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

1 Overview

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 et al., 2000). JYAG enables interactive applications to adapt natural language output to the 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.

2 An Overview of YAG

YAG (Yet Another Generator) (Channarukul, 1999; McRoy et al., 2000; Channarukul et al., 2001) is a template-based text-realization system that generates text in real-time. YAG uses templates to express text structures corresponding to fragments of the target language. Templates in YAG are declarative and modular. Complex texts can be generated by embedding templates inside other templates.

YAG has a layered architecture as shown in Figure 1. This architecture allows an application to realize texts from two kinds of input, a knowledge representation or a feature structure. In addition, we separate the knowledge sources used by YAG from the "Core YAG", which includes the processes that retrieve and evaluate templates. The knowledge sources include a specification for mapping expressions from the knowledge representation language onto an associated template, a collection of templates that have been organized into libraries, and a lexicon.

2.1 The Core YAG

The **Knowledge Representation Realizer** realizes a knowledge representation into an appropriate feature structure. Its input contains two parts: a semantic network that represents content, and a set of control features that provides supporting information and optional syntactic constraints. Some of these control features are used by YAG to select the appropriate template, the remainder are used to select options within a template.

The **Feature Structure Realizer** realizes a feature structure into a surface text. This feature structure specifies the template to be used along with other features and their values. In addition, this layer will use defaults to specify any missing values in a feature structure input.

The **Template Manager** stores a template into a specified library, and retrieves template structures for the Knowledge Representation Realizer and the Feature Structure Realizer.

2.2 The Knowledge Base

There are two types of knowledge bases in YAG; *domain dependent* and *domain independent*. The domain dependent knowledge base includes the *Mapping Table* and *Domain Template Libraries*. The domain independent knowledge base includes the *Syntactic Template Library* and the *Lexicon*.

The **Mapping Table** is used to map a knowledge representation to its associated template. The Knowledge Representation Realizer accesses a mapping table with selected control features to pick an appropriate template when realizing a text from a knowledge representation. Each mapping entry provides a declarative specification for constructing a

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

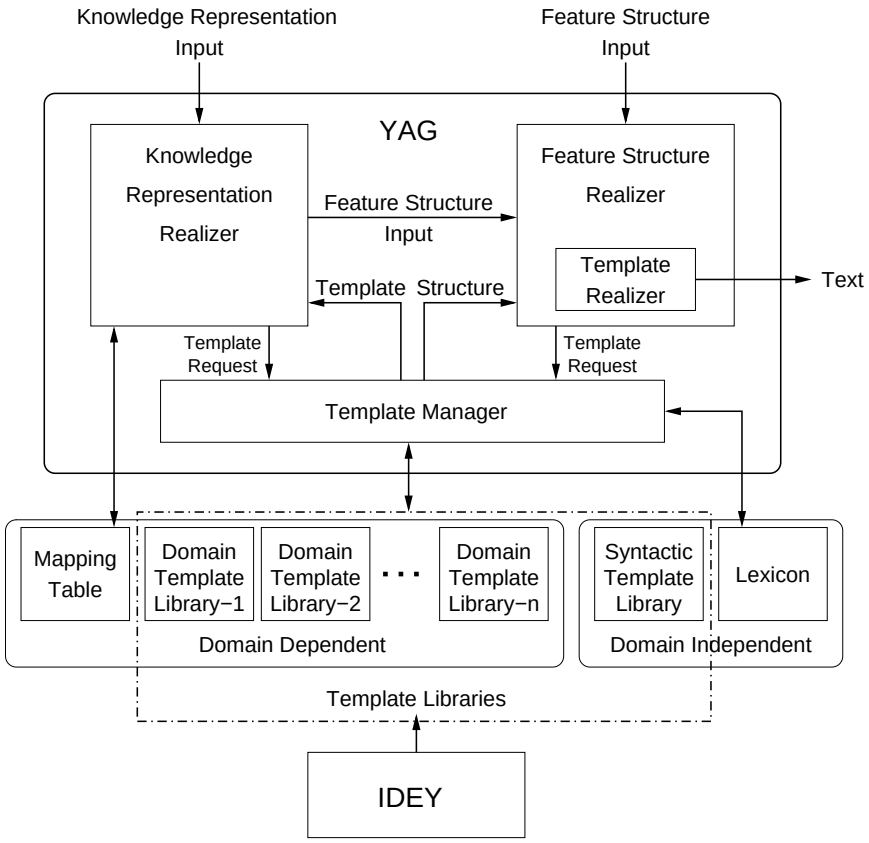

Figure 1: Architecture of YAG

feature structure from the propositions and control features.

