Portable Natural Language Generation using SPOKESMAN

Marie Meteer
Sage Lab
Rensselaer Polytechnic Institute
Troy, New York 12180
METEER@CS.RPI.EDU

Abstract[1]

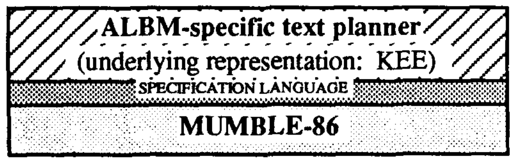
This paper reports on the SPOKESMAN natural language generation system, which is a domain independent text generator designed to incrementally produce text for an underlying application program. This work is a direct outgrowth of the work we reported on at the last ACL Applied Conference in 1988, where we connected an application program directly to the linguistic component, Mumble-86. The major addition has been a new component to the system, a text planner that provides the capability to compose the utterance incrementally. The central feature of the text planning component is a new level of representation that both captures more linguistic generalizations and makes the system more portable, so that we can easily interface to different domains and different kinds of application programs. This larger system is called "Spokesman", as it acts as the mouthpiece for a number of application programs.

1. Introduction

There are generally two approaches to achieving portability. One is to build peripheral components that can automatically or semi-automatically acquire the knowledge needed to port to a new domain. The second is to modularize the system so that the components that are not domain specific are kept separate from those that are, and to try to maximize the amount of knowledge in the more general components. In the long term, a combination of these approaches will certainly be needed. In the work presented here, we have concentrated on the later. For example, by taking advantage of the fact that most of language is not particular to any domain, we have isolated the linguistic realization component and used the Text Structure to capture abstract linguistic generalizations. Also by using the knowledge base of the application directly we can not only capture generalizations about objects which are expressed similarly, but also handle those cases where the means of expression is specific to a particular domain.

2. Modularization

In our earlier work, generation involved three modules: a linguistic realization component (LRC) MUMBLE-86 (Meteer, et.al 1987), an underlying application program, and a special purpose text planner to handle the mapping from one to the other. The text planner used the input specification language to Mumble-86 as a means of compensating for the semantic deficits of linguistically naive underlying application programs without compromising principled grammatical treatments in natural language generation.



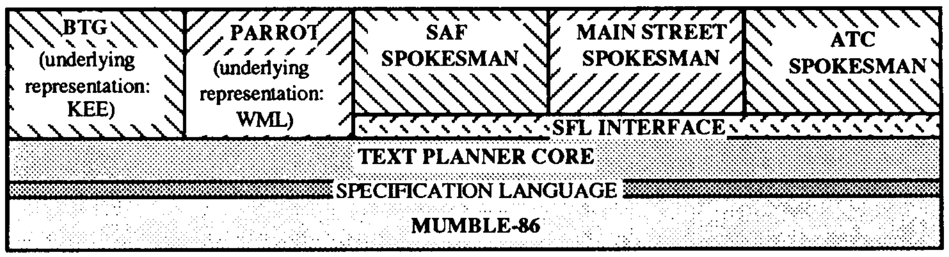
1988 Architecture: Interfacing to Mumble-86

While this modularization isolated the linguistic component, using it directly required the developer to be aware of very low level linguistic details. For example, the specification of a noun phrase requires that information about number, person, and determiner be included. Furthermore, there was no way to ensure that a particular specification built by a text planner would actually be expressible by the linguistic component. For example, there was nothing to prevent a planner from composing a specification combining a completive event with a duration (e.g. *"the plane landed for ten minutes"). Also, the specification language itself cannot capture certain generalizations about what features can co-occur in language and what is expressed by certain combinations of features, leaving them to the text planner. For example, a single noun phrase with a definite article indicates that the entity is known (e.g. "the dog"); however if a proper name is used, the article is omitted even when the entity is known (e.g. "Fluffy").

While this architecture was a successful means of working directly with MUMBLE-86, it left a great deal of work to be done by the planner, most of which must be built specifically for each application. Our approach in developing a text planner for the current system was to introduce modularity into the text planner, separating what is general to language from that which is specific to an application. The resulting system is called SPOKESMAN, and its architecture is shown below. The general knowledge used by the text planner resides in the TEXT PLANNER CORE; the domain specific portions of the text planner are again indicated by diagonal lines.

