Natural Language Engineering **3** (1): 57–87. Printed in the United Kingdom
© 1997 Cambridge University Press

Building applied natural language generation systems

EHUD REITER
Department of Computing Science, University of Aberdeen, Aberdeen AB24 3UE, Scotland
(*e-mail:* ereiter@csd.abdn.ac.uk)

ROBERT DALE
Microsoft Research Institute, Macquarie University, Sydney, NSW 2109, Australia
(*e-mail:* Robert.Dale@mq.edu.au)

(*Received 19 December 1996; Revised 19 May 1997*)

Abstract

In this article, we give an overview of Natural Language Generation (NLG) from an applied system-building perspective. The article includes a discussion of when NLG techniques should be used; suggestions for carrying out requirements analyses; and a description of the basic NLG tasks of content determination, discourse planning, sentence aggregation, lexicalization, referring expression generation, and linguistic realisation. Throughout, the emphasis is on established techniques that can be used to build simple but practical working systems now. We also provide pointers to techniques in the literature that are appropriate for more complicated scenarios.

1 Introduction

Natural language generation (NLG) 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.[1] Natural language generation systems combine knowledge about language and the application domain to automatically produce documents, reports, explanations, help messages, and other kinds of texts.

In this article, we look at NLG from an applied system-building perspective. We describe the tasks that must be performed by a language generation system, and

[1] Our focus here is on the production of text, but work in natural language generation is also concerned with systems that can communicate using speech. Many of the techniques we discuss here carry across to such systems; however, discussion of the particular problems that arise and the solutions that are appropriate in the production and synthesis of human-like speech is beyond the scope of this article.

discuss possible algorithms and supporting representations for performing each task. We also suggest techniques, often based on corpus analysis, that can be used to acquire the various kinds of knowledge needed in order to build such NLG systems. Prior to consideration of these technical issues, we discuss when NLG technology is likely to be appropriate, and when alternative or simpler techniques may provide a more appropriate solution. Throughout, our focus is on the use of well-established techniques that can be used to build simple but practical working systems today; we also provide pointers to ideas in the research literature that are likely to be relevant to the development of more sophisticated NLG systems in the future. Our intention is that the overview we present here should be useful both for software developers who are considering using NLG techniques in their systems, and for researchers interested in developing applied NLG technology.

The article is structured as follows. Section 2 briefly summarises some of the many ways in which NLG can be used, and discusses when NLG is and is not appropriate. Section 3 discusses requirements analysis and system specification. Section 4 surveys the different tasks that NLG systems perform, and describes some of the architectures that can be used to perform these tasks. Sections 5, 6 and 7 provide a more detailed look at the system components in an architecture that distributes these tasks across the three areas of text planning, sentence planning and linguistic realisation. Finally, Section 8 presents a summary and some concluding remarks, and provides some pointers to the research literature on natural language generation.

2 Using natural language generation

2.1 Applications of natural language generation

The most common use of natural language generation technology is to create computer systems that present information to people in a representation that they find easy to comprehend. Internally, computer systems use representations which are straightforward for them to manipulate, such as airline schedule databases, accounting spreadsheets, expert system knowledge bases, grid-based simulations of physical systems, and so forth. In many cases, however, these representations of information require a considerable amount of expertise to interpret. This means that there is often a need for systems which can present such data in an understandable form to non-expert users. When the best presentation of the data is in English, Spanish, Chinese, or some other human language, NLG technology can be used to construct the presentation system.

For example, NLG techniques can be used to:

- generate textual weather forecasts from representations of graphical weather maps, as in Goldberg *et al.* (1994);
- summarise statistical data extracted from a database or spreadsheet, as in Iordanskaja *et al.* (1992);
- explain medical information in a patient-friendly way, as in Buchanan (1995) and Cawsey *et al.* (1995);

- describe a chain of reasoning carried out by an expert system, as in Swartout (1983); and
- produce answers to questions about an object described in a knowledge base, as in Reiter *et al.* (1995).

This list of possible uses is only indicative, and is by no means complete; many other applications are described in the research literature.

NLG technology can also be used to build *authoring aids*, systems which help people create routine documents. Many people spend large chunks of their time producing documents, often in situations where they do not see document production as their main responsibility. A doctor, for example, may spend a significant part of her day writing referral letters, discharge summaries, and other routine documents; while a computer programmer may spend as much time writing text (code documentation, program logic descriptions, code walkthrough reviews, progress reports, and so on) as writing code. Tools which help such people quickly produce good documents may considerably enhance both productivity and morale.

Examples of using NLG as an authoring aid include:

- helping customer service representatives write letters for customers, as in Springer *et al.* (1991) and Coch (1996);
- helping engineers produce management summaries of design paths they have explored, as in McKeown *et al.* (1994);
- helping personnel officers produce job descriptions, as in Caldwell and Korelsky (1994); and
- helping technical authors produce instructions for using software, as in Paris *et al.* (1995).

Again, this list is indicative only.

2.2 When are natural language generation techniques appropriate?

Our concern in this article is with the development of practical NLG systems to meet current real needs. In this context, one has to bear in mind that developing a system based on NLG techniques is not always the best way to fulfil a client's needs. For example, in some cases, the information that needs to be delivered may be best presented graphically rather than textually. In other cases, text is the best presentation, but solutions based on the simple mail-merge facilities found in most word processors may suffice, and there may be no need to use more complex NLG techniques. In still other cases, the best solution is to hire a person to write documents or to explain things to users. Which approach is most appropriate in any given circumstance depends on a number of factors including the type of information being communicated, the amount of variation needed in the texts to be produced, and the volume of text to be produced. We briefly discuss some of these issues below.

2.2.1 Text versus graphics

In many situations, pictures, schematic diagrams, maps, charts and plots can be used to communicate information more effectively or more efficiently than text. Software developers need to consider whether text, graphics, or some mixture of the two is the best way to fulfil the user's information needs.