Creating the mapping table is the primary task in constructing a new knowledge representation realization component for a given knowledge representation framework.

The **Domain Template Libraries** contain templates that are specific to a particular application. Developers can author their own templates when necessary by manually editing a text file that contains templates in any text editors, or by using the template authoring tool (IDEY, see Section 4).

The **Syntactic Template Library** contains templates that are used as a grammar of English, such as the `CLAUSE`, `NOUN-PHRASE`, and `PRONOUN` templates. Other templates can embed these templates to form more complex structures. They can also be combined with templates from the Domain Template Library.

The **Lexicon** contains word level information. Templates can access the lexicon directly with a template rule. YAG includes morphological functions to inflect a given verb according to verb features (*e.g.,* tense, person, and aspect), and to generate the singular or plural form of a noun. Additional functions can be added, if required.

3 The JYAG API

The JYAG package (jyag) provides an application program interface (API) for applications to use. It currently supports only feature structure realization. The significant API components comprise:

- Generator is the main service class that contains all resources necessary for text realization. It allows applications to generate texts through its realize() functions. Other services include addition and retrieval of templates, lexicons, morphology functions, etc.; conversion of templates from both YAG's original declarative format and XML to Java object (Template class) and vice versa.
- FeatureStructure is an input structure required by JYAG to generate an intended text. It specifies the template to be used and optionally the template library (a *default* template library will be used if none is specified). Semantic information as well as syntactic constraints can be specified in a feature structure seamlessly. Possible value types for each feature are FeatureString, FeatureConstant, FeatureDirectSlot, FeatureIndirectSlot, and FeatureNil. In addition, a feature structure can also embed another feature struc-

