Hybrid NLG in a Generic Dialog System

Martin Klarner¹ and Bernd Ludwig²

¹ University Erlangen-Nuremberg, Chair for Computer Science 8, Haberstraße 2, D-91058 Erlangen, Germany, klarner@cs.fau.edu, http://www8.informatik.uni-erlangen.de/~klarner/
² University Erlangen-Nuremberg, Chair for Computer Science 5, Haberstraße 2, 91058 Erlangen, Germany, ludwig@forviss.de, http://www-informatik.uni-erlangen.de/~bdludwig/

Abstract. Natural Language Generation (NLG) systems are increasingly becoming available as “market-ready” products, mainly due to the now removed boundary between shallow and deep generation and the emergence of hybrid systems as a de-facto standard. In this paper, we present HYPERBUG³, a novel approach towards hybrid NLG, coupling shallow and deep processing not only with respect to the resources used for parsing and generation, but also on the architectural level to increase processing efficiency for deep generation as well as generative power for shallow generation. The architecture is discussed both in theory and in practice, using a comprehensive example spanning the complete output part of our dialog system.

Keywords: Hybrid NLG architecture, dialog integration, bottom-up generation

1 Introduction

NLG systems are increasingly becoming available as “market-ready” products, mainly because the boundary between shallow and deep generation has been overcome and hybrid systems have emerged as the de-facto standard.

In this paper, we propose a novel variant of hybrid NLG, extending and combining existing approaches. The paper is organized as follows: In the remainder of this section, we will discuss some aspects of hybrid NLG in general. In sect. 2 we describe the system core of HYPERBUG, our hybrid NLG system which couples and interleaves shallow and deep processing not only with respect to the resources, but also on the architectural level to increase processing efficiency as well as generative power. The complete system architecture is discussed in sect. 3. In sect. 4, we will use a comprehensive example to describe our system at work, also addressing some interesting implementation problems and evaluation results. Finally, we discuss some relevant work in our context in sect. 5.

³ The acronym stands for hybrid pragmatically embedded realization with bottom-up generation.
Not to use hybrid systems is quite out of the question now for NLG researchers: Everybody needs them, everybody builds them, and everybody claims to have one. But there are several notions of hybridism around, including stochastic approaches [1], machine learning (ML) [2], and XSLT generation [3]. However, we want to stick to a definition of hybridism that concentrates on a mixing strategy for the two classical approaches, shallow and deep generation. [4] describes two types of such hybrid NLG systems: Type I has shallow generation with deep elements, type II the opposite, i. e. deep generation with shallow elements. We want a third type to be added to this typology: Type III uses separate shallow and deep processing branches and combines the results appropriately, in analogy to the approach taken in VERBMOBIL [5] for the language analysis part of the system.

2 System Core

Our generation system HYPERBUG integrates all three types of hybrid NLG approaches mentioned in sect. 1, i. e. it combines shallow processing with deep elements, deep processing with shallow elements, and concuring shallow and deep processing in a single system, relying on a decision module as front-end and a feedback module from deep to shallow generation as back-end. The system core is displayed in fig. 1.

![Diagram of HYPERBUG system core](image)

**Fig. 1. System Core**

The input structure, a representation of the semantic content in a DRS [6], is fed into a decision module which determines whether shallow or deep generation is more appropriate to produce the desired output. The default decision rule is to use deep generation only if shallow generation is unavailable, i. e. if
there is no pointer to an appropriate template in the index table. The decision is rather based on values for certain XPath variables, resulting in an analysis which is only shallow but nevertheless sufficient for the task. In the decision module, we do not want to preoccupy any computational resources needed only for deep or shallow generation. The XPaths are possibly domain dependent and have to be replaced when a domain change is envisaged. But this need not be the case if the interface specifications remain unchanged. Concurring shallow and deep processing branches combined with the decision module result in our system becoming hybrid type III in our classification.

If shallow generation is chosen, a template system with an advanced pattern matching technique is invoked, using a database with complex templates. The entries in the template database are recursive and modular, they contain subparts and repeatable sections, syntactic features and pointers to wave files for speech synthesis. Furthermore, access to the NLG lexicon and to the morphology component, linguistic resources normally used only for deep generation, is encoded in the templates. This enriched template system makes our NLG approach one of type I in our classification in sect. 1.

