A Flexible Interface for Linking Applications to Penman's Sentence Generator

Robert T. Kasper
USC/Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292 U.S.A.

Abstract

The Penman text generation system has been used within several different experimental application domains, demonstrating that it provides the basis for an adaptable general purpose text generation capability. Linking with these applications also indicated several ways that Penman's interface with applications could be improved. Penman's interface with applications is described, focusing on SPL, a newly developed sentence plan language. SPL is a notation that can be used by text planning programs to specify plans for sentences at multiple levels of abstraction and varied amounts of detail. Sentence plans are interpreted with respect to a collection of predefined knowledge sources, thereby minimizing the size and complexity of inputs that must be dynamically constructed by the application to generate individual sentences.

1 Introduction

Penman is designed as a domain-independent text generator that can be installed in an application system to generate text for the application on demand. Motivating this design is the hypothesis that a significant amount of knowledge about language and how it is produced can be re-used in different application domains. Assuming that this hypothesis is correct, the construction of an effective interface between a general-purpose text generator and application programs is a significant problem to be solved in text generation research. Penman's interface with applications is the subject of this report, focusing on how it has been designed to achieve a high degree of flexibility.

In designing Penman's interface, we faced the following dilemma: how to make the system easy to use by an application programmer without compromising its expressive power. Penman's grammar can control several hundred different (semantic) features. If the application program were required to specify values for all of these features for every sentence to be generated, the system would be too complex for most practical purposes. If multiple texts are to be generated for the application, the interface can be made more efficient by factoring it into two components:

1. *preparatory knowledge sources*: making knowledge of the application domain available to the generator;
2. *demands for text*: inputs specifying information to be expressed in each particular sentence.

By installing various preparatory knowledge sources for each application domain, the size and complexity of inputs that must be dynamically constructed to generate individual sentences can be minimized.

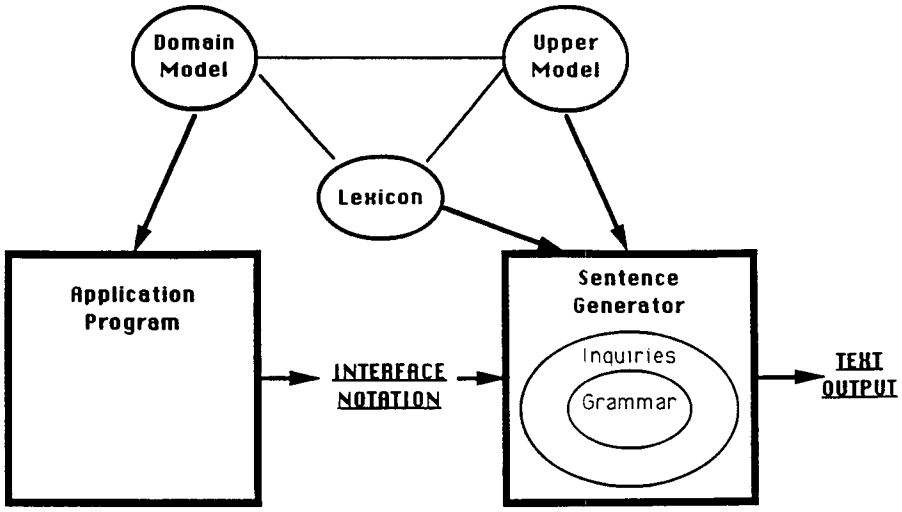

Figure 1: Linking an Application to Penman.

2 Basic Design of the Interface

The principal components and sources of information used by Penman are shown in Figure 1. This diagram shows only those components that are necessary to generate one sentence at a time, although the same design has been augmented to generate coherent paragraphs. We describe first the components that are provided by Penman, and comprise its knowledge about the (English) language. Then, we describe those components whose contents depend on the application.

2.1 Resources Provided by Penman

- **Grammar:** Penman's sentence generator is based on Nigel (Mann & Matthiessen 83), a large systemic-functional grammar of English based primarily on the work of Halliday (Halliday 85). Nigel is a network of interdependent points of minimal grammatical contrast, called *systems*. Each system defines a collection of alternatives, called *grammatical features*, that specify a particular aspect of a sentence (e.g., active or passive). The semantics of the Nigel grammar is defined by a set of *inquiries* that control choices of grammatical features by acquiring information from the knowledge sources in Penman's operating environment.