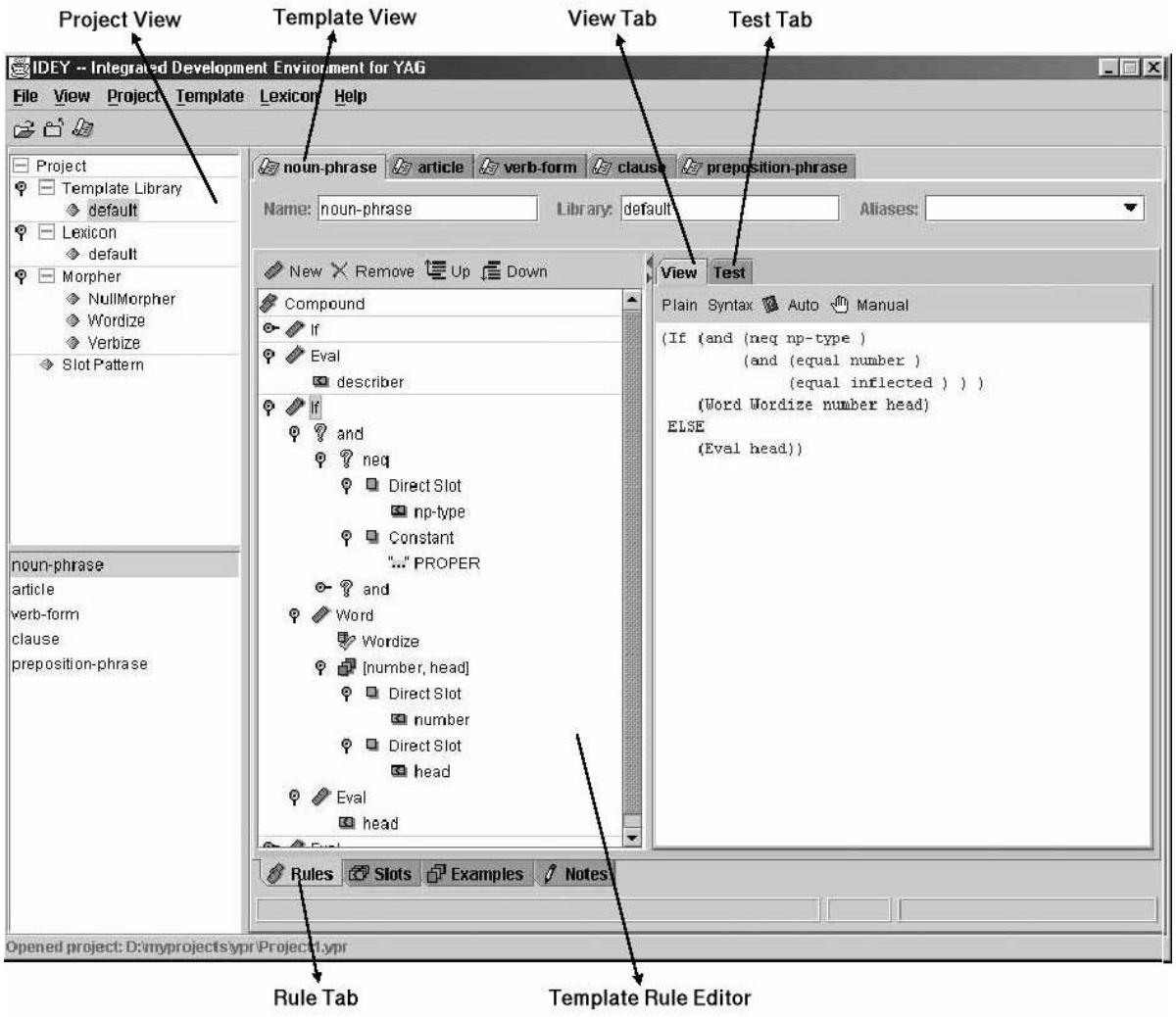

Figure 2: A screenshot of IDEY

ture as a value of its features. This allows applications to generate a more complicated text using several templates at the same time.

- Template is composed of two main parts: template slots (Slot) and template rules (Rule). Template slots are parameters or variables that applications can fill with values. Template rules express how to realize a text given such template slots. A template is realized by instantiating its slots using values specified in a feature structure input. These slots will be accessed when template rules are realized. An attribute grammar portion (AttributeGrammar) of a template is optional. It will be used by JYAG's preprocessor to resolve under-specified inputs and inconsistencies (Channarukul et al., 2000).

- Rule is an abstract class inherited by all template rules (based on the original implementation of YAG). They are StringRule, EvaluationRule, IfRule, TemplateRule, ConditionRule, ConcatenationRule, InsertionRule, AlternationRule, PunctuationRule, and WordRule. A set of template rules can be stored in CompoundRule.

4 IDEY

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. A screenshot of IDEY is given in Figure 2. For legibility we have expanded components of IDEY (Figure 2) in the remaining discussion (Figure 3,4,6, and 7).

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. We will now examine the different parts of the IDEY interface.

4.1 Project View

The top left section of the IDEY interface (Figure 2) is a *Project View*. It shows resources currently included in a project. These resources are template libraries, templates, lexicons, morphology functions, etc. (Figure 3) When IDEY creates a new project, some default units such as a default template library, morphology functions, and lexicon will be created. Other resources can be added to a project at any time. Since IDEY can also save these resources separately, they can be reused across projects of different domains. For example, the morphology funtions and lexicon can be easily reused.

The bottom left section of the IDEY interface shows the corresponding contents inside the selected item on the project view. For example, as illustrated in Figure 3, the default template library is selected, therefore, the lower section of the project view displays a list of templates residing in that library (Figure 3 is an abbreviated version of the default template library).

4.2 Template View

When a user creates a new template or selects an existing template for authoring and testing. The *Template View* is shown on the *Workplace* area (the right section of IDEY), see Figure 2. This section contains several panels used for specification of template components (such as template rules and template slots).