On the other hand, if deep generation is selected, a “sentence planner” first converts the input DRS into an extended logical form (ELF). The extensions of a conventional LF include syntactic features like tense and mode, topicalization information, and subordination clause type. Surface forms like proper nouns are also allowed in the ELF; hence the deep generation module is of hybrid type II. The ELF is then processed by the deep syntactic realization module. As the acronym HYPERBUG already suggests, we have implemented an extended version of the bottom-up generation algorithm presented in [7] as our deep generation component, using a unification grammar and re-using the system lexicon and morphology component. During the generation process, syntactic information is collected by the unification algorithm and stored in an extended derivation tree. Its leaves represent the surface structures of the words which make up the generated sentence as well as their syntactic categories and agreement features.

After deep generation has been processed, a “bridge” generates a new template out of the derivation tree and feeds it back to the shallow generation branch for further use in the current dialog or in subsequent dialogs. The decision module is also informed to update its index table of available templates, so that utterances of the same type do not have to be processed by the deep generation branch any more. The workload on deep generation is thus consistently reduced at runtime. – In any case, HYPERBUG produces a sentence which is then converted into a wave file and uttered to the user by the speech synthesizer.

3 Embedding Generation into our Dialog System

Our NLG system is not implemented as a stand-alone product, but embedded in a complete generic multi-agent dialog system containing agents for speech

---

4 This spares using the somewhat more complex template matching algorithm.
5 We will provide an example in sect. 4.
analysis, dialog management, speech output, and for the technical system which encapsulates application information and control.

The dialog manager (DM) integrates user and system utterances into the discourse context. It is also responsible for content determination, i.e., it provides the generation module with an output specification, albeit without any linguistic knowledge: The DM is amodal, and from its perspective the language output part of the system is just another application, comparable to the technical system. In our dialog system, this desired output is written in a Discourse Representation Structure (DRS) and encoded in XML.⁶

Fig. 2 sorts a zoom out of fig. 1 and shows the distributed architecture of the dialog system in which the generation module is integrated. In this system, content for utterances can be determined at various places: pragmatic feedback to user requests is generated within the technical system; it is responsible for content generation in cases when system initiative is required as well. The DM itself can generate content for utterances if the current dialog situation requires it to do so (e.g., for the clarification of syntactic ambiguities in a previous user utterance).

---

⁶ Technically, the XML structure is the content part of a message in the agent communication language KQML, but this is not in our focus here.
The DM also decides which speech act is adequate to integrate the new utterance to be generated into the current dialog context\textsuperscript{7}. For planning the utterance in detail, all this information is handed over to the generation module. It has to decide how the speech act and the content of the utterance are verbalized.

4 A comprehensive example

In this section, we want to give an overview of our system at work, using an example in a home A/V management domain which is simple but nevertheless points to some interesting linguistic and implementation problems.

**Dialog Manager.** Suppose the user has just requested a VCR recording of the program “Tatort” tonight at 08:15 PM and the system wants to inform the user that the request has been processed successfully. In a situation like this, the dialog manager (DM) receives a message from the technical system whose content is shown in fig. 3.\textsuperscript{8}

\texttt{applicationStatus} \texttt{hasTimeValue(a, c), hasTimeValue(c, l), StartTime(e), hasStartTime(a, s), EndTime(c), hasEndTime(a, c), StartTimeValue(2004-03-28 20:15:00), hasStartTimeValue(a, 2004-03-28 20:15:00), AVProgram(d, p), hasAVProgram(a, p, AVContentInfo(c), TitleSpec(d, e), MainTitle(m, n, m), MainTitleValue(Tatort), hasMainTitleValue(a, Tatort), ValueLang(e), hasValulang(e, a, ValueLang), AVInstanceInfo(f), AVLocation.ID(1, t, f), AVLocation.IDValue(1, 25), hasAVLocationIDValue(f, 1, ARD), ActionTypeValue(RECORD), hasActionTypeValue(a, RECORD), ActionStatus(a, 0), hasActionStatus(a, WAITING), ActionStatusValue(25), hasActionStatusValue(a, 25, WAITING)}

Fig. 3, DRS representation of the system response to a user request

The DRS is accompanied by the information that the message is a response to a request, that a reaction to the request has taken place (the request has been satisfied), and that the effects of the reaction are described in the content of the message. From this information, the DM concludes that it can update the dialog situation: On the one hand, the pending user request needs a response about the successful outcome of the related system activities, and on the other hand, the information in the system message must be communicated to the user.

According to the current state of the interaction between user and system, the DM decides how both update requirements can be satisfied. For an utterance that serves as a correct and useful response a decision has to be made about the speech act and the content for a new utterance. In our example, all information contained in the system message is relevant. The DM encapsulates

\textsuperscript{7} This decision procedure is described elsewhere and cannot completely be spelled out here.