- **Basic Lexicon:** The basic lexicon provided by Penman contains the definitions of almost all closed-class words (e.g., prepositions, auxiliary verbs), and other frequently used words (approximately 900 root words). The organization of Penman's lexicon is described by (Cumming & Albano 86).

- **Upper Model:** Penman must be able to treat the symbols given in demands for text appropriately (for example, actions are typically expressed as verbs and objects as nouns). Therefore these symbols must have some definition in terms of Penman's taxonomy of knowledge. This taxonomy is called the *upper model*, and it contains abstract categories that reflect grammatical distinctions made in English. The upper model is maintained as a property-inheritance network, using the LOOM knowledge representation system (MacGregor & Bates 87); it has also been encoded in several other frameworks.

2.2 Resources Provided by Specific Applications

These are the preparatory knowledge sources that identify those expressive options in English vocabulary and grammar that can be used to represent information from a particular application domain.

- **Lexical Item Definitions:** The lexicon should contain definitions of any words (with the exception of proper names) that will be generated from the vocabulary of the application domain. Penman

provides a programmed tool (Cumming & Albano 86) to help application developers define words with appropriate grammatical features so that they are under the full control of Penman's grammar.

- **Domain Model:** The *domain model* is a taxonomy of knowledge that is specific to the application domain. In general, the categories of the domain model are more specific than those of the upper model. When a category of some application domain is defined to be more specific than an upper model category, Penman is able to make inferences concerning how the application category might be expressed in English. Most applications that are sophisticated enough to need a text generator require such a model as a natural part of their work, either explicitly or implicitly (e.g., the field types and the relations among them in relational data bases).
  - **Links to the upper model:** All domain model concepts and relations used in demands for text by the application must specialize some concept or relation in Penman's upper model.
  - **Links to the lexicon:** Any number of lexical items may be associated with a domain model concept, and these items will be used by Penman when they have features that do not conflict with other constraints imposed by the sentence plan or grammar. It is not necessary to provide lexical associations for every concept of the domain model, but concepts without any lexical associations are only be expressible using words that are associated with more general concepts.

3 Experience with the Basic Design

Penman was used during 1988 within several projects at the Information Sciences Institute as an experimental English text generator in the following different application domains:

- Navy Pacific Fleet Briefing (CINCPAC): used by the *Integrated Interfaces* project (Arens et al. 88) to report facts from a database about the positions and activities of ships;
- Digital Circuit Diagnosis (DCD): used by the *Explainable Expert Systems* project (Swartout 83) to describe electronic components and actions suggested by an expert system to repair them;
- Program Enhancement Advisor (PEA): used by the *Explainable Expert Systems* project to describe potential improvements to computer programs;
- German-English Machine Translation: used in cooperation with the *Eurotra-D* project (Bateman et al. 89) to produce English translations of German text.

These different applications demonstrated that Penman provides the basis for an adaptable general-purpose text generation capability. They also indicated several ways that Penman's interface with applications could be improved. In a case study of linking the DCD application to Penman, it was found that substantial effort was spent on two tasks:

1. subordinating the application domain model to Penman's upper model;
2. constructing input specifications for Penman's sentence generator.

The first task was difficult mostly because application programmers are likely to be unfamiliar with the upper model. To overcome this difficulty, descriptions of the upper model hierarchy have been encoded in an *upper model construction tool.* This tool traverses the upper model hierarchy, under guidance of the application programmer, to place domain model concepts under appropriate upper model concepts.

The second task was difficult because Penman's former input notation (see Sondheimer & Nebel 86) proved to be cumbersome, requiring all information to be stated in a form similar to the predicate calculus. Although it provided a strong foundation for formal reasoning, the predicate calculus style of notation was relatively inflexible. Some linguistic constraints could not be stated directly, and other kinds of information had to be stated redundantly.

Hence we developed SPL, a new interface notation with the following characteristics:

- constraints can be stated at multiple levels of abstraction: both propositional content and linguistic features can be selectively controlled;
- more information can be predefined, including linguistic features that rarely vary in the application domain (e.g., default to present tense), and frequently used clusters of information (e.g., how to refer to an object by a proper name);
- constraints can be separately specified on different occurrences of an entity.