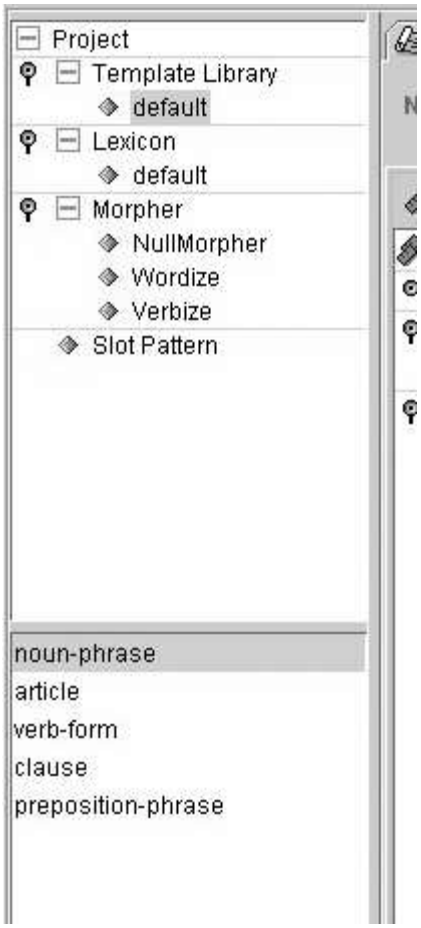

Figure 3: The Project View

4.2.1 Authoring

Template authoring is mainly focused on defining template rules. Template rules are shown on the left part of the *Rule* tab (the lower left of Figure 4, inside the template view). A hierarchical interface (tree) is used for visualization and navigation of template rules (Figure 4). Each template rule can be moved around and deleted using buttons placed on top of this tab. A user can modify a template rule by direct manipulation. IDEY is context-sensitive in that it shows possible and valid options for certain locations such as template slot choices bound by the selected template (as illustrated in Figure 5).

The other side of the rule tab, the *View* tab, shows corresponding template rules in a declarative format (Figure 6). Whenever a template rule is modified, IDEY updates the information shown instantly. Sometimes, large template rules are unavoidable. IDEY decreases information overload on its users by allowing them to see a portion of template rules at a time. On the tree, they can hide template rules that out of focus, and show just parts that are of interest. When selecting a template rule, the view tab shows

Figure 4: The Template Rule Editor

only a declarative format of that rule. Choosing the root of a tree results in a whole template rule being displayed.

4.2.2 Testing

The other tab on the right part of the Rule tab is the *Test* tab (Figure 7). Its bottom section allows a user to enter values into a feature structure associated with the template being edited. This interface is a combination of a tree and a table; where a tree shows a feature structure hierarchically (when it embeds other feature structures) and a table allows a user to enter feature values. When an input is ready, the user can click on the *Realize* button to see what would be the generated text given such an input and template rule.

5 Deployment

IDEY collects all resources necessary for text realization (such as templates, lexicons, morphology functions), and saves them into a single file. This file keeps an instantiation of the JYAG's container class called Generator (using Java's serialization technique). Applications only need to add a few line of codes to create a Generator object and load it from

