Course Project

MMIS 670 - Dr. James Cannady

Bryan P. Blue

Nova Southeastern University

July 11, 2004

Table Of Contents

Table Of Contents 2

Abstract 3

Artificial Intelligence in Natural Language Generation 3

Overview 4

Content Determination 5

Discourse Planning 6

Microplanning 8

Sentence Aggregation 8

Lexicalization 9

Referring Expression Generation 10

Sentence Realisation 11

Realisation Packages 12

Size Constraints 13

Machine Learning in Discourse Planning 14

Evolutionary Algorithms and Content Planner Construction 15

Explanation-based Learning 17

Examples Of Uses Today 18

Future Work 18

References 20

Abstract

Natural Language Generation (NLG) is an area of computational linguistics that deals with computer systems that can generate texts in human languages from a non-linguistic data. An overview of current NLG steps is given and then followed with the descriptions of some packages that are under development that may be general enough to be used under most NLG domains. Artificial intelligence techniques that have been used to enhance current NLG procedures are then shown. Finally, a short discussion of current applications where NLG is already in use is reviewed.

Artificial Intelligence in Natural Language Generation

Natural Language Generation studies the various ways in which a system can respond to humans by using natural language. Reiter and Dale (1997) give a clean explanation:

> Natural Language Generation is the subfield of artificial intelligence and computational linguistics that is concerned with the construction of computer systems that can produce understandable texts in English or other human languages from some underlying non-linguistic representation of information.

Many systems already exist, each focusing on it's own domain of knowledge. They are used to automatically generate texts for simple reporting and help messages to longer, more complex documents. In general, a knowledge base is assembled and then used along with appropriate subsystems to determine what needs to be said and then how to say it.

NLG is a completely different problem than Natural Language Understanding (NLU). There has been extensive work on NLU, which can be attributed to the wide availability of input as well as this area having a well-defined set of tasks. This is not always the case with NLG. Using NLU it is easy to process and understand this text, but with NLG, it is not clear what to

generate the natural language from. The representation of information is key to NLG (Dale, Eugenio, Scott, 1998).

NLG is most commonly used where systems are needed to automate the presentation of information to humans in a form that is familiar and easy to understand. Many times it is easy to capture data or information and store it in databases and spread sheets, but these often require an expert to understand. NLG can be used to take this same information and transform it into something more easily understood by a non-expert (Dale, Reiter 1997).

One problem with NLG is the lack of standardized knowledge structures. Each solution that focuses on NLG uses its' own representation to handle the unique problems associated with its' own domain. Input may take the form of an expert systems knowledge base or as simple as stock market prices or weather data. This makes comparison of the various approaches difficult. It also makes reuse almost impossible (Cote, Moulin 1990).

Overview

The most common solutions for NLG are given by a common pipelined structure that consists of the following modules: content planning, microplanning and realization. Each of these steps is normally performed independently and sequentially.

The content planning module is responsible for identifying a set of domain facts to generate the texts. It is responsible for organizing the facts into structures that represent a logical text. Microplanning takes these domain facts and recodes them into suitable linguistic terms. Realization then takes this structure and generates the final sentences and paragraph structures. Realization is also responsible for the presentation such as displaying the texts on the computer screen or translating the output into an appropriate format for subsequent speech synthesis (Bleam, Doran, Palmer, Stone, Webber 2001).

Content Determination

As mentioned above, content determination is the process of determining what information should be used to generate the texts. This portion is usually very domain and application dependent. It relies heavily on a database of information that is usually constructed for only the specific domain. For example, stock ticker information is very different than meteorological data or real estate listing information.

The data to be used in text generation is usually a summarization of part of this database of information. In a common approach, this information can be translated into a formal grammar. This structure can then classify the data as being an entity, concept or relation in the given domain. Often these values are stored as attribute-value matrices.

Example of an attribute-value matrix saying that a Trans-Am is a sports car and information about standard performance:

Message-id: 001 *Relation: IDENTITY* *Arguments:* *arg1. SPORTS CAR* *arg2. TRANS-AM*	*Message-id: 002* *Relation: PERFORMANCE* *Arguments:* *Horsepower: 285* *TopSpeed: 140*

Deciding what needs to be said is very similar to the type of problem that has to be solved in the creation of an expert system. The knowledge acquisition steps can be modified to the task of content determination based on the type of data being handled and type of texts that need to be generated. One particular way to analyze content rules is through the use of a corpus (Dale, Reiter 1997).