4 SPL: A New Sentence Plan Notation

SPL representations are lists of terms describing the types of entities and the particular attributes of those entities to be expressed in English. The attributes of SPL terms provide control at several levels of abstraction. At the two most basic levels, attributes may specify semantic relations to be expressed from the application's knowledge base, or they may directly specify responses to Penman's inquiries, which determine grammatical features of sentences.

The syntax of the SPL notation is defined in Figure 2, using BNF-style productions (note that A^* denotes zero or more occurrences of A, and A^+ denotes one or more occurrence of A). The SPL notation is formally similar to the typed feature logic developed by Smolka (Smolka 88) in a similar context of sharing information between language processing and knowledge representation systems. The formal properties of this kind of notation provide a sound method for merging partial descriptions together into more complete descriptions, thus making it straightforward to merge predefined information with information contained in the specification for a particular sentence.

Plan	→	Term$^+$	Type	→	ConceptName | (ConceptName$^+$)
Term	→	(Variable / Type Attribute*) |	Attribute	→	Keyword Term
		Variable |			
		Constant |	Keyword	→	RelationName |
		(Term$^+$) |			MacroName |
		(:and Term$^+$) |			InquiryName |
		(:or Term$^+$)			InquiryName (Variable$^+$) |
					SpecialKeyword

Figure 2: Syntax of the SPL notation.

4.1 A Simple Example

```
(e1 / enroute
    :actor (s2 / ship  :name kennedy)
    :destination.r (p3 / port  :name san-diego)
    :ebeg.r (d4 / date  :day.r 20  :month.r 2)
    :theme d4
    :tense past)
```

Figure 3: A sentence plan for: *On 2/20 the Kennedy was en route to San Diego.*

A simple example of the SPL notation is shown in Figure 3. The main term of this plan informs Penman that it should generate a sentence to express `e1`, which is a variable denoting an entity in the application's knowledge base. The type of `e1` is `enroute`, the name of a concept in the domain model that specializes the

Material-Process concept (i.e., a kind of action) of Penman's upper model. Penman uses this information about the type of `e1` to choose an appropriate verb, *be en route*, for the sentence.

The term describing `e1` also contains five attributes. The interpretation of attributes depends on the type of their keywords. The keywords `:actor`, `:destination.r` and `:ebeg.r` are the names of relations in the domain model. These three attributes inform Penman that the action `e1` has an actor denoted by `s2`, a destination denoted by `p3`, and a time denoted by `d4`. `:theme` is a special keyword that may optionally be used to control thematization. In this case, its value specifies that the phrase referring to `d4` should come at the front of the generated sentence. The keyword `:tense` is the name of a macro which expands the value `past` into a collection of attributes that specify responses to some of Penman's inquiries, as described below.

4.2 Macros

The SPL notation provides macros to allow predefinition of frequently used clusters of information. Penman's grammar can control a large number of grammatical features when it builds sentences. In many cases, the grammar is capable of expressing far more delicate shades of meaning than a particular application may require. Rather than require that the application repeatedly specify all the necessary inquiry responses to generate some grammatical phenomenon, Penman makes it possible to use macros to abbreviate the specification at a level of detail that is supported by the application.

For example, in order to specify English tense in a fully general way, one must specify ordering relations between three times: the actual speaking time, the event time, and the time of reference with which the event is contrasted. For many applications, such delicate control of temporal relations is not required; some distinguish simply between present and past. For this case, we define a macro called `:tense` that takes the values `present` or `past` and expands them into the appropriate inquiry responses. Penman provides a predefined package of common macro keywords, such as the macro for tense described above. It also provides functions for creating new macros that can be used by an application to customize its interface to Penman.

4.3 Defaults

Often it is useful to be able to predefine features of sentences that do not change frequently within an application domain. To enable this, Penman provides a facility for defining default values for any of the inquiries that it uses to obtain information from an application. Many of Penman's inquiries come supplied with initial default values that will be used unless specific information in a sentence plan overrides them.

