A Flexible Pragmatics-driven Language Generator for Animated Agents

Paul Piwek

ITRI – Information Technology Research Institute
University of Brighton
Paul.Piwek@itri.bton.ac.uk

Abstract

This paper describes the NECA MNLG; a fully implemented Multimodal Natural Language Generation module. The MNLG is deployed as part of the NECA system which generates dialogues between animated agents. The generation module supports the seamless integration of full grammar rules, templates and canned text. The generator takes input which allows for the specification of syntactic, semantic and pragmatic constraints on the output.

1 Introduction

This paper introduces the NECA MNLG; a Multimodal Natural Language Generator. It has been developed in the context of the NECA system.[1] The NECA system generates dialogue scripts for animated characters. A first demonstrator in the car sales domain (ESHOWROOM) has been implemented. It allows a user to browse a database of cars, select a car, select two characters and their attributes, and subsequently view an automatically generated film of a dialogue about the selected car. The demonstrator takes the following input:

- A database with facts about the selected car (maximum speed, horse power, etc.).
- A database which correlates facts with value judgements.

---

[1]NECA stands for 'Net Environment for Embodied Emotional Conversational Agents' and is an EU-IST project.

- Information about the characters: 1. Personality traits such as extroversion and agreeableness. 2. Personal preferences concerning cars (e.g., a preference for safe cars). 3. Role of the character (seller or customer).

This input is processed in a pipeline that consists of the following modules in this order:

- A DIALOGUE PLANNER, which produces an abstract description of the dialogue (the dialogue plan).
- A MULTI-MODAL NATURAL LANGUAGE GENERATOR which specifies linguistic and non-linguistic realizations for the dialogue acts in the dialogue plan.
- A SPEECH SYNTHESIS MODULE, which adds information for Speech.
- A GESTURE ASSIGNMENT MODULE, which controls the temporal coordination of gestures and speech.
- A PLAYER, which plays the animated characters and the corresponding speech sound files.

Each step in the pipeline adds more concrete information to the dialogue plan/script until finally a player can render it. A single XML compliant representation language, called RRL, has been developed for representing the Dialogue Script at its various stages of completion (Piwek et al., 2002).

In this paper, we describe the requirements for the NECA MNLG, how these have been translated into design solutions and finally some of aspects of the implementation.

2 Requirements

The requirements in this section derive primarily from the use case of the NECA system. We do, however, try to indicate in what respects these requirements transcend this specific application and are desirable for generation systems in general.

In the NECA system, the dialogue planner creates a dialogue plan consisting of (1) a description of the participants, (2) a characterization of the common ground at the outset of the dialogue in terms of Discourse Representation Theory (Kamp and Reyle, 1993) and (3) a set of dialogue acts and their temporal ordering. For each dialogue act, the type, speaker, set of addressees, semantic content, what it is a reaction to (i.e., its rhetorical relation to other dialogue acts), and emotions of the speaker can be specified. The amount of information which the dialogue planner actually provides for each of these attributes varies, however, per dialogue act: for some dialogue acts, a full semantic content can be provided –in the form of a Discourse Representation Structure– whereas for other acts, no semantic content is available at all. Typically, the dialogue planner can provide detailed semantics for utterances whose content is covered by the domain model (e.g., the car domain) whereas this is not possible for utterances which play an important role in the conversation but are not part of the domain model (e.g., greetings). This state of affairs is shared with most real-world applications.

Since generation by grammar rules is primarily driven by the input semantics, for certain dialogue acts full grammar rules cannot be used. These dialogue acts may be primarily characterized in terms of their, possibly domain specific, dialogue act type (greeting, refusal, etc.). Thus, we need a generator which can cope with both types of input, and map them to the appropriate output. Input with little or no semantic content can typically be dealt with through templates or canned text, whereas input with fully specified semantic content can be dealt with through proper grammar rules. Summarizing, we need a generator which can cope with (linguistic) resources that contain an arbitary combination of grammar rules, templates and canned text.

REQUIREMENT **2**: *The generator should allow for combinations of different types of constraints on its the output, such as syntactic, semantic and pragmatic constraints*

In the NECA project the aim is to generate behaviour for animated agents which simulates affective situated face-to-face conversational interaction. This means that the utterances of the agents have to be adapted not only to the content of the information which is exchanged but also to many other properties of the interlocutors, such as their emotional state, gender, cultural background, etc. The generator should therefore allow for such parameters to be part of its input.

