A Framework for Customizable Generation of Hypertext Presentations

Benoit Lavoie and **Owen Rambow**
CoGenTex, Inc.
840 Hanshaw Road, Ithaca, NY 14850, USA
benoit, owen@cogentex.com

Abstract

In this paper, we present a framework, PRE-SENTOR, for the development and customization of hypertext presentation generators. PRE-SENTOR offers intuitive and powerful declarative languages specifying the presentation at different levels: macro-planning, micro-planning, realization, and formatting. PRESENTOR is implemented and is portable cross-platform and cross-domain. It has been used with success in several application domains including weather forecasting, object modeling, system description and requirements summarization.

1 Introduction

Presenting information through text and hypertext has become a major area of research and development. Complex systems must often deal with a rapidly growing amount of information. In this context, there is a need for presentation techniques facilitating a rapid development and customization of the presentations according to particular standards or preferences. Typically, the overall task of generating a presentation is decomposed into several subtasks including: *macro-planning* or *text planning* (determining output content and structure), *micro-planning* or *sentence planning* (determining abstract target language resources to express content, such as lexical items and syntactic constructions and aggregating the representations), *realization* (producing the text string) and *formatting* (determining the formatting marks to insert in the text string). Developing an application to present the information for a given domain is often a time-consuming operation requiring the implementation from scratch of domain communication knowledge (Kittredge et al., 1991) required for the different generation subtasks. In this technical note and demo

we present a new presentation framework, PRE-SENTOR, whose main purpose is to facilitate the development of presentation applications. PRE-SENTOR has been used with success in different domains including object model description (Lavoie et al., 1997), weather forecasting (Kittredge and Lavoie, 1998) and system requirements summarization (Ehrhart et al., 1998; Barzilay et al., 1998). PRESENTOR has the following characteristics, which we believe are unique in this combination:

- PRESENTOR modules are implemented in Java and C++. It is therefore easily portable cross-platform.

- PRESENTOR modules use declarative knowledge interpreted at run-time which can be customized by non-programmers without changing the modules.

- PRESENTOR uses rich presentation plans (or *exemplars*) (Rambow et al., 1998) which can be used to specify the presentation at different levels of abstraction (rhetorical, conceptual, syntactic, and surface form) and which can be used for deep or shallow generation.

In Section 2, we describe the overall architecture of PRESENTOR. In Section 3 to Section 6, we present the different specifications used to define domain communication knowledge and linguistic knowledge. Finally, in Section 7, we describe the outlook for PRESENTOR.

2 PRESENTOR Architecture

The architecture of PRESENTOR illustrated in Figure 1 consists of a core generator with several associated knowledge bases. The core generator has a pipeline architecture which is similar to many existing systems (Reiter, 1994): an incoming request is received by the generator interface triggering sequentially the macro-planning, micro-planning, realization and fi-
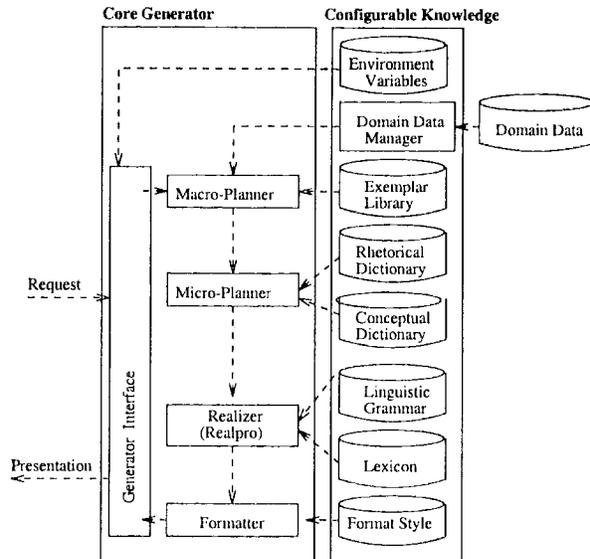
Figure 1: Architecture of PRESENTOR

nally the formatting of a presentation which is then returned by the system. This pipeline architecture minimizes the interdependencies between the different modules facilitating the upgrade of each module with minimal impact on the overall system. It has been proposed that a pipeline architecture is not an adequate model for NLG (Rubinoff, 1992). However, we are not aware of any example from practical applications that could not be implemented with this architecture. One of the innovations of PRESENTOR is in the use of a common presentation structure which facilitates the integration of the processing by the different modules. The macro-planner creates a structure and the other components add to it.