A corpus is large collection of text (Norvig, Russel 2003). A large enough collection can be used to extract rules for what should be said. Dale and Reiter (1997) proposed the following general outline on how to accomplish this:

1. Break the texts down into manageable parts, texts into paragraphs, paragraphs into sentences and finally sentences into phrases.
2. Take each phrase and find a relation back to its' data source.
3. Find groupings of phrases that can be treated as classifications of similar messages.
4. Take the classifications of similar messages and then try to determine the conditions that they appear.
5. Review the findings with a domain expert, making modifications as necessary.
6. Once the domain expert is satisfied with the rules, collect a new, larger set of texts from the corpus and start with step 1.

Although this may seem straight forward, it will take a considerable amount of time and effort to create the initial content determination engine (Dale, Reiter 1997). In effect, the system and the system designer must become an expert in the domain.

Discourse Planning

Once the initial content determination has been done, the information, encoded as messages, must be organized into a logical fashion such that rational text can be generated. This process is usually placed within the content determination module and is referred to as discourse planning. Dale and Reiter (1997) give a generalization of this step: this process can be likened to making the decision to either start the generated text with a summary, and then give specifics, or visa versa.

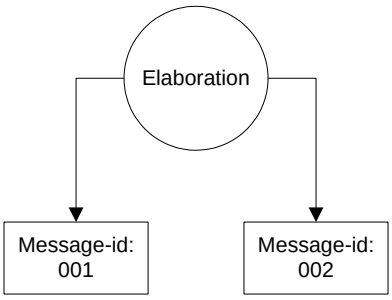
The discourse planning stage create a text plan that can be represented as a tree where the leaf nodes are the messages from the content determination phase and the intermediate nodes specify conceptual groupings. These groupings are key in specifying how all the various text fragments are related (Dale, Reiter 1997).

A simple example can be given with the following messages that were generated by some content determination phase:

Message-id: 001 *Relation: IDENTITY* *Arguments:* *Name: JOHN* *sex: MALE* *pastime: COOK*	*Message-id: 002* *Relation: COOKING* *Arguments:* *rating: 10* *name: BAKED-ALASKA*

Literally, this can be described in the following manner. Message 001 states John is a male who likes to cook. From message 002, one of his favorite recipes is Baked Alaska.

The two texts can be thought of being related by Baked Alaska being and example of a dish that John likes to make. These two texts could be represented as being related by specifying that the relation is “Exemplification” or “Elaboration”. Part of a tree that relates the two would logically look like:



This diagram can be interpreted, as message 002 is an elaboration of message 001. As of the writing of this paper, there is no official set of discourse relations. Much of this is due to the variations of ideas that are domain dependent (Dale, Reiter 1997).

Two of the most common methods used for discourse planning are based on Rhetorical Structure Theory or schemata. In either case, the rules are typically coded by hand. This requires a great expertise in linguistics. It also makes future modification difficult (Androutsopoulos, Dimitromanolaki 2003). This is not to say that this is the only way in which discourse rules are created or represented. Other options exists and are typically based on the form that is required as input for the next phase, microplanning.

Microplanning

Microplanning is the process that is responsible for creating the association between domain knowledge and the linguistic representation (Bleam, Doran, Palmer, Stone, Webber 2001). The process of microplanning can be broken down into three general functions.

Sentence Aggregation

Microplanning is responsible for sentence aggregation. It locates information that can be combined during sentence generation to create more complex and compact texts. It is often used to take a series of facts that relate to the same object and combine the facts into a compound structure. E.g. “gray car” and “sports car” could be two pieces of information about a vehicle. The aggregation step would take these and create an appropriate structure like “the gray sports car”.

Sentence aggregation takes the text plan tree and its messages and transforms the tree into a new text plan. This time, the leaf messages are arranged such that the topics that will be combined into sentences are specified. It is responsible for choosing the appropriate messages to

place together within the same sentence as well as specifying the way in which they should be combined (Dale, Reiter 1997).

In the example of John given before, there could be a couple ways the aggregation could be made.