There are no hard and fast principles for deciding upon when information should be presented textually, and when it should be presented graphically. It depends on the type of information being communicated, and also on the expertise of the people reading the generated texts. The papers collected in Maybury (1993) are a good source for further reading in this area, while Kosslyn (1994) is a good general study of what graphics can and cannot do.

From a practical perspective, a good way to determine whether text or graphics should be used is to examine existing documents used to present the information. This is not foolproof, of course, because there is no guarantee that the way things happen to be done now is the best way possible. However, in many areas the task of information presentation has become a well-developed art, and so it is worth seeking out best practice in the area. In some cases the choice is dictated not by which medium is most effective, but rather by legal requirements, convention, or distribution constraints. For example, the use of graphics will not be an option if the material is to be delivered via teletext or a slow Internet link.

2.2.2 Natural language generation versus mail-merge

Natural language generation techniques are not the only way to generate text on a computer. Most operational software systems that generate text do so with the kind of mail-merge technology found in Microsoft Word and other popular document creation packages. While the simplest mail-merge systems just insert input data into pre-defined slots in a template document, more complex mail-merge systems are essentially programming languages that allow the output text to vary in arbitrary ways depending on the input data.

From a theoretical perspective, there is no difference in the functionality that can be implemented with NLG techniques and with complex mail-merge systems; both are Turing-equivalent computational systems. Indeed, it can be argued that mail-merge systems are a kind of NLG technology, being just one point on a scale of complexity and sophistication with full-blown NLG techniques lying at the other end of the scale.

From a practical perspective, we still have a very limited understanding of when NLG techniques are and are not needed. However, various authors have pointed out that

- NLG techniques can produce higher-quality texts than mail-merge, especially in applications where there is a lot of variation in the output texts Springer *et al.* (1991); Coch, (1996); and

- NLG systems can be easier to update in applications where it is necessary to regularly change the content or structure of the generated documents (Goldberg *et al.,* 1994).

2.2.3 Natural language generation versus human authoring

Software systems are expensive to build, and developers need to demonstrate that developing such a system is a better solution than hiring (and training) someone to manually write the documents the NLG system is supposed to produce. A large part of this decision is based on economics: is it cheaper to hire and train a person, or to create and maintain the software? The economic decision will usually depend largely on the volume of text produced. An NLG system which produces millions of pages of text per year will be easier to justify as cost-effective than a system which produces only hundreds of pages of text per year.

Cost is not the only factor, however. In some cases it is technically impossible to generate the required texts with current NLG technology; in these cases, manual production is the only option. On the other hand, in some cases NLG techniques may be preferred over manual document creation because they increase accuracy and reduce updating time, or because they guarantee conformance to standards (Reiter *et al.,* 1995).

3 Requirements analysis and system specification

The first step in building any type of software system, including a natural language generation system, is to perform a requirements analysis and, from this, to produce an initial system specification. We will not provide a survey of standard requirements analysis techniques here; these are covered in any good software engineering textbook (see, for example, Pressman, 1994). In general, software engineering methodologies that stress iterative development and prototyping usually work better for NLG systems than waterfall-type models. This is because the technology is still relatively immature, and hence it is difficult to predict in advance all the implications of requirements or design decisions.

In the rest of this section, we describe an approach to requirements analysis where the developer uses a collection of example inputs and associated output texts to describe to users the system she proposes to build. We believe that in most cases it is easier to discuss functionality with users by showing such examples than by discussing NLG in a more abstract way, especially since many users have no previous experience with NLG technology. We call this collection of input and output data a *corpus*, and this approach a *corpus-based approach.* A good corpus is also a very useful resource for knowledge acquisition when specific NLG modules are being designed, as will be discussed later in this paper. Note that while the corpora used in many natural language analysis projects consist solely of collections of example texts, corpora in NLG projects usually contain examples of system inputs as well as examples of (corresponding) output texts.

There has been relatively little published in the research literature on the use of corpora in building NLG systems, but see McKeown *et al.* (1994) for a description of the use of a corpus in developing the PLANDOC system. Reiter *et al.* (1997) discuss some problems with corpus-based approaches.

3.1 Assembling an initial corpus of output texts

The first step in carrying out a corpus-based requirements analysis is to create an *initial corpus* of human-authored texts and, where appropriate, their associated inputs. In the simplest case, the initial corpus can be created by using archived examples of human-authored texts. A business letter corpus can be based on real letters sent out in the past, for example; while a weather report corpus can be based on real reports written in the past. As far as possible, the corpus should cover the full range of texts expected to be produced by the NLG system: it should include boundary and unusual cases as well as typical cases.

If no human-authored examples of the required texts currently exist, the best strategy is often to ask domain experts to write examples of appropriate output texts. For example, an initial corpus for a patient information system could be created by asking doctors to write examples of good texts for specific patients that they know. For subsequent analysis of these texts, it can be very useful to record the expert 'thinking aloud' while writing some of the texts, if he or she agrees to this (see Reiter *et al.*, 1997).

3.2 Creating a target text corpus

In many cases, the NLG developer will want to modify the content of the initial corpus. This can happen for many reasons, including the following:

- It may be technically impossible, or prohibitively expensive, to automatically generate the texts in the initial corpus, often because the required input data is not available in a usable form; this is discussed further in Section 3.3.
- The texts may appear suboptimal, and open to improvement. In such cases, the NLG developer, who may have a better understanding of effective writing than some of the domain experts, may wish to suggest improvements to the texts. This of course needs to be discussed with the users and domain experts, especially since there may be domain-specific writing constraints or conventions of which the NLG developer is not aware.
- Different experts may suggest very different texts corresponding to the same input data. Conflicts between experts are, unfortunately, a common occurrence in many knowledge acquisition exercises. In many cases, conflicts can be resolved simply by bringing them to the attention of the experts and asking them as a group to decide on what the best text is for the given input data (Reiter *et al.,* 1997).