All modules use declarative knowledge bases distinguished from the generator engine. This facilitates the reuse of the framework for new application domains with minimal impact on the modules composing the generator. As a result, PRESENTOR can allow non-programmers to develop their own generator applications. Specifically, PRESENTOR uses the following types of knowledge bases:

• *Environment variables*: an open list of variables with corresponding values used to specify the configuration.

• *Exemplars*: a library of schema-like structures (McKeown, 1985; Rambow and Korelsky, 1992) specifying the presentation to be generated at different levels of abstraction (rhetori-

cal, conceptual, syntactic, surface form).

• *Rhetorical dictionary*: a knowledge base indicating how to realize rhetorical relations linguistically.

• *Conceptual dictionary*: a knowledge base used to map language-independent conceptual structures to language-specific syntactic structures.

• *Linguistic grammar*: transformation rules specifying the transformation of syntactic structures into surface word forms and punctuation marks.

• *Lexicon*: a knowledge base containing the syntactic and morphological attributes of lexemes.

• *Format style*: formatting specifications associated with different elements of the presentation (not yet implemented).

As an example, let us consider a simple case illustrated in Figure 2 taken from a design summarization domain. Hyperlinks integrated in the presentation allow the user to obtain additional generated presentations.

Data Base	Description of FDBMgr
Project **ProjAF-2** System DBSys Site Ramstein Host Gauss Soft FDBMgr Site Syngapour Host Jakarta Soft FDBClt	FDBMgr is a software component which is deployed on host Gauss. FDBMgr runs as is a server and a daemon and is written in C(ANSI) and JAVA.

Figure 2: Presentation Sample

The next sections present the different types of knowledge used by PRESENTOR to define and construct the presentation of Figure 2.

3 Exemplar Library

An exemplar (Rambow et al., 1998; White and Caldwell, 1998) is a type of schema (McKeown, 1985; Rambow and Korelsky, 1992) whose purpose is to determine, for a given presentation request, the general specification of the presentation regarding its macro-structure, its content and its format. One main distinction between the exemplars of PRESENTOR and ordinary schemas is that they integrate conceptual, syntactic and surface form specifications of the content, and can be used for both deep and shallow generation, and combining both generality and simplicity. An exemplar can contain dif-

ferent type of specifications, each of which is optional except for the name of the exemplar:

- *Name*: Specification of the name of the exemplar.

- *Parameters*: Specification of the arguments passed in parameters when the exemplar is called.

- *Conditions of evaluation*: Specification of the conditions under which the exemplar can be evaluated.

- *Data*: Specification of domain data instantiated at run-time.

- *Constituency*: Specification of the presentation constituency by references to other exemplars.

- *Rhetorical dependencies*: Specification of the rhetorical relations between constituents.

- *Features specification*: Open list of features (names and values) associated with an element of presentation. These features can be used in other knowledge bases such as grammar, lexicon, etc.

- *Formatting specification*: Specification of HTML tags associated with the presentation structure constructed from the exemplar.

- *Conceptual content specification*: Specification of content at the conceptual level.

- *Syntactic content specification*: Specification of content at the lexico-syntactic level.

- *Surface form content specification*: Specification of the content (any level of granularity) at the surface level.

- *Documentation*: Documentation of the exemplar for maintenance purposes.

Once defined, exemplars can be clustered into reusable libraries.

Figure 3 illustrates an exemplar, *soft-description*, to generate the textual description of Figure 2, Here, the description for a given object *$SOFT*, referring to a piece of software, is decomposed into seven constituents to introduce a title, two paragraph breaks, and some specifications for the software type, its host(s), its usage(s) and its implementation language(s). In this specification, all the constituents are evaluated. The result of this evaluation creates seven presentation segments added as constituents (daughters) to the current growth point in the presentation structure being generated. Referential identifiers (ref1, ref2, ..., ref4) assigned to some constituents

are also being used to specify a rhetorical relation of elaboration and to specify syntactic conjunction.

```
Exemplar:
[
  Name:  soft-description
  Param: [ $SOFT ]
  Const: [ AND
          [ title ( $SOFT )
            paragraph-break ( )
            object-type ( $SOFT ) : ref1
            soft-host ( $SOFT ) : ref2
            paragraph-break ( )
            soft-usage ( $SOFT ) : ref3
            soft-language ( $SOFT ) : ref4  ]
          ]
  Rhet:  [ ( ref1 R-ELABORATION ref2 )
           ( ref3 CONJUNCTION ref4 )    ]
  Desc:  [ Describe the software ]
]
```