1. No aggregation. This could result in a structure of simple sentences such as what follows. Notice the use of "he" that is generated from the referring expression generation.
    a. "John is a male who likes to cook. He likes to make Baked Alaska."
2. Use a conjunction such as "and".
    a. "John is a male who likes to cook and he likes to make Baked Alaska."

There are other possible forms that can be generated; the challenge is picking the structure that will produce the most fluid, or human like, sentences (Dale, Reiter 1997).

Lexicalization

The microplanning stage also performs lexicalization by making lexical choices about what items should appear in the generated texts. Often this takes the form of taking domain specific information and choosing the appropriate word or phrase that describes it. Here information contained in the knowledge base such as "incline = 60 degrees" may more appropriately be used in a sentence as "very steep incline". Lexicalization is also the important piece that helps allow for multiple languages to be generated from the same system (Dale, Reiter 1997).

In the example message-id: 001, it is clear that we have entities of a name John, his gender male, and his pastime or hobby of cooking. The lexicalization routines are responsible for determining how each of these entities will be realized.

Suppose we choose the following words for each entity: “John” for JOHN, “is a male” for the gender MALE and “likes to cook” for the pastime of COOK. This could then be realized as the sentence “John is a male that likes to cook.”

Alternately, the content determination phase could have noticed that gender MALE is not needed and choose not to output this fact. An alternate sentence, and arguably better, would be “John likes to cook.”

Lexicalization is responsible for making sure words are not overused. In our example, “cooking” could be replaced in some instances with the other words such as “preparing”. This process can add variety to the generated sentences. Also, differing levels of formalism can be realized by changing the words that are used. For example, using “father” is usually considered more formal than the use of “dad” (Dale, Reiter 1997).

Referring Expression Generation

Microplanning also needs to be able to handle referring expression generation. Here, the system needs to determine identifying descriptions for the generated text that take the place of any knowledge base representation and enhance it with any important referential words (Bleam, Doran, et. al. 2001). An agent may need to generate text that tells the reader to press the “eject button”. It may be helpful to know that this is also the red button. This information would then be transformed into the string “red eject button” for easier reference.

The referring expression generation is responsible for coming up with other alternatives for identities to help alleviate the generated text from sounding monotonous (Dale, Reiter 1997). It could also use “it”, “the button” or “eject button” in various contexts. The proper selection depends on what has already been generated in previous texts. For instance, if “red eject button”

had already been mentioned, using “it” would be ok. If “red eject button” had not been mentioned, it would not be appropriate to use “it”.

Although these three tasks seem independent at first, Bleam, Doran, et. al. (2001) note that they do interact. This interaction, along with the fact that the interactions are related to the order in which the functions are performed, has lead to real challenges in creating any sort of uniform microplanning process.

The microplanner’s tasks must be performed before any of the realisers are applied to generate texts (Dale, Essers n.d.). The microplanner is responsible for determining the structures of what constitutes a good sentence. It also needs to take into consideration what items will and will not be used to realize the final sentences. Therefore, the microplanner must be intricately tied to any knowledge base for the given domain (Dale, Essers n.d.).

Sentence Realisation

Realisation is the final task. It is where grammatically correct sentences are finally generated. It is the module that contains the knowledge of the actual grammar for the natural language (Dale, Reiter 1997).

Here are some simple examples of the type of knowledge that is applied during the realisation process (Dale, Reiter 1997).

Agreement: In English, some words must agree in number, such as singular and plural. E.g. “His car” vs. “Their cars”.

Verb Tense: The realiser has to construct the proper tense of the verb as requested. E.g. a past tense verb is requested so generate “That was fun.” If a present tense form was requested it would need to generate “That is fun.”

Realisation Packages

Many different ways to realise sentences have been developed. The initial overviews that are given tie directly into the discussion of this paper. Alternates methods are then given. There are three largely accepted packages that can be used for general sentence realisation, FUF/SURGE, PENMAN/NIGEL and KPML/NIGEL (Dale, Eugenio, Scott, 1998).

PENMAN is a natural language sentence generator. It was developed by the University of South Carolina and has been in use since 1978. The English grammar is named NIGEL. It is written in Common Lisp and contains over 90,000 words in its lexicon (Single-sentence natural… n.d.).