The result of all these changes is a set of texts which characterises the output that will be generated by the NLG system; we will refer to this as the *target text corpus.*

There are 20 trains each day from Aberdeen to Glasgow. The next train is the Caledonian Express; it leaves Aberdeen at 10am. It is due to arrive in Glasgow at 1pm, but arrival may be slightly delayed because of snow on the track near Stirling.

Thank you for considering rail travel.

Fig. 1. A sample required output from a rail travel information system.

It can take a considerable amount of effort to build a target text corpus with which both the developers and the users feel comfortable. However, this is time well spent, and a good corpus makes it much easier to develop a good NLG system.

3.3 Analysing the information content of corpus texts

One important step in the creation of the target text corpus is the analysis of the information content of the texts in the initial corpus. In particular, the developer needs to identify parts of the human-authored corpus texts which convey information which is not available to the NLG system.

In the remainder of this article, to make things concrete we will use a simple example based on a rail travel information system.[2] Suppose that the initial corpus of human-generated texts contains the text shown in Figure 1, generated in response to the inquiry *When is the next train to Glasgow?* Furthermore, assume that the input data consists (in addition to the actual inquiry) of a train scheduling database that lists departure and arrival times (both planned and predicted) of trains at all stations in the network, and the name (if any) of each train.

Information-content analysis requires classifying each sentence or clause of a corpus text into one of the following categories:

Unchanging text: a textual fragment that is always present in the output texts. An example of such a constituent in Figure 1 is the closing sentence *Thank you for considering rail travel.*

Directly-available data: text that presents information that is available directly in the input data or an associated database or knowledge base. Examples of such constituents in Figure 1 are the clauses *The next train is the Caledonian Express*, *it leaves Aberdeen at 10am*, *it is due to arrive in Glasgow at 1pm* and *arrival may be slightly delayed.*

Computable data: text that presents information that can be derived from the input data via some computation or reasoning. An example of this in Figure 1 is *There are 20 trains each day from Aberdeen to Glasgow*; this can be generated by selecting and counting appropriate records about train journeys in the scheduling database.

[2] This scenario, and the example we will use throughout this article, does not correspond to a real-world case study; we have compressed a number of phenomena into a simple example so that we can demonstrate a range of issues in the limited space available. Our example is partially motivated by the growing commercial interest in spoken-dialogue train-information systems (such as the system described in Aust *et al.* (1995)), although as mentioned earlier, we will not address speech-related issues in this article.

Unavailable data: text that presents information which is not present in or derivable from the input data. An example of this in Figure 1 is *because of snow on the track near Stirling*.

Unchanging text fragments are of course the easiest kind to generate; indeed they can simply be inserted in the text as canned strings. Textual constituents that present directly-available data texts pose no problems from an information perspective, although of course they present other difficulties for the NLG system. For textual constituents that present computable data, a decision has to be made as to whether the results are worth the cost involved, or whether the results fall into the category of unjustified 'bells and whistles'.

Texts that present unavailable data texts cause the most problems. They are of course impossible to generate: no matter how sophisticated an NLG system is, it cannot include information in its output that is not present in its input. In our experience, they are also, unfortunately, fairly common in human-authored texts. There are a number of solutions in principle to the problem of unavailable data, including making more information available to the NLG system, changing the target text corpus to eliminate the affected parts of the texts, and (if the NLG system is being used as an authoring aid for a human author) expecting the human to write such texts. Which solution is best of course depends on the application in question.

4 Architecture and components of an NLG system

Once a target text corpus and other requirements analysis documents have been produced, the developer can then start designing the NLG system. In Section 4.1 below, we enumerate the tasks that NLG systems perform; in Sections 4.2 and 4.3, we go on to discuss possible architectures and representations for supporting these tasks.

4.1 NLG tasks

The task of a natural language generation system can be characterised as mapping from some input data to an output text. However, as with most computational processes, it is useful to decompose this task into a number of more finely characterised substeps. In the context of automatic text generation, the nature of this decomposition is one of the major differences between NLG systems and systems based on mail-merge technology.

There is room for debate as to what the appropriate subtasks in language generation should be. However, it is probably true to say that, within the natural language generation community, a consensus has arisen that there are six basic kinds of activity that need to be carried out in going all the way from input data to a final output text. Note that this does not mean that an NLG system needs six modules; most systems use a computational architecture where one module simultaneously performs several tasks, as we discuss in Section 4.2.

These six basic activities are as follows.

4.1.1 Content determination

Content Determination is the process of deciding what information should be communicated in the text. We will describe this process as one of creating a set of *messages* from the system's inputs or underlying data sources; these messages are the data objects then used by the subsequent language generation processes. Both the message-creation process and the form and content of the messages created are highly application-dependent. Generally, the message creation process largely consists of filtering and summarising input data, and the messages created are expressed in some formal language that labels and distinguishes what we might think of as the *entities*, *concepts* and *relations* in the domain. We use these terms here in a relatively intuitive fashion, best made clear by example:[3] in our rail travel information system, we can view specific trains, places and times as entities; the property of being the next train as a concept; and notions such as departure and arrival as relations between trains and times.

Some examples of messages corresponding to the text shown in Figure 1, together with English glosses, are as follows.[4]

(1) a.
$$\begin{bmatrix} \text{message-id: msg01} & \\ \text{relation: IDENTITY} & \\ \text{arguments:} & \begin{bmatrix} \text{arg1: NEXT-TRAIN} \\ \text{arg2: CALEDONIAN-EXPRESS} \end{bmatrix} \end{bmatrix}$$

b. The next train is the Caledonian Express.

