JYAG & IDEY: A Template-Based Generator and Its Authoring Tool*

Songsak Channarukul and **Susan W. McRoy** and **Syed S. Ali**
{*songsak, mcroy, syali*}*@uwm.edu*

Natural Language and Knowledge Representation Research Group
Electrical Engineering and Computer Science Department
University of Wisconsin-Milwaukee

JYAG (**J**ava 2.0 Platform **YAG**) is the Java implementation of a real-time, general-purpose, template-based generation system (YAG, **Y**et **A**nother **G**enerator) (Channarukul 1999; McRoy, Channarukul, & Ali 2000). JYAG enables interactive applications to adapt natural language output to the interactive context without requiring developers to write all possible output strings ahead of time or to embed extensive knowledge of the grammar of the target language in the application. Currently, designers of interactive systems who might wish to include dynamically generated text face a number of barriers; for example designers must decide (1) How hard will it be to link the application to the generator? (2) Will the generator be fast enough? (3) How much linguistic information will the application need to provide in order to get reasonable quality output? (4) How much effort will be required to write a generation grammar that covers all the potential outputs of the application? The design and implementation of our template-based generation system, JYAG, is intended to address each of these concerns.

A template-based approach to text realization requires an application developer to define templates to be used at generation time; therefore, the tasks of designing, testing, and maintaining templates are inevitable. JYAG provides a set of pre-defined templates. Developers may also define their own templates to fit the requirements of a domain-specific application. Those templates might be totally new or they can be a variation of existing templates.

Even though developers can author a template by manually editing its textual definition in a text file (in YAG's declarative format or XML), it is more convenient and efficient if they can perform such tasks in a graphical, integrated development environment. A developer might have to spend a substantial amount of time dealing with syntax familiarization, authoring templates, testing their natural language output, and managing them. IDEY (**I**ntegrated **D**evelopment **E**nvironment for **Y**AG) provides these services as a tool for JYAG's templates authoring, testing, and managing. IDEY's graphical interface reduces the amount of time needed for syntax familiarization through direct manipulation and template visualization. It also allows a developer to test newly constructed templates easily. The interface helps prevent errors by constraining the way in which templates may be constructed or modified. For example, values of slots in templates are constrained by context-sensitive pop-up menu choices.

In addition, JYAG and IDEY offer the following benefits to applications and application designers:

Speed: JYAG has been designed to work in real-time. The JYAG template processing engine does not use search to realize text, thus the speed of generation depends on the complexity of the template that the application selects, not on the size of the grammar. Short, simple texts are always realized faster than longer ones. (In many other approaches, speed is a function of the grammar size, because it is searched during realization (Elhadad 1992; 1993; Mann 1983; McKeown 1982; 1985).)

Robustness: In JYAG, the realization of a template cannot fail. Even if there are inconsistencies in its input (such as subject-verb disagreement), the generator will produce an understandable (if not grammatical) output. Applications that need to enforce grammaticality can use the JYAG preprocessor to detect missing or conflicting features and to supply acceptable values. The preprocessor makes use of a declarative specification of slot constraints, based on an attribute grammar (Channarukul, McRoy, & Ali 2000). This specification is modifiable and extensible by the application designer.

Expressiveness: JYAG offers an expressive language for specifying a generation grammar. This language can express units as small as a word or as large as a document equally well. Unlike the typical template-based approach, the values used to instantiate slots are not limited to simple strings, but can include a variety of structures, including conditional expressions or references to other templates. Any declarative grammar, such as one based on feature structures, would be expressible in JYAG.

Coverage: The coverage of JYAG depends on the number of templates that have been defined in its specification language. In theory, any sentence may be realized given an appropriate template. In practice, an application builder must be concerned with whether it is possible to re-use existing templates or whether it is necessary to create new

*This work has been supported by the National Science Foundation, under grants IRI-9701617 and IRI-9523666, and by Intel Corporation.

ones. JYAG simplifies the task of specifying a generation grammar in several ways:

- It provides an expressive, declarative language for specifying templates. This language supports template re-use by allowing template slots to be filled by other templates.
- It includes a general-purpose, template-based grammar for a core fragment of English. These templates include default values for many of the slots, so an application may omit a feature if it has no information about it. Currently, the JYAG distribution includes about 30 domain-independent syntactic templates, along with some semantic templates.
- As mentioned, IDEY helps people edit templates and see what text would be realized from a template, given a set of values for its slots.

Easy deployment: IDEY collects all resources necessary for text realization (such as templates, lexicons, morphology functions), and saves them into a single file. This file contains an instantiation of the JYAG's container class called Generator (using Java's serialization technique). Applications only need to add a few lines of code to create a Generator object and load it from the saved file. To generate a text, applications create an input as a feature structure (an object of the FeatureStructure class), and pass it as an argument to a generator.

JYAG and IDEY are 100% Java implementation, therefore they can run on virtually any platform that supports Java. Its original implementation (YAG, implemented in CLISP) runs on both Linux and Windows 95/98. More details can be found in (Channarukul 1999; McRoy, Channarukul, & Ali 2000; Channarukul, McRoy, & Ali 2001). YAG is a part of our ongoing research on intelligent dialog systems that collaborate with users (McRoy *et al.* 1999).

References

Channarukul, S.; McRoy, S. W.; and Ali, S. S. 2000. Enriching Partially-Specified Representations for Text Realization using An Attribute Grammar. In *Proceedings of The First International Natural Language Generation Conference*.

Channarukul, S.; McRoy, S. W.; and Ali, S. S. 2001. YAG: A Template-Based Text Realization System for Dialog. *The Internation Journal of Uncertainty, Fuzziness, and Knowledge-based Systems*. Forthcoming.

Channarukul, S. 1999. YAG: A Natural Language Generator for Real-Time Systems. Master's thesis, University of Wisconsin-Milwaukee.

Elhadad, M. 1992. *Using argumentation to control lexical choice: A functional unification-based approach*. Ph.D. Dissertation, Computer Science Department, Columbia University.

Elhadad, M. 1993. FUF: The universal unifier - user manual, version 5.2. Technical Report CUCS-038-91, Columbia University.

Grosz, B. J.; Sparck-Jones, K.; and Webber, B. L. 1986. *Readings in Natural Language Processing*. Los Altos, CA: Morgan Kaufmann Publishers.

Mann, W. C. 1983. An overview of the Penman text generation system. In *Proceedings of the Third National Conference on Artificial Intelligence (AAAI-83)*, 261–265. Also appears as USC/Information Sciences Institute Tech Report RR-83-114.

McKeown, K. R. 1982. The TEXT system for natural language generation : An overview. In *Proceedings of the 20th Annual Meeting of the ACL*, 113–120.

McKeown, K. R. 1985. Discourse strategies for generating natural-language text. *Artificial Intelligence* 27(1):1–42. Also appears in (Grosz, Sparck-Jones, & Webber 1986), pages 479-499.

McRoy, S. W.; Ali, S. S.; Restificar, A.; and Channarukul, S. 1999. Building intelligent dialog systems. *intelligence: New Visions of AI in Practice* 10(1):14–23. The Association of Computing Machinery.

McRoy, S. W.; Channarukul, S.; and Ali, S. S. 2000. Text Realization for Dialog. In *Proceedings of the 2000 International Conference on Intelligent Technologies*. Also appears in *Building Dialogue Systems for Tutorial Application*. Technical Report FS-00-01, American Association for Artificial Intelligence, North Falmouth, Massachusetts.