Komet-Penman Multilingual (KPML) is based on Systemic Functional Grammar and is a multilingual extension of the Penman system (Dale, Essers n.d.). KPML was specifically designed for multi-language generation. It currently supports the English, German, Dutch, Chinese, Spanish, Russian, Bulgarian and Czech languages. Its creators targeted it to be used where a flexible and fast solution is needed such as in web page responses or “live” conversations (Bateman 2003).

KPML has been in development for over a decade and is maintained at the University of Bremen. It is written in ANSI Common LISP and uses the Common Lisp Interface Manager for its Graphical User Interface (GUI). Internally, it uses a Systemic-Functional Linguistics (SFL) grammar framework, which is fairly well known in the NLG community (Bateman 2003).

Functional Unification Formalism (FUF) is a package written in Common Lisp, which uses graph unification techniques. It takes a sentence specification, and a grammar for the output language, to generate a syntactically specified structure that can be used to produce the required sentence (Dale, Essers n.d.). FUF takes the sentence specification and the grammar in the same

form called a Functional Descriptions (FDs). FDs are recursive attribute-value matrices. SURGE is a syntactic realization grammar written in FUF and used to generate English texts (Elhadad 1997).

Dale and Essers (n.d.) make a general comment about generation grammars such as NIGEL and SURGE. Even though they have been in existence for many years now, they are far from complete. As in the NLU area, there is no one lexical analyzer that can properly handle all sentences, there is no generation grammar that will have all the necessary structures in place to be able to handle all the various generation tasks. To compound the problem, changes are not easily made to the generation grammars. To make such a change requires a good understanding of the theories behind the model of the grammar. This is one reason that reuse of these systems is still a problem (Dale, Essers n.d.).

There has been some exploration into the idea of realisers being the inverse of the parsing problem. In theory, it sounds like a viable option, but the problem stems from the notion that the output of a parser is the same as what is needed as the input to a realiser. This is not currently the case and continues to be an area of exploration. The benefit of such a system would be the ability to have a bi-direction grammar. The system would have the ability to understand and generate syntax for a given domain. This would be very useful in maintaining a dialog (Dale, Reiter 1997). Unfortunately, this is still just a nice theory.

Size Constraints

Also, size constraints need to be considered while writing. In many cases, the output is limited to character counts or page sizes. NLG will need to address this issue as well. The real problem can be defined as including as much information as possible in the space allowed. (Reiter, 2000)

Size constraints are not always an easy attribute to capture. In many cases, the true "size" can only be determined after it has been processed through a presentation system. Changes to fonts, type size and other variable can be applied in such a way that the same amount of text can fit it many different physical forms.

Reiter points out that NLG community has generally accepted the notion that NLG must be done in stages or modules. Each module supports it's own specific function within the NLG processes. Typically the output of one module is cascaded into the input of the next. This type of structure is often referred to as pipelining. That is, there is a single flow, or "pipe" of information. One of the drawbacks of this type of structure is the lack of feedback between modules. Once a module has finished its task, the subsequent module cannot "request changes" to it's input. Although this may seem like a large problem, Reiter is quick to point out that every system they had reviewed used such a structure. The problem is usually handled in differing ways using size and space optimizers at one or more locations within the output pipeline.

Machine Learning in Discourse Planning

Limited work with machine learning and NLG has been performed to date. Much of it is used in the content determination phase. There has been some work done that focuses on the discourse planning phase. The machine learning approach was applied to a system that was required to generate descriptions about museum exhibits from a database of facts and short fragments of text. The system uses supervised learning algorithms to "learn" the ordering of important facts (Androutsopoulus, Dimitromanolaki 2003).

Data was collected from a database that contained information about 50 museum exhibits. The system was based on a fixed number of facts being conveyed in all generated texts, in this case 6 facts. The system would then be required to be able to use any 6 arbitrary facts

from the database to generate the texts. The combinations of 6 facts per exhibit were computed and obviously incorrect sets were discarded (such as sets that were missing key, required information). As the authors note, this would have normally been the work of the content determination module. The sets of 6 facts were then presented to a domain expert for ordering. These ordered fact sets were then used as the training sets for the machine learning algorithms. In particular, they system experimented with the k-nearest neighbor algorithm with k=1 as well as the C4.5 algorithm (Androutsopoulus, Dimitromanolaki 2003).