(2) a.
$$\begin{bmatrix} \text{message-id: msg02} & \\ \text{relation: DEPARTURE} & \\ \text{arguments:} & \begin{bmatrix} \text{departing-entity: CALEDONIAN-EXPRESS} \\ \text{departure-location: ABERDEEN} \\ \text{departure-time: 1000} \end{bmatrix} \end{bmatrix}$$

b. The Caledonian Express leaves Aberdeen at 10am.

[3] From a philosophical point of view, much could be (and has been) written about the ontological status of these categories. These issues are beyond the scope of the present work.

[4] Here, we represent each message as an attribute–value matrix. Each describes some relation that holds between those entities or concepts specified as the arguments of that relation. Note that messages do not necessarily correspond one-to-one to sentences; similarly, it is not necessarily the case that the domain entities, relations and concepts that occur here will correspond one-to-one to lexical items in the resulting text.

(3) a.

$$\begin{bmatrix} \text{message-id: msg03} & \\ \text{relation: NUMBER-OF-TRAINS-IN-PERIOD} & \\ \text{arguments:} & \begin{bmatrix} \text{source: ABERDEEN} \\ \text{destination: GLASGOW} \\ \text{number: 20} \\ \text{period: DAILY} \end{bmatrix} \end{bmatrix}$$

b. There are 20 trains each day from Aberdeen to Glasgow.

4.1.2 Discourse planning

Discourse planning is the process of imposing ordering and structure over the set of messages to be conveyed. A text is not just a random collection of pieces of information: the information is presented in some particular order, and there is usually an underlying structure to the presentation. In the simplest possible terms, this is akin to a story having a beginning, a middle and an end; but most documents have much more discernible structure than this. Good structuring can make a text much easier to read: that this is so can easily be demonstrated by trying to read a version of a newspaper story where sentences and paragraphs have been randomly reordered.

The result of discourse planning is usually a represented as a tree structure, along the lines of that shown in Figure 2; here, the leaf nodes of the tree are individual messages, and the internal nodes specify how messages are grouped together and related to each other. The clustering decisions made in the tree will have an impact on the determination of sentence and paragraph boundaries in the resulting text. In some cases, the internal nodes also specify *discourse relations* between their children: in this example, the NUMBER-OF-TRAINS-IN-PERIOD and IDENTITY messages are placed in a SEQUENCE relationship, and the DEPARTURE message is an taken to be an ELABORATION of the IDENTITY message. The notion of discourse relations is discussed further in Section 5.

4.1.3 Sentence aggregation

Sentence aggregation is the process of grouping messages together into sentences. In our current example, sentence aggregation could combine the IDENTITY and DEPARTURE messages into a single sentence, which would be realised as *The next train, which leaves at 10am, is the Caledonian Express.*

Aggregation is not always necessary – each message can be expressed in a separate sentence – but in many cases good aggregation can significantly enhance the fluency and readability of a text. In principle, aggregation techniques can be used to form paragraphs and other higher-order structures as well as sentences; however, this is a less well understood process, and will not be discussed further here.

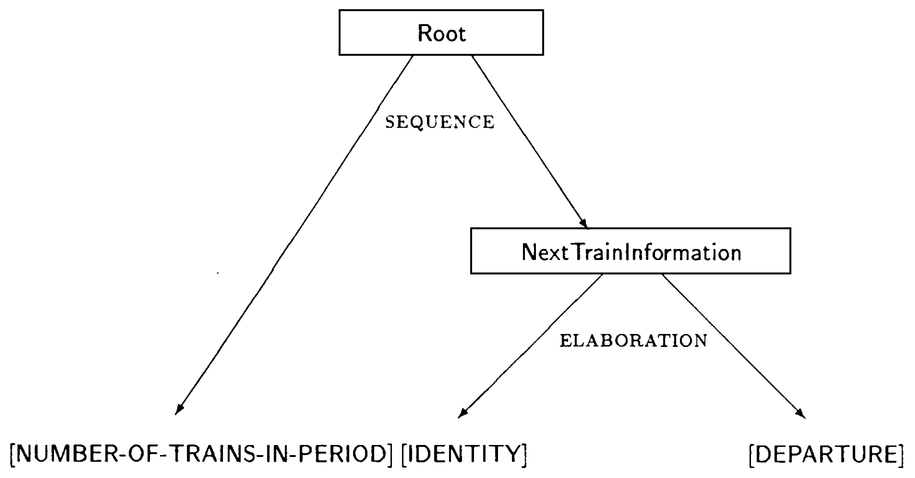

Fig. 2. An example discourse structure tree.

4.1.4 Lexicalization

Lexicalization is the process of deciding which specific words and phrases should be chosen to express the domain concepts and relations which appear in the messages. This covers questions such as how the event represented in the DEPARTURE message should be expressed: the words *leave* and *depart* are both possibilities.

In many cases, lexicalization can be done trivially by hard-coding a specific word or phrase for each domain concept or relation; for example, we might simply specify that a DEPARTURE message should always be expressed by the word *leave*. In some cases, however, fluency can be improved by allowing the NLG system to vary the words used to express a concept or relation, either to achieve variety or to accommodate subtle pragmatic distinctions (for example, *depart* is perhaps more formal than *leave*). Lexicalization is especially important, of course, when the NLG system produces output texts in multiple languages.

4.1.5 Referring expression generation

Referring expression generation is the task of selecting words or phrases to identify domain entities. For example, the text in Figure 1 uses the referring expressions *the Caledonian Express* and *it* to refer to the domain entity CALEDONIAN-EXPRESS.

Referring expression generation is closely related to lexicalization, since it is also concerned with producing surface linguistic forms which identify domain elements. However, unlike lexicalisation, referring expression generation is usually formalised as a discrimination task, where the system needs to communicate sufficient information to distinguish one domain entity from other domain entities. This generally requires taking account of contextual factors, including in particular the content of previous communications with the user (generally referred to as the DISCOURSE

HISTORY). For example, whether or not *it* can be used to refer to CALEDONIAN-EXPRESS depends on what other objects have been mentioned in previous sentences in the text.