Figure 3: Exemplar for Software Description

Figure 4 illustrates an exemplar specifying the conceptual specification of an object type. The notational convention used in this paper is to represent variables with labels preceded by a $ sign, the concepts are upper case English labels preceded by a # sign, and conceptual relations are lower case English labels preceded by a # sign. In Figure 4 the conceptual content specification is used to built a conceptual tree structure indicating the state concept #HAS-TYPE has as an object $OBJECT which is of type $TYPE. This variable is initialized by a call to the function ikrs.getData($OBJECT #type) defined for the application domain.

```
Exemplar:
[
  Name:    object-type
  Param:   [ $OBJECT ]
  Var:     [ $TYPE = ikrs.getData( $OBJECT #type ) ]
  Concept: [
              #HAS-TYPE (
                          #object $OBJECT
                          #type   $TYPE
                        )
           ]
  Desc:    [ Describe the object type ]
]
```

Figure 4: Exemplar for Object Type

4 Conceptual Dictionary

PRESENTOR uses a conceptual dictionary for the mapping of conceptual domain-specific rep-

resentations to linguistic domain-indepenent representations. This mapping (transition) has the advantage that the modules processing conceptual representations can be unabashedly domain-specific, which is necessary in applications, since a broad-coverage implementation of a domain-independent theory of conceptual representations and their mapping to linguistic representations is still far from being realistic.

Linguistic representations found in the conceptual dictionary are deep-syntactic structures (DSyntSs) which are conform to those that REALPRO (Lavoie and Rambow, 1997), PRESENTOR's sentence realizer, takes as input. The main characteristics of a deep-syntactic structure, inspired in this form by I. Mel'čuk's Meaning-Text Theory (Mel'čuk, 1988), are the following:

- The DSyntS is an unordered dependency *tree* with labeled nodes and labeled arcs.

- The DSyntS is *lexicalized*, meaning that the nodes are labeled with lexemes (uninflected words) from the target language.

- The DSyntS is a *syntactic* representation, meaning that the arcs of the tree are labeled with syntactic relations such as "subject" (represented in DSyntSs as I), rather than conceptual or semantic relations such as "agent".

- The DSyntS is a *deep* syntactic representation, meaning that only meaning-bearing lexemes are represented, and not function words.

Conceptual representations (ConcSs) used by PRESENTOR are inspired by the characteristics of the DSyntSs in the sense that both types of representations are unordered tree structures with labelled arcs specifying the roles (conceptual or syntactic) of each node. However, in a ConcS, concepts are used instead of lexemes, and conceptual relations are used instead of relations. The similairies of the representions for the ConcSs and DSyntSs facilitate their mapping and the sharing of the functions that process them.

Figure 5 illustrates a simple case of lexicalization for the state concept *#HAS-TYPE* introduced in the exemplar defined in Figure 4. If the goal is a sentence, *BE1* is used with *$OBJECT* as its first (I) syntactic actant and *$TYPE* as its second (II). If the goal is a noun phrase, a complex noun phrase is used (e.g., *software component FDBMgr*). The lexicalization can be

controlled by the user by modifying the appropriate lexical entries.

```
Lexicalization-rule:
[
 Concept: #HAS-TYPE
 Cases: [ Case:
        [#HAS-TYPE (#object $OBJ
                    #type    $TYPE)]   | [goal:S]
        <-->
        [ BE1 ( I  $OBJ
                II $TYPE ) ]           | [ ]

        Case:
        [#HAS-TYPE (#object $OBJ
                    #type    $TYPE)]   | [goal:NP]
        <-->
        [ $TYPE ( APPEND $OBJECT ) ]   | [ ]
        ]
]
```

Figure 5: Conceptual Dictionary Entry

5 Rhetorical Dictionary

PRESENTOR uses a rhetorical dictionary to indicate how to express the rhetorical relations connecting clauses using syntax and/or lexical means (cue words). Figure 6 shows a rule used to combine clauses linked by an elaboration relationship. This rule combines clauses *FDBMgr is a software component* and *FDBMgr is deployed on host Gauss* into *FDBMgr is a software component which is deployed on host Gauss.*

```
Rhetorical-rule:
[
 Relation: R-ELABORATION
 Cases:    [
           Case:
           [ R-ELABORATION
             ( nucleus   $V ( I $X II $Y )
               satellite $Z ( I $X ) ]
           <-->
           [ $V ( I $X  II $Y  ( ATTR $Z ) ) ]
           ]
]
```