For example, consider the sentence plan given in Figure 3. It does not contain any specification of whether the sentence should be a statement, a question, or a command, nor does it contain any specification of whether it should express positive or negative polarity. Penman's predefined default values provide the necessary inquiry responses to generate a statement with positive polarity. In addition to the initial defaults supplied by Penman, functions are provided to enable the application to dynamically define new default values in packages called *default environments*. Default environments are maintained in a stack-like memory, with the Penman-supplied defaults at the bottom. The stack-like organization of default environments makes it possible for an application to temporarily change default settings for a particular portion of a text, and then return to the default environment that was previously in effect.

4.4 Interpretation of Sentence Plans

A sentence plan in the SPL notation is interpreted in two phases. First, the plan is pre-processed and transformed into an internal representation. This pre-processing step includes expansion of macros, distribution of type information to variable terms, and a check of the consistency of terms. The first term of the plan is identified to the sentence generator as the initial unit of information to be expressed (usually as the main clause of a sentence). Then, Penman invokes its sentence generator to produce a sentence according to the expanded plan.

Penman's sentence generator uses a series of inquiries to the sentence plan and other knowledge sources in

order to guide the generation process. Inquiries may obtain answers from several sources, according to the following sequence:

1. **SPL keyword:** The sentence plan is searched for a keyword that matches the name (and, optionally,, the parameters) of the inquiry, and the corresponding value is returned.

2. **knowledge sources:** Each inquiry may have an executable (i.e., lisp) function associated with it, called an *inquiry implementation*, which searches knowledge sources for appropriate information. Inquiry implementations generally obtain information from the domain and upper models about the type or relational attributes of SPL terms.

3. **active default value:** When the inquiry implementation returns an undefined value for the inquiry, or when the inquiry has no implementation, the current active default value for the inquiry is used.

In general, the attributes in a SPL specification that correspond directly to linguistic distinctions, such as inquiry responses, take precedence over other attributes, such as relations from the domain knowledge base. In addition, all attributes contained in the SPL specification for a particular sentence take precedence over any default values that have been defined for an inquiry.

5 Conclusions

Penman's interface with applications has been enhanced by the development of SPL, a new interface notation, and facilities to aid the definition and linking of predefined knowledge sources. These knowledge sources include a model of the application domain, and default attributes that may be modified dynamically by the application.

Because SPL representations may contain linguistic attributes in addition to propositional knowledge, they are able to specify constraints on how something is expressed, when necessary, in addition to specifying what to express. SPL also provides control at varying levels of detail, accommodating partial specifications that may be augmented by merging information from coreferential terms and default attribute values. The flexibility of SPL makes Penman relatively easy to use for simple applications, without limiting the power of a large general purpose grammar.

References

Arens, Y., Miller, L., Shapiro, S.C. and Sondheimer, N.K. Automatic Construction of User-Interface Displays. In *Proceedings of the 7th AAAI Conference*, St. Paul, MN, August 1988.

Bateman, J., Kasper, R., Schütz, J. and Steiner, E. Interfacing an English Text Generator with a German Machine-Translation Analysis. In *Proceedings of the 4th European ACL Conference*, Manchester, England, April 1989.

Cumming, S. and Albano, R. A Guide to Lexical Acquisition in the JANUS System. USC/Information Sciences Institute, Research Report RR-85-162, February 1986.

Halliday, M.A.K. *Introduction to Functional Grammar.* Edward Arnold Press: London, England, 1985.

MacGregor, R. and Bates, R. The Loom Knowledge Representation Language. In *Proceedings of the Knowledge-Based Systems Workshop*, St. Louis, MO, April 1987. Also available as USC/ISI Research Report RS-87-188.

Mann, W.C. and Matthiessen, C.M.I.M. Nigel: A Systemic Grammar for Text Generation. In *Systemic Perspectives on Discourse: Selected Papers Papers from the Ninth International Systemics Workshop*, Benson, R. and Greaves, J. (eds), Ablex: London, England, 1985. Also available as USC/ISI Research Report RR-83-105.

Smolka, G. A Feature Logic with Subsorts. LILOG Report 33, IBM Deutschland, Stuttgart, West Germany, 1988.

Sondheimer, N. K. and Nebel B. A Logical-Form and Knowledge-Base Design for Natural Language Generation. In *AAAI-86: Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, 1986.

Swartout, W. R. XPLAIN: A System for Creating and Explaining Expert Consulting Systems. *Artificial Intelligence*, Vol. 21:3, pp. 285-325, 1983.