4.1.6 Linguistic realisation

Linguistic realisation is the process of applying the rules of grammar to produce a text which is syntactically, morphologically, and orthographically correct. For example, a linguistic realisation process may decide to express the NUMBER-OF-TRAINS-IN-PERIOD message above as the following sentence:

(4) There are 20 trains each day from Aberdeen to Glasgow.

In this example, the syntactic component of the realiser has decided to add the function words *from* and *to* to mark those parts of the sentence which specify the train's source and destination; the morphological component has produced the plural form *trains* of the root word *train*; and the orthographic component has capitalised the first word of the sentence and added a full stop at the end of the sentence.

4.2 NLG architectures

There are many ways of building a system that performs the tasks we have just described. The simplest approach is to build a separate module for each task, and connect these modules via a one-way pipeline. In such an architecture, the content determination module first decides on all the messages to be included in the text; the discourse planning module then organises these messages into a discourse structure tree; and so on. At the other extreme, we might not have separate modules at all, but simply represent each task as a set of constraints or axioms, and put all the constraints into a general constraint-solver or theorem-prover whose job is to determine some solution that maximally satisfies the constraints; see Appelt (1985) for an example of this approach.

From a pragmatic perspective the most common architecture in present-day applied NLG systems, and the one we will emphasize in this paper, is a three-stage pipeline with the following stages:[5]

Text planning: this stage combines the content determination and discourse planning tasks described above. This reflects the fact that in many real applications, it can be difficult to separate these two activities.

Sentence planning: this stage combines sentence aggregation, lexicalization, and referring expression generation. This combination is not universally accepted in the NLG field; for example, Matthiessen (1991) argues that lexicalisation should be combined with linguistic realisation. Nevertheless, most applied NLG systems have chosen to combine these three tasks into one stage, and that is the approach we will take in this article.

[5] See Reiter (1994) for a discussion of the emerging consensus in NLG architectures.

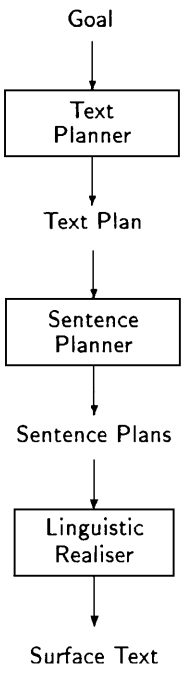

Fig. 3. Architecture of a natural language generation system.

Linguistic realisation: as described above, this task involves syntactic, morphological, and orthographic processing.

Figure 3 diagrammatically shows the three-stage pipelined architecture.

4.3 Intermediate representations

Once the overall architecture of an NLG system has been decided, the other main design issue is how the inputs and outputs of the different stages should be represented. The initial input to the system is, of course, application-dependent, and the final output is text (perhaps with some logical or physical mark-up for presentation purposes, such as HTML tags). However, the NLG developer must also specify what internal representation is passed from the text planning stage to the sentence planning stage, and from the sentence planning stage to the linguistic realisation stage. We will call the Text Planner output the *Text Plan*, and the Sentence Planner output for any one sentence a *Sentence Plan.*

4.3.1 Text plans

Text plans are usually represented as trees whose leaf nodes specify individual messages, and whose internal nodes show how messages are conceptually grouped together. By gathering together related material, this grouping imposes constraints on the scope of subsequent sentence planning operations, and perhaps also on the possible locations for paragraph boundaries. As we noted above, the text plan may specify discourse relations between nodes.

There are a variety of ways of representing the messages that form a text plan's leaf nodes, including logical formulae and templates. Probably the most common strategy in current applied NLG systems is to represent messages in as similar a form as possible to the representation used for sentence plans, which we turn to below. For example, if the NLG system uses templates for sentence plans, it might also use templates in the text plan leaf nodes. The templates in the text plan may then contain parameters represented by pointers into the domain knowledge base, whereas, when the sentence plans are constructed, these pointers will be replaced by words or phrases.

4.3.2 Sentence plans

A wide variety of mechanisms and notations for sentence plans have been proposed and used in the literature, of which the most common are probably templates and what we might call *abstract sentential representations.*

Template systems represent sentences as boilerplate text and parameters that need to be inserted into the boilerplate. Classic template systems simply insert the parameter into the boilerplate without doing any further processing. Some newer systems, however, perform limited linguistic processing as well (see, for example, Geldof and Van de Velde, 1997). For example, such systems may enforce number agreement by choosing the appropriate morphological variant of a word; this typically requires adding some annotations or structure to the words in the boilerplate.

The other common way of representing sentences plans is to use an abstract representation language which specifies the content words (nouns, verbs, adjectives and adverbs) of a sentence, and how they are related. We will refer to such representations here as *abstract sentential representations.* From the perspective of linguistics, these usually convey information at a roughly similar level to the 'deep syntactic' representation in Mel'čuk's *meaning-text theory* (Mel'čhuk, 1988), as discussed in Section 7.4; a number of other grammatical theories offer not dissimilar representational levels.

One of the most popular abstract representation languages for sentence plans is SPL (Sentence Planning Language) (Kasper, 1989). The SPL representation for the sentence *There are 20 trains each day from Aberdeen to Glasgow* would be something like that shown in Figure 4. The SPL representation characterises the sentence by means of a number of named attributes and their values, and allows values themselves to consist of named attributes and their values. Note that SPL allows certain variations in the text to be easily specified; for example, the future-tense version of the sentence *There will be 20 trains to Glasgow* can be produced by adding a single `(:tense future)` attribute–value pair to the SPL shown in Figure 4.