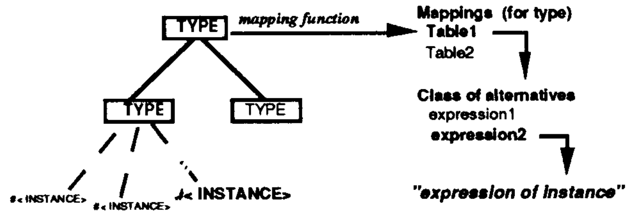
Note that three of the applications shown all use the same knowledge representation language, the Simple Frame Language (SFL, Abrett, et al. 1989). Following our overall approach of modularizing those portions of the system that are shared, we built a subsystem for interfacing with the representation language that contains all the routines for accessing objects in SFL and for handling what is common to all SFL-based applications, such as the concept THING.

[1] The work reported here was done at BBN Systems and Technologies, Cambridge, Massachusetts. I would like to thank all of my BBN colleagues, in particular Glenn Abrett, Stephen Deutsch, Dawn MacLaughlin, and Varda Shaked, for their contributions to this work during my tenure at BBN.



1992 Architecture: The Spokesman Generation System

Spokesman is essentially an *object-oriented* program in that the routines for mapping from one level to the other are specialized for the type of object being mapped, just as generic methods in CLOS or Flavors are specialized for different classes or flavors in those object oriented programming languages. Each mapping function is a table which, when given an object, will walk up the KB hierarchy until it finds a routine associated with that object's type or a type it inherits from. If that routine is a template, it will execute the template; if it is a class of alternatives, it will select one and execute that. This process is shown schematically below. There are different tables for the mappings between each level of representation in Spokesman, and, in some cases, different tables depending on the context defined by representational level.



Mappings

As we discussed earlier, one of our goals has been to isolate what is common to a language (though not necessarily all languages) from what is particular to the application the generator is speaking for. In particular, we wanted to both capture the generalizations available from the cooccurance of features in the linguistic specification and ensure that the specifications that are built are expressible in language. Within the text planner core, these generalizations are captured in the level of representation called the Text Structure (TS), which is used to compose the text. TS is a tree representing the constituency of the utterance, where constituents may be entire paragraphs related by rhetorical relations, or they may be lexically headed constituents internal to a clause. The terms of the TS are abstractions over the concrete resources of language (words, phrases, morphological markers). This vocabulary and the structure built with it provides the text planner with a representation of what decisions have already been made, thus constraining further decisions, and of what opportunities are available for further composition.

3. Capturing differences between domain

In what we have presented so far, the focus has been on taking advantages of similarities within language and among applications to isolate domain independent components from those that need to be specific to the application program. However, there are some things that are intrinsically domain specific, both in what information is expressed and in how it is expressed. A generation system that is to produce realistic texts in a domain must allow the developer to specialize routines at all levels of the generation process.

One example of a domain specific expression is the way pilots are addressed in the Air Traffic Control domain. Rather than using the pilot's name, the controller addresses the pilot using the flight ID of the plane the pilot is flying—in effect he addresses the plane; similarly, pilots address controllers using their function (e.g approach, tower). In SPOKESMAN, this is handled using the mappings. Rather than using the mapping for PERSON, which pilot inherits from, a mapping specific to the concept PILOT is set up, which puts the aircraft instance rather than the pilot instance in the resultant Text Structure node. In the next phase of the generation process, which maps from the text structure to the linguistic specification, the mapping from the aircraft to the lexical resource is used, which combines the airline and the plane's ID number into a phrase, such as "United four fifty one".

4. Conclusion

We have described the modularization of the SPOKESMAN generation system, which is designed to increase its portability, and we have briefly shown how the use of mappings directly from the application's knowledge base can both capture generalities in how information is expressed and allow specializations for domain specific expressions. (For a more detailed description of SPOKESMAN and Text Structure, see Meteer 1991, 1992.)

Abrett, G., Burstein, M., & Deutsch, S. (1989) TARL: Tactical Action Representation Language, An Environment for Building Goal Directed Knowledge Based Simulations. BBN Technical Report No. 7062.

McDonald, D. & Meteer, M. (1988) From Water to Wine: Generating Natural Language Text from Today's Applications Programs, *Proceedings of the 2nd Conference on Applied Natural Language Processing*, Austin, Texas.

Meteer, M. (1991a) SPOKESMAN: Data Driven, Object Oriented Natural Language Generation, *Proceedings of the Seventh IEEE Conference on Artificial Intelligence Applications*, Miami Beach, Florida, February 26-28.

Meteer, M. (1991b) Abstract Linguistic Resources for Text Planning" *Computational Intelligence*. 7(4).

Meteer, M. (1992) *Expressibility and the problem of efficient text planning*. Pinter Publishers. (forthcoming)

Meteer, M., McDonald, D., Anderson, S., Forster, D., Gay, L., Huettner, A., and Sibun, P. 1987. Mumble-86: Design and Implementation. UMass Technical Report 87-87. University of Massachusetts, Amherst, MA.