REQUIREMENT **3**: *The generator should be sufficiently fast to be of use in real-world applications*

The application in which our generator is being used is currently fielded as part of a net-environment. The application will be evaluated with users through online questionnaires which are integrated in the application and analysis of log files (to answer questions such as 'Do users try different settings of the application?', etc. See Krenn et al., 2002). Therefore, the generator will have to be fast in order for it not to negatively affect the user experience of the system.

3 Design Solutions

The NECA MNLG adopts the conventional pipeline architecture for generators (Reiter and Dale, 2000). Its input is a RRL dialogue plan. This is parsed and internally represented as a PROFIT typed feature structure (Erbach, 1995). Subsequently, the dialogue acts in the plan are realized in accordance with their temporal order. For each act, first a deep syntactic structure is generated. The deep structure of referring expressions is dealt with in a separate module, which takes the common ground of the interlocutors into account. Subsequently, lexical realization (agreement, inflection) and punctuation is performed. Finally, turn-taking gestures are added and the output is mapped back into the RRL XML format.

Here let us concentrate on our approach to the generation of deep syntactic structure and how it satisfies the first two requirements. The input to the MNLG is a node (i.e., feature structure) stipulating the syntactic type of the output (e.g., sen-

tence: <s), semantics and further information on
the current dialogue act in PROFIT:[2]

```
(<s &
 sem!drs([c_27],
         [type(c_27,prestigious),
          arg1(c_27,x_1)])&
 currentAct!speaker!
    (name!john &
     polite!yes & ...)
)
```

Thus various types of information are combined
within one input node. Generation consists of tak-
ing the input node and using it to create a tree
representation of the output. For this purpose,
the MNLG tries to match the input node with the
mother node of one of the trees in its tree repos-
itory. This tree repository contains trees which
can represent proper grammar rules, templates and
canned text. Matching trees might in turn have in-
complete daughter nodes. These are recursively
expanded by matching them with the trees in the
repository, until all daughters are complete.

A daughter node is complete if it is lexically
realized (i.e., the attribute form of the node has
a value) or it is of the type <np and the seman-
tics is an open variable. In the latter instance, the
node is expanded in a separate step by the refer-
ring expressions generation module. This module
finds the discourse referent in the common ground
which binds the open variable and constructs a de-
scription of the object in question. The descrip-
tion is composed of the properties which the ob-
ject has according to the common ground, but can
also be empty if the object is highly salient. The
module is based on the work of Krahmer and The-
une (2002). The (empty) description is mapped
to a deep syntactic structure using the tree repos-
itory. Lexicalization subsequently yields expres-
sions such as 'it' (empty descriptive content) or,
for instance, 'the red car'.

Let us return to the tree repository and illus-
trate how templates and rules can be represented
uniformly. The representation of a tree is of the

form (Node, [Tree1, Tree2, ...]), where
the list of trees can be empty, yielding a tree con-
sisting of one node: (Node, []). The following
is a template for dialogue acts of type greeting
with no semantic content and a polite speaker.

```
(<s &
 currentAct!
    (type!greeting &
     speaker!polite!"yes" &
     speaker!name!Speaker) &
 sem!"none",
        [(<s & form!"hello!",[]),
         (<fragment &
          form!"My name is",[]),
         (<np &
          sem!concept(Speaker),[])
        ]).
```

This is a template for the text 'Hello! My name is
SPEAKER'. Where SPEAKER is a variable which
is bound to the name of the speaker of the utter-
ance. The noun phrase (<np) for this name is gen-
erated by the referring expression generation mod-
ule. The following is a tree representing a gram-
mar rule of the familiar type $S \rightarrow NP\ VP$:

```
(<s &
 currentAct!type!statement &
 currentAct!CA &
 argGap!ArgGap &
 auxGap!AuxGap &
 sem!drs(_,[negation(
            drs(_,
                [type(E,Type)
                 arg1(E,X)|R]))]
            ),
     [(<np &
       currentAct!CA &
       sem!X,[]),
      (<vp &
       argGap!ArgGap &
       auxGap!AuxGap &
       negated!<true &
       sem!drs(_,[type(E,Type)|R]) &
       currentAct!CA,_)
     ]).
```

Note that this rule applies to an input node whose
semantic content contains a negation. The nega-
tion is passed on to the VP subtree via the feature
negated. The attributes argGap and auxGap
allow us to capture unbounded dependencies via
feature perlocation. Our use of trees is related to
the Tree Adjoining Grammar approach to genera-
tion (e.g., Stone and Doran, 1997).[3]

---

[2]That is, PROLOG with some sugaring for the rep-
resentation of feature structures. Feature structures are
also used in the FUF/SURGE generator. It is different
from the NECA MNLG in that it takes as input thematic
trees with content words. Furthermore, it allows for con-
trol annotations in the grammar and uses a special inter-
preter for unification, rather than directly PROLOG. See
http://www.cs.bgu.ac.il/surge/.