Figure 6: Rhetorical Dictionary Entry

6 Lexicon and Linguistic Grammar

The lexicon defines different linguistic characteristics of lexemes such as their categories, government patterns, morphology, etc., and which are used for the realization process. The linguistic grammars of PRESENTOR are used to transform a deep-syntactic representation into

a linearized list of all the lexemes and punctuation marks composing a sentence. The format of the declarative lexicon and of the grammar rules is that of the REALPRO realizer, which we discussed in (Lavoie and Rambow, 1997). We omit further discussion here.

7 Status

PRESENTOR is currently implemented in Java and C++, and has been used with success in projects in different domains. We intend to add a declarative specification of formatting style in the near future.

A serious limitation of the current implementation is the fact that the configurability of PRESENTOR at the micro-planning level is restricted to the lexicalization and the linguistic realization of rhetorical relations. Pronominalization rules remain hard-coded heuristics in the micro-planner but can be guided by features introduced in the presentation representations. This is problematic since pronominalization is often domain specific and may require changing the heuristics when porting a system to a new domain.

CoGenTex has developed a complementary alternative to PRESENTOR, EXEMPLARS (White and Caldwell, 1998) which gives a better programmatic control to the processing of the representations that PRESENTOR does. While EXEMPLARS focuses on programmatic extensibility, PRESENTOR focus on declarative representation specification. Both approaches are complementary and work is currently being done in order to integrate their features.

Acknowledgments

References

Barzilay, R., Rambow, O., McCullough, D, Korelsky, T., and Lavoie, B. (1998). DesignExpert: A Knowledge-Based Tool for Developing System-Wide Properties, In *Proceedings of the 9th International Workshop on Natural Language Generation*, Ontario, Canada.

Ehrhart, L., Rambow, O., Webber F., McEnerney, J., and Korelsky, T. (1998) DesignExpert: Developing System-Wide Properties with Knowledge-Based Tools. Lee Scott Ehrhart, Submitted.

Kittredge, R. and Lavoie, B. (1998). MeteoCogent: A Knowledge-Based Tool For Generating Weather Forecast Texts, In Proceedings of American Meteorological Society AI Conference (AMS-98), Phoenix, AZ.

Kittredge, R., Korelsky, T. and Rambow, R. (1991). On the Need for Domain Communication Knowledge, in *Computational Intelligence*, Vol 7, No 4.

Lavoie, B., Rambow, O., and Reiter, E. (1997). Customizable Descriptions of Object-Oriented Models, In *Proceedings of the Conference on Applied Natural Language Processing (ANLP'97)*, Washington, DC.

Lavoie, B. and Rambow, O. (1997). RealPro – A Fast, Portable Sentence Realizer, In *Proceedings of the Conference on Applied Natural Language Processing (ANLP'97)*, Washington, DC.

Mann, W. and Thompson, S. (1987). *Rhetorical Structure Theory: A Theory of Text Organization*, ISI technical report RS-87-190.

McKeown, K. (1985). *Text Generation*, Cambridge University Press.

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

Rambow, O., Caldwell, D. E., Lavoie, B., McCullough, D., and White, M. (1998). Text Planning: Communicative Intentions and the Conventionality of Linguistic Communication. In preparation.

Rambow, O. and Korelsky, T. (1992). Applied Text Generation, In *Third Conference on Applied Natural Language Processing*, pages 40–47, Trento, Italy.

Reiter, E. (1994). Has a Consensus NL Generation Architecture Appeared, and is it Psycholinguistically Plausible? In *Proceedings of the 7th International Workshop on Natural Language Generation*, pages 163–170, Maine.

Rubinoff, R. (1992). Integrating Text Planning and Linguistic Choice by Annotating Linguistic Structures, In *Aspects of Automated Natural Language Generation*, pages 45–56, Trento, Italy.

White, M. and Caldwell, D. E. (1998). EXEMPLARS: A Practical Exensible Framework for Real-Time Text Generation, In *Proceedings of the 9th International Workshop on Natural Language Generation*, Ontario, Canada.