The pros and cons of template versus abstract sentential representations are further discussed in Section 7.

```
(S1/exist
 :object (O1/train
         :cardinality 20
         :relations ((R1/period
                       :value daily)
                      (R2/source
                       :value Aberdeen)
                      (R3/destination
                       :value Glasgow))))
```

Fig. 4. An SPL expression

5 Text planning

In this section, we describe in more detail the two tasks we take to be involved in the process of text planning. Section 5.1 addresses the subtask of content determination, and Section 5.2 addresses the subtask of discourse planning.

5.1 Content determination

5.1.1 The task

Content determination is the task of determining what information should be communicated in the text. As above, we will describe this process as one of creating a set of *messages* from the system's input data and any databases or knowledge bases that provide domain and background information. In many applications, content determination is also affected by a *user model*, which may specify (amongst other things) the user's expertise level, the task he or she is trying to carry out, and the previous interactions he or she has had with the NLG system; see Reiter *et al.* (1995) for an example of this.

It is hard to generalise about content determination because it is very dependent on the details of the target application. A rail travel information system has a very different content determination module from a system that helps customer service representatives write business letters. But one theme that is common to many content determination systems is the need to filter, summarise and otherwise process input data. To return again to our example of a rail travel information service: suppose the user asks about trains from Aberdeen to Leeds, and there are no trains listed in the database which stop in both of these points. In such a case the system should not report that no such trains exist, but rather should search the database for a sequence of two or more trains that will take the user from Aberdeen to Leeds.

5.1.2 Deep reasoning approaches

Many researchers have suggested that content determination should be formalised in terms of reasoning about what information users need to accomplish their goals. For example, Allen and Perrault (1980) suggest that an NLG system should use *plan recognition* techniques to determine what plan the user is executing (for example, a plan to go to Glasgow by train), and then analyse this plan to determine what information the user needs to complete it (perhaps an indication of the departure platform as well as the departure time). This information then becomes the content of the generated response.

Because superficially similar requests for information can be used in different situations to execute quite different plans, plan recognition requires sophisticated reasoning and a considerable amount of knowledge about the world and the user. Although there is a considerable body of research literature on plan recognition, at the time of writing we are not aware of any current applied NLG systems which actually use this technique in deciding what to say.

5.1.3 Domain-specific approaches

Most current applied NLG systems base content determination on domain-specific rules acquired from domain experts: this amounts to implementing rules of thumb as to what information should be included in general categories of situations. The results are somewhat less flexible than the deep reasoning approaches described above, and will sometimes produce less than perfect results. However, for practical systems given the current state of the art, these approaches are much easier to implement. They also have some other advantages, including the following:

- It is easy to accommodate bureaucratic and legal concerns that would otherwise be difficult to motivate from first principles. For example, in some medical contexts it may be important for legal reasons to explicitly list all possible side-effects of a drug, even if from the user's perspective a summary statement such as *One in a million patients taking this drug suffer severe side effects* would be adequate.
- Using domain-dependent methods, it is easier to produce computer-generated documents that are similar to existing human-authored documents. This may help users accept the NLG system; indeed, they may not even be aware that the NLG system exists.

The process of working out rules for content determination is in many ways similar to the kinds of Knowledge Acquisition (KA) task faced by the designer of an expert system (see, for example, Scott *et al.* 1991), and most expert system KA techniques can probably be adapted for this task. Perhaps the currently most popular technique for acquiring content rules is to analyse texts in a corpus. One variant of this process is to select some representative example texts from the target text corpus, and perform the following operations:

1. Break each text down into its constituent sentences, and then break each sentence down into a set of information-conveying phrases.
2. Relate each phrase to the source data, as described in Section 3.3.
3. Group phrases into classes of similar messages.
4. Characterise which classes of messages appear in which types of texts, and try to determine the conditions under which messages appear. The result of this step might be a collection of rules of the form 'a text of type t will contain a message of type m under conditions $c_1, c_2, \ldots c_n$'.

5. Discuss this analysis with domain experts, and modify it accordingly. In some cases, the analysis may suggest modifying the target text corpus as well as the content rules.
6. When satisfied with the analysis, repeat the process with a larger selection of texts from the target text corpus.

Many of the details here (for example, determining what constitutes an 'information-conveying phrase') will depend on the circumstances of the particular application.

A very general point is that an essential component of successful knowledge acquisition is becoming familiar with the domain and with the data (in this case, the corpus). You are going to have to spend a considerable amount of time learning about the domain, poring over and analysing the corpus, and discussing your observations with the domain experts via the process described above. The amount of effort required should not be underestimated; however, there is no easy alternative if your goal is to build a system that delivers appropriate results.

5.2 Discourse planning

5.2.1 The task

Discourse planning is the task of structuring the messages produced by the content determination process into a coherent text. In effect, while content determination involves using knowledge about what information should be communicated (for example, whether the departure time of the next train should be included in the text to be generated), discourse planning uses knowledge about how these messages should be organised into a text (for example, the decision to start with a summary and then to give information about specific trains, rather than the other way around).

The output of the discourse planner is a text plan (Section 4.3.1), which is a tree whose leaf nodes are messages, and whose internal nodes specify how these messages are conceptually grouped together. The text plan may also specify discourse relations that hold between messages or groups of messages. These indicate how the text fragments are related. For example, consider the following text:

(5) a. I like to collect old Fender guitars.
 b. My favourite instrument is a 1951 Stratocaster.

Here, the second sentence is providing an example of the proposition expressed in the first; we might say that a discourse relation of ELABORATION or EXEMPLIFICATION holds between the two sentences. On the other hand, consider an example like the following:

(6) a. I like to collect old Fender guitars.
 b. However, my favourite instrument is a 1991 Telecaster.