[3]Their generation algorithm is, however, very different
from the one proposed here. Whereas they propose an in-
tegrated planning approach, we advocate a very modular sys-

The value of the attribute `currentAct` is passed on from the mother node to the daughter nodes. Thus any pragmatic information (personality, politeness, emotion, etc.) is passed on through the tree and can be accessed at a later stage, for instance, when lexical items are selected.

4 Implementation

The NECA MNLG has been implemented in PROLOG. The output is in the form of an RRL XML document. Table 1 provides a sample of the response times of the compiled code running on a Pentium III Mobile 1200 Mhz with Sicstus 3.8.5 PROLOG. We timed the complete generation process from parsing the XML input to producing XML output, including generation of deep syntactic structure, referring expressions, turn taking gestures (not discussed in this paper), etc.

input	# acts	= 1	≤ 10
A	19	0.230s	0.741s
B	22	0.290s	0.872s
C	23	0.290s	0.801s
D	31	0.431s	1.372s

Table 1: Response Times of the MNLG

The results show generation times for entire dialogues and according to whether the generator was asked to produce exactly one solution or select at random a solution from a set of at most ten generated solutions (the latter strategy was implemented to obtain more variation in the generator output). On average for $= 1$ the generation time for an individual dialogue act is almost $\frac{1}{100}$ of a second. For ≤ 10 it is $\frac{4}{100}$ of a second. The generator uses a repository of 138 trees (including the two examples given above). The repository has been developed for and integrated into the ESHOWROOM system which is currently being fielded. A start is being made with porting the MNLG to a new domain and documentation is being created to allow our project partners to carry out this task. We hope that our efforts will contribute to addressing a challenge expressed in (Re-

tem, supporting fast generation. Moreover, by using features for unbounded dependencies we do not require the adjunction operation, which is incompatible with our topdown generation approach. We follow Nicolov et al. (1996), who also use TAG, in their commitment to flat semantics. Their generator does, however, not take pragmatic constraints into account.

iter, 1999): "We hope that future systems such as STOP will be able to make more use of deep techniques, because of advances in linguistics and the development of reusable wide-coverage NLG components that are robust, well-documented and well engineered as software artifacts."

In our view the best way to approach this goal is by providing a framework which allows for the flexible integration of shallow and deep generation, thus making it possible that in the course of various projects, deep analyses can be developed alongside the shallow solutions which are difficult to avoid altogether in software development projects, due to the pressure to deliver a *complete* system within a certain span of time.

Acknowledgements

References

Gregor Erbach. 1995. PROFIT *1.54 user's guide*. University of the Saarland, December 3, 1995.

Hans Kamp and Uwe Reyle. 1993. *From Discourse to Logic*. Kluwer, Dordrecht.

Emiel Krahmer and Mariët Theune. 2002. Efficient context-sensitive generation of referring expressions. In: Kees Van Deemter and Rodger Kibble (eds.), *Information Sharing*, CSLI, Stanford.

Brigitte Krenn, Erich Gstrein, Barbara Neumayr and Martine Grice. 2002. What can we learn from users of avatars in net environments?. In: *Proc. of the AAMAS workshop "Embodied conversational agents - let's specify and evaluate them!"*, Bologna, Italy.

Nicholas Nicolov, Chris Mellish & Graeme Ritchie. 1996. Approximate Generation from Non-Hierarchical Representations, *Proc. 8th International Workshop on Natural Language Generation*, Herstmonceux Castle, UK.

Paul Piwek, Brigitte Krenn, Marc Schröder, Martine Grice, Stefan Baumann and Hannes Pirker. 2002. RRL: A Rich Representation Language for the Description of Agent Behaviour in NECA. *Proc. of the AAMAS workshop "Embodied conversational agents - let's specify and evaluate them!"*, Bologna, Italy.

Ehud Reiter. 1999. Shallow vs. Deep Techniques for handling Linguistic Constraints and Optimisations. *Proc. of KI-99 Workshop "May I speak freely"*.

Ehud Reiter and Robert Dale. 2000. *Building natural language generation systems*. Cambridge University Press, Cambridge.

Matthew Stone and Christy Doran. 1997. Sentence Planning as Description Using Tree-Adjoining Grammar. *Proc. ACL 1997*, Madrid, Spain.