaus Netter, and Holger Stenzhorn. MIETTA - a framework for uniform and multilingual access to structured database and web information. In *Proceedings of the 5th International Workshop on Information Retrieval with Asian Languages (IRAL'00)*, Hong Kong, 2000.