Here, we might say that a discourse relation of CONTRAST or EXCEPTION holds between the two sentences. Note that particular cue words are often used to signal the particular discourse relations that reside in the text; for example, *however* is commonly used to signal contrast. Discourse relations can hold between groups of messages as well as individual messages. For example, a text might contain an entire paragraph whose function is to provide EVIDENCE for some claim made earlier in the text.

There is no consensus in the research literature on what specific discourse relations should be used in an NLG system. Probably the most commonly used set is that suggested by *Rhetorical Structure Theory* (RST) (Mann and Thompson, 1988), although many developers modify this set to cater for idiosyncrasies of their particular domain and genre. For a general discussion of different ways of classifying discourse relations, see Maier and Hovy (1993).

5.2.2 Planning-based approaches

In an ideal world, a discourse planner should be able to take an arbitrary assortment of content messages and organise these into a coherent whole, perhaps with occasional requests to the content determination process for additional messages that might add to the text's coherence. A sizeable body of work, originating at the Information Sciences Institute at the University of Southern California, has focused on operationalising this idea by representing discourse relations using artificial-intelligence-style planning operators, with preconditions that specify what needs to be the case before the relation can be used, and effects that specify the impact on the reader's beliefs of using the relation. A planning mechanism can then be used to build up a discourse plan in a sophisticated way from the available messages given some top level goal. See Hovy (1993) and Moore and Paris (1993) for descriptions of systems that use this approach.

At present, we have only very limited understanding of the different discourse relations that can appear in texts, what the precise effect of these relations is, and when they can be used. This fact, in combination with the computational expense of planning-based approaches and the large amounts of knowledge they require, means that approaches based on these ideas are not widely used in current real-world NLG systems.

5.2.3 Schema-based approaches

Planning-based approaches of the kind just described hold out promise for the development of very general text planning engines. However, for the development of practical systems in limited domains, it is generally easier to adopt a somewhat more domain-specific approach. It is often the case that the texts required of a given application will conform in structure to a relatively small number of patterns. These patterns can be determined by careful analysis of the target text corpus, and by talking to domain experts. If a corpus-based approach is used to acquire both content determination and discourse planning rules, it is natural to combine the two knowledge acquisition efforts. This combined process is very similar to that discussed in Section 5.1.3, except that you should analyse the structure of texts (what order messages occur in, and what discourse relations or cue words connect messages) as well as what messages they contain. Similarly, when obtaining feedback from experts, you can ask for comments on text structure as well as comments on what information should be included.

The most popular technique used to implement this kind of discourse planning makes use of what are called *schemas* (McKeown, 1985; Kittredge *et al.*, 1991). Each schema is a pattern that specifies how a particular text plan should be constructed using smaller schemas or atomic messages, and also the discourse relation that holds between these constituents. For example, a text plan that provides a response to the query *When is the next train to x?* might be constructed using the following two schemas:

```
(7)     a. Inform-Next-Train-Schema ⟶
                     Sequence(Message:NUMBER-OF-TRAINS-IN-PERIOD,
                              Next-Train-Information-Schema)
        b. Next-Train-Information-Schema ⟶
                     Elaboration(Message:IDENTITY, Message:DEPARTURE)
```

A schema-based discourse planning mechanism can be started after the content determination process has already decided what messages to include in the text. It is more common, however, for the schema system to call the content determination system 'on demand' whenever it needs a particular type of message. In the above example, for instance, the NLG system might start by executing the Inform-Next-Train-Schema, which would then request the construction of a NUMBER-OF-TRAINS-IN-PERIOD message; once this has been retrieved, the subsequent call to instantiate the Next-Train-Information-Schema will result in requests for an IDENTITY message and a DEPARTURE message. In this way, content determination is interleaved with discourse planning, with the discourse planning process in overall control.

Most schema-based systems allow general programming constructs, such as local variables and conditional tests, to be included in the schema. Indeed, schema languages can be thought of as special-purpose programming languages, and they are often implemented as macro or class libraries on top of an underlying programming language. To date, most NLG developers have created their own schema languages; however, as experience in the construction of NLG systems grows, we may see the development of standardised schema languages.

6 Sentence planning

In this section, we describe in more detail the three tasks we take to be involved in the process of sentence planning. Section 6.1 addresses the subtask of sentence aggregation; Section 6.2 looks at lexicalization; and Section 6.3 addresses the subtask of referring expression generation.

6.1 Sentence aggregation

6.1.1 The task

Sentence aggregation is the task of combining two or more messages into one sentence. In terms of the representations we are using here, the sentence aggregation process takes as input a tree-structured text plan whose leaf nodes are messages, and produces as output a new text plan whose leaf nodes are combinations of messages that will eventually be realised as sentences. The aggregation system must decide both what messages to aggregate to form each sentence, and also what syntactic mechanism should be used to combine the messages.

As a simple example of sentence aggregation, consider again the example in Figure 2, where the text plan contains three messages, roughly corresponding to the following three clauses:

(8) a. there are 20 trains each day from Aberdeen to Glasgow
b. the next train is the Caledonian Express
c. the Caledonian Express leaves at 10am

Given this input, a few (by no means all) possible aggregation strategies are:

1. No aggregation: realise each message as a single sentence. The resulting text, assuming some process that performs pronominalization, will be as follows:

(9) There are 20 trains each day from Aberdeen to Glasgow. The next train is the Caledonian Express. It leaves at 10am.

2. Combine the last two messages using a relative clause. This strategy produces either of the following texts, depending on where the relative clause is attached:

(10) a. There are 20 trains each day from Aberdeen to Glasgow. The next train is the Caledonian Express, which leaves at 10am.
b. There are 20 trains each day from Aberdeen to Glasgow. The next train, which leaves at 10am, is the Caledonian Express.

3. Combine the first two messages with the conjunction *and*. This results in the following text:

(11) There are 20 trains each day from Aberdeen to Glasgow, and the next train is the Caledonian Express. It leaves at 10am.