Results were very encouraging. Overall, the C4.5 algorithm seemed to perform slightly better than the k-nearest neighbor, but both algorithms performed significantly better than the base cases. Here are the contrasting examples given in the paper by Androutsopoulus and Dimitromanolaki (2003):

> *Human Annotator:* This exhibit is a portrait. It is made of marble and portrays Alexander the Great. It was created during the Hellenistic period, but what we see in the picture is a roman copy. Today it is located in the archaeological museum of Thassos.
>
> *C4.5 Fact Ordering:* This exhibit is a portrait. It portrays Alexander the Great and was created during the Hellenistic period. It is made of marble. What we see in the picture is a roman copy. Today it is located at the archaeological museum of Thassos.

Evolutionary Algorithms and Content Planner Construction

Duboue and McKeown used a genetic-search process to try to create an automatic way of specifying the tree-structured content planner used by their MAGIC system. The MAGIC system generates post-cardiac surgery reports or briefings based on the raw data that was collected in the

operating room. This is already a fully functional system, but the goal was to see if the genetic-search could improve upon the manually created content planner (Duboue, McKeown n.d.).

In their experiment, they defined the population of supposed solutions (or chromosomes in genetic terminology) as a tree that represented one possible content planner. Each of these chromosomes had an associated fitness value that could be calculated. This value determined how well the content planner matched the desired output based on externally specified requirements such as mandatory inclusion of certain facts. Fitness was also determined by evaluating the text that could be generated from the genetically altered content planner when compared against actual physicians briefings (Duboue, McKeown n.d.).

The experiment was run with an initial population of 2000 chromosomes (content planners). After each iteration of mutations, the worst 25% of the chromosomes as defined by the fitness functions were discarded. This 25% was then replaced with mutations based on the remaining chromosomes. This was repeated for 20 generations and took 8 days, 14 hours of total CPU time to complete! At the completion, the best chromosome that was created was compared against the original, manually created content planner that had been in use with the MAGIC system (Duboue, McKeown n.d.).

Evaluation was then done comparing the best chromosome produced by the simulation, as well as the initial 2000 chromosomes that were the starting point, against the manually create version. Findings were evaluated by using a function, that when evaluated, gives values closer to 0 to imply more similarity to the manually created content planner. Similarity analysis was performed yielding an average value of 3.08 for the 2000 initial un-tuned content planners, but a value of 1.16 for the best chromosome produced by the genetic algorithm (Duboue, McKeown

n.d.). The system had indeed “learned” how to improve its output, although it was not as good as the manually created version.

Experiments such as this are yielding information that may be used for future NLG systems that could automatically learn how to generate domain specific texts based on sample data and pre-generated texts.

Explanation-based Learning

The explanation-based learning (EBL) describe by Gunter Neumann (1997) is the idea of transforming many different explanations (solutions) that come from a problem solver into more compact and generalized forms. In particular, Neumann explains a way to automatically extract sub-grammars for control and speeding up natural language generation.

The system that was developed can be viewed as an inverse parsing solution. It uses structures that capture semantic meaning and maps their relationships as input, and then maps this into texts using a grammar and lexicon. He used a typed feature-based language for constraint-based grammars called TDL. This structure was key in allowing for finding similar patterns allowing for compacting and generalizing the grammar. Also, minimal recursive semantics (MRS) were used in the representation of the semantic information.

Training was accomplished by taking input sentences and generating the corresponding MRS. This MRS was then evaluated and generalized into a more basic form. This was then turned into a template for future sentence generation. As the system received more training information, it could generate additional MRS, repeating its calculations to determine if it had seen this general pattern before. A generative grammar could be built in this fashion using a large enough corpus. It is beyond the scope of this paper to give the full details, see Neumann (1997) for complete information on this system.

Neumann (1997) created a fully operational system based on these principals. On a training set of 179 sentences, it was able to recognize and then generate the corresponding generative grammar in around 1 second per sentence. This was estimated by the author as being a 10 to 20-time decrease in the time required to perform this step as compared to a human expert.