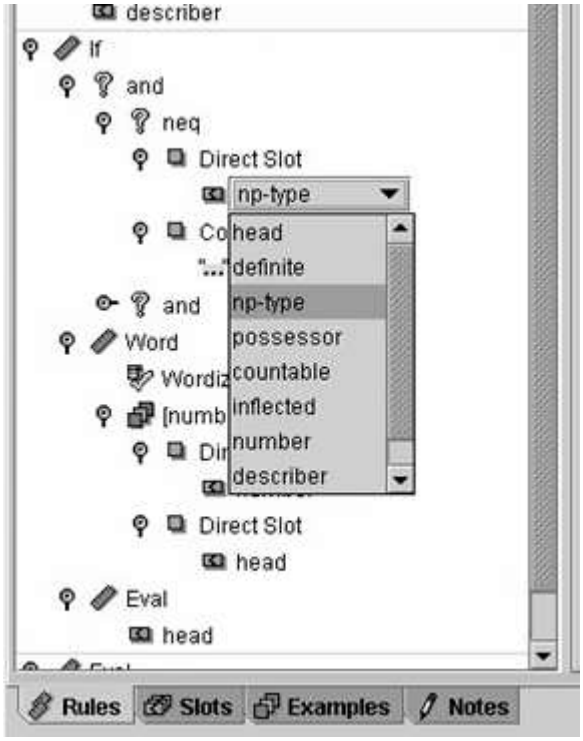

Figure 5: Context-sensitive Feature in IDEY

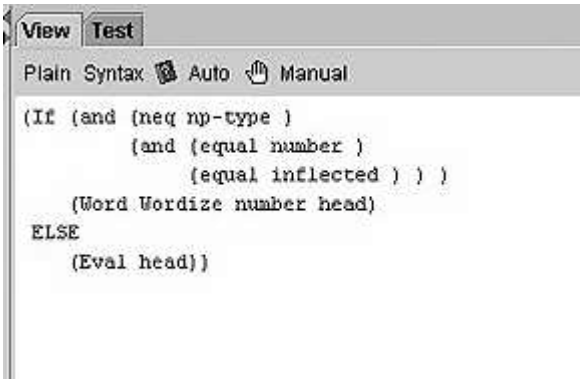

Figure 6: The Rule View

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.

5.1 Creating a generator

An application can create a generator object in two ways. First, it can create a default generator by using the following code:

```
Generator g = new Generator();
```

This will create a generator with several default resources built-in (such as an empty template library named `default`). The other way is to specify

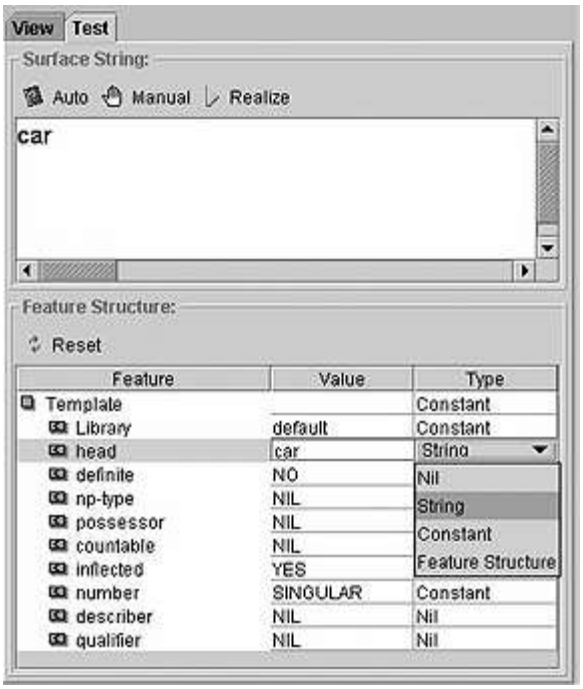

Figure 7: The Test Tab

a file containing a desired generator (created using IDEY). The code is as follows:

```
Generator g = new Generator(
            new File("D:mygenerator.ypr"));
```

5.2 Creating a feature structure

The process of creating a feature structure as an input to a generator is two-fold. First, a new feature structure must be created. Then a set of feature values is created and added into a feature structure. An example is as follows:

```
FeatureStructure fs =
            new FeatureStructure();
FeatureString f1 = new
   FeatureString("Template","noun-phrase");
FeatureString f2 = new
   FeatureString("head","car");

fs.add(f1);
fs.add(f2);
```

5.3 Realizing a text

An application can realize a constructed feature structure by passing it to a generator. The code is:

```
String result = g.realize(fs);
```

6 Summary

JYAG is a text generation tool implemented in Java. It provides API for applications to generate natural language texts. IDEY is a graphical, integrated development environment that allows application programmers to author and test their templates with ease. The NLKRRG web page provides more information on JYAG and IDEY. Its URL is `http://tigger.cs.uwm.edu/~nlkrrg`.

References

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

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

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

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