\textsuperscript{8} The relevant parts for processing the example are marked bold in fig. 3.
it in a message inform speech act that serves for communicating information without expecting a reaction and therefore meets the needs in the current dialog situation.

Therefore, the dialog manager (DM) initiates the generation of the German sentence “Ich werde um 20:15 h ’Tatort’ aufnehmen.” (“I’m going to record ’Tatort’ at 08:15 PM.”). The DRS in fig. 3 is thus transformed into the XML structure displayed in fig. 4, which serves as input for our generation system.

![XML input structure for the system response](image)

**Preprocessing:** The decision module. At first, the XML input is analyzed and information relevant to determine the appropriate generation branch is extracted. This is done using XPath expressions.

In our example, the XML input indicates that the desired action is of the Record type. Suppose we have not yet generated a sentence of this type before in our dialog history. Therefore, the decision module cannot find an appropriate entry in its index table of templates and must pass the task of generating the sentence to the deep generation branch.

**Deep generation.** The “sentence planner” converts the DRS encoded in the XML structure in fig. 4 into the following ELF:

```
(AT (20:15 (FUT (RECORD SYSTEM TATORT))))
```

To this end, it completes the analysis performed by the decision module and extracts the relevant XPaths needed for conversion. These paths are stored for further use. The ELF contains a proper noun, TATORT, which is normally not allowed in a semantic representation for deep generation, but is valid in our type II hybrid NLG system. After processing the ELF input, BUG produces the derivation tree displayed in fig. 5.

---

9 From the linguistic point of view, this example is simple in several ways: It is just a single main sentence in normal word-order (no topicalization), the verb form is not separated, and there are no complex noun phrases. However, we have an analytical tense form (future tense) and a proper noun which has to be realized without any article here.
Post-processing: The "bridge". Coupling deep and shallow processing in our NLG system is completed by the "bridge" between deep and shallow generation. This module uses the output of deep generation to generate the template shown in fig 6 which is then passed on to the shallow generation branch where it is stored and can be used later for similar sentences.

![Fig. 5. Derivation tree for our example](image)

![Fig. 6. Template for our example](image)

The dialog act for the generated template is taken from the initial input structure, as well as the relevant XPath entries which were stored by the “sentence planner” before. The resulting entry in our template database contains tokens with XPaths and syntactic features (category and agreements), several constant and variable parts¹, and an interesting generalization feature: The brackets () indicate a potential enumeration (in our example, this is useful if more than one program will be recorded).

Shallow generation. Hence, the next time a sentence of the type described in our example has to be generated, the decision module is now able to find an appropriate entry in its index table. Therefore the shallow generation branch is invoked. A matching template like the one depicted in fig. 6 is found and instantiated, producing the sentence mentioned above.

Speech synthesis. Finally, the generated sentence is sent to the speech synthesis agent which is essentially a wrapper agent around the open-source synthesizer MBROLA. The agent checks for each constant part of the sentence whether it has already been synthesized before. If so, it uses a pointer to previously stored wave files; if not, the newly synthesized constant parts are stored for further use. Then, the variable parts are synthesized as well, the wave file is concatenated and uttered.

¹The distinction between constant and variable parts is important for our speech synthesis agent, see below.
5 Related Work

In [8], the author applies Conversation Acts Theory to NLG, namely to the text planning part. In our system, we extend this application to linguistic realization by identifying appropriate realization methods for each conversation act type.

Explanation-based generalization or explanation-based learning (EBL), an approach known from Machine Learning (ML), contains ideas similar to the ones described here and has already been applied to NLG: In [9], templates for sub-grammars are generated from a corpus. However, normally a training phase, a running phase, and a separate training corpus are needed in EBL, which is not the case in our approach: We just process our dialogs directly, generating everything on-the-fly.

As we use XML as input structure, generation from XSLT [3] can be seen as an alternative to our approach, albeit without ML ideas and deep processing. Therefore XSLT generation is rather shallow and static in our point of view.

6 Conclusion and Further Work

We have presented a hybrid approach to linguistic realization embedded in a natural language dialog system. The NLG system is fully implemented in Java and currently used in three different domains, namely home A/V management, model train controlling and B2B e-procurement. Domain shifts were carried out without major problems, but admittedly not completely automatically.

We are going to introduce deep processing into our “sentence planner” in the near future using the DL inference engine RACER [10]. Furthermore, we want to improve the decision module by implementing a planning procedure which makes use of the speech act, the discourse situation and the user model to ensure that the most appropriate processing branch is selected.

References