Examples Of Uses Today

Domain knowledge representation is key in determining what is to be said. Legal document drafting systems are being developed that do not use the standard template based approach. Instead, the information can be stored in a database and then recalled by a NLG to create unique documents by making inferences and grouping similar ideas based on topics (Callaway, Lester, Mott, 1999).

A more recent problem to be worked on by NLG is content summary. This is particularly useful with the World Wide Web. Small abstracts that retain the meaning of the original page are useful for search results and research. This would be the equivalent of electronically “reading” a web page and the generating an abstract about the page. This would be a summary of the page, written from scratch, using the information that was deciphered from the original page.

Many other systems have been created, such as automatic textual forecasts generated from weather maps and meteorological data, the generation of patient-friendly medical information and textual summarizations of statistical data (Dale, Reiter 1997).

Future Work

The field of NLG is only in its infancy. There are many different areas that all need explored in order to create the Holy Grail of systems that can generate natural language for any domain. The problem of connecting NLU to NLG is also an interesting area of study. Logically,

the two fields of study need to work harmoniously as it takes both types of systems to create something that is capable of not only “writing”, but also being able to “read”.

The introduction of the ideas that AI techniques can be used in various ways to help the machine learn on it’s own is a topic that deserves more attention. There is a wealth of information contained in countless texts around the world. It should be possible to take this information and use it to teach a machine how to write. It will be interesting to see in a few years what computers start “talking” about.

References

Androutsopoulos, I., Dimitromanolaki, A. (2003) Learning to order facts for discourse planning in natural language generation. Retrieved September 1, 2004, from http://arxiv.org/ftp/cs/papers/0306/0306062.pdf

Bateman, J., (January 2003). What is KPML? Retrieved September 4, 2004, from http://www.fb10.uni-bremen.de/anglistik/langpro/kpml/kpml-description.htm

Bleam, T., Doran, C., Palmer, M., Stone, M., Webber, B. (30 April 2001). Microplanning with communicative intentions: The SPUD system. Retrieved August 30, 2004, from http://ernie.ecs.soton.ac.uk/opcit/cgi-bin/pdf?id=oai%3AarXiv.org%3Acs%2F0104022

Branting, L. K., Callaway, C. B., Lester, J. C., Mott, B. W. (1999). *Integrating discourse and domain knowledge for document drafting*. ACM.

Cote, D., Moulin, B. (1990). *Refining Sowa's conceptual graph theory for text generation*. ACM.

Dale, R., Essers, V. (n.d.). *Choosing a surface realiser: Exploring the differences in using KPML/NIGEL and FUF/SURGE*. Microsoft Research Institute. Retrieved August 24, 2004, from http://www.ics.mq.edu.au/~rdale/publications/papers/1998/pricai98.pdf

Dale, R., Eugenio, B. D., Scott, D. (1998). *Introduction to the special issue on natural language generation*. Association for Computational Linguistics.

Dale, R., Reiter, E., (20 May 1997). Building applied natural language generation systems. Retrieved August 30, 2004, from http://citeseer.ist.psu.edu/cache/papers/cs/29943/http:zSzzSzwww.mri.mq.edu.auzSz~rdalezSzpublicationszSzpaperszSz1997zSzjnle97.pdf/reiter97building.pdf

Elhadad, M. (30 July 1997). *FUF/SURGE*. Retrieved August 30, 2004, from http://www.cs.bgu.ac.il/surge/

Neumann, G., (1997). Applying explanation-based learning to control and speeding-up natural language generation. *European chapter meeting of the ACL: Proceedings of the eighth conference on European chapter of the Association of Computational Linguistics.* (pp. 214 - 221). Retrieved September 11, 2004, from http://0-portal.acm.org.novacat.nova.edu/citation.cfm?id=979645&coll=ACM&dl=ACM&CFID=24228203&CFTOKEN=34934431#citings

Norvig, P., Russel, S. (2003). Learning from observations. *Artificial intelligence: A modern approach* (pp. 649-677). Upper Saddle River, NJ: Pearson Education, Inc.

Reiter, E. (2000). *Pipelines and size constraints*. Association for Computational Linguistics: Squibs and discussions.

Single-sentence natural language generation: Pennman. (n.d.) Retrieved September 5, 2004, from http://www.isi.edu/natural-language/penman/penman.html