4. Combine all three messages using a conjunction and a relative clause. This results in the following text:

(12) There are 20 trains each day from Aberdeen to Glasgow, and the next train is the Caledonian Express, which leaves at 10am.

Although aggregation does not change the information content of a text, it does contribute to readability and fluency: compare the first and second alternatives above, for example.

6.1.2 Types of sentence aggregation

There are several kinds of sentence-formation aggregation, including the following.

Simple conjunction: the simplest form of aggregation is to use a connective such as *and* to produce a sentence plan which communicates more than one message.

Ellipsis: if the two messages being aggregated have a common constituent, it may be possible to elide the repeated constituent. For example, messages which might be independently realised as *John went to the bank* and *John deposited £50* can be aggregated to produce a sentence plan that is realised as *John went to the bank and deposited £50*. See Quirk and Greenbaum (1972) for a discussion of the types of ellipsis that are possible in English.

Set formation: if the messages being grouped together are identical except for a single constituent, it may be possible to replace these with a single sentence plan that contains a conjunctive constituent. For example, three messages that would otherwise be realised independently as *John bought an apple*, *John bought a banana* and *John bought a pear* can be combined into a sentence plan realised as *John bought an apple, a banana and a pear*. Sometimes there are alternative, usually domain-dependent, ways of describing sets. For example, instead of explicitly enumerating the set *Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, and Saturday*, the set of elements could be replaced by a single concept corresponding to the expression *every day of the week*.

Embedding: this involves embedding one clause as a constituent of another. Relative clauses (see Section 6.1.1 for examples) are perhaps the most common form of embedding.

In some cases it may be necessary to add cue words such as *also* or *as well* to increase fluency where aggregation was possible, but nevertheless, was not performed; see Dalianis and Hovy (1996).

6.1.3 Creating appropriate aggregation rules

The most difficult aspect of aggregation is deciding which of the numerous potential aggregations should be performed. Potential sources of aggregation rules include psycholinguistic knowledge on reading comprehension, writing handbooks and the advice of experienced practitioners. Unfortunately, the advice given in such sources is often too vague to be directly computationally implementable.

An alternative approach to developing aggregation rules is to try to determine what types of aggregation occur in a corpus, and then propose rules which result in similar aggregations being performed by the NLG system. This has the advantage of naturally incorporating sublanguage or genre constraints: for example, newspaper articles typically have longer sentences than technical manuals.

One general constraint used by many aggregation modules is to only aggregate nodes that are siblings in the text plan. Given the text plan in Figure 2, for example, this would rule out the third and fourth possibilities in Section 6.1.1. A weaker version of this rule is to also allow all descendants of an internal node to be aggregated; this would allow the fourth alternative, but still reject the third.

6.2 Lexicalization

6.2.1 The tasks of lexicalization and referring expression generation

Lexicalization and the generation of referring expressions are tasks which focus on the problem of choosing which words should be used to pick out or describe particular domain concepts or entities. For our present purposes, we will take the view that lexicalization is the task of choosing words to express a concept or relation, while referring expression generation is the task of choosing words to identify a particular domain entity.

Consider the following message:

(13)
$$\left[\begin{array}{ll} \text{message-id: msg02} & \\ \text{relation: DEPARTURE} & \\ \text{arguments:} & \left[\begin{array}{l} \text{departing-entity: CALEDONIAN-EXPRESS} \\ \text{departure-location: ABERDEEN} \\ \text{departure-time: 1000} \end{array}\right] \end{array}\right]$$

This message mentions one domain relation, DEPARTURE, and three domain entities: the train CALEDONIAN-EXPRESS, the location ABERDEEN, and the time 1000. This message could be realised in a number of ways including the following:

- We could use the word *departs* for the DEPARTURE relation, the phrase *The Caledonian Express* for CALEDONIAN-EXPRESS, the name *Aberdeen* for ABERDEEN, and *10am* for the time of departure. This would result in the following sentence:

(14) The Caledonian Express departs from Aberdeen at 10am.

- Alternatively, we could use the word *leaves* for DEPARTURE, the pronoun *it* for CALEDONIAN-EXPRESS, the deictic expression *here* for ABERDEEN, and the temporal expression *in five minutes* for the departure time. This would result in the following sentence:

(15) It leaves here in five minutes.

As the above examples show, lexicalization involves finding a word or phrase that can communicate a concept such as DEPARTURE to the hearer, while the generation of referring expressions involves finding a noun phrase that identifies an entity such as CALEDONIAN-EXPRESS to the hearer in the current discourse context.

Like aggregation, good lexicalization and referring expression generation are essential for producing texts that are fluent and easily readable. A poorly lexicalized text may still be understandable, but readers will need to work harder to extract its meaning.

Although lexicalization and referring expression generation are related tasks, and in principle NLG systems should probably use an integrated algorithm which carries out both tasks, in practice these tasks have generally been discussed separately in the research literature. We return to the task of referring expression generation in Section 6.3, and focus first on lexicalization.

6.2.2 Approaches to lexicalization

The most common model of lexicalization is one where the lexicalization module converts an input graph whose primitives are domain concepts and relations into an output graph whose primitives are words and syntactic relations. Lexicalization researchers have developed powerful graph-rewriting algorithms which use general 'dictionaries' that relate domain primitives and linguistic primitives; see, for example, Nogier and Zock (1992) and Stede (1996). The dictionaries usually allow a single domain primitive to be mapped into a linguistic structure that includes multiple words (for example, DEPARTURE might be mapped to *is the departure time of*). They also allow a single word to correspond to a structure containing several domain entities (for example, $\langle \mathsf{CHILD}(x) \wedge \mathsf{FEMALE}(x) \rangle$ can be mapped to the single word *girl*).

Graph-rewriting lexical choice is most useful in multilingual generation, when the same conceptual content must be expressed in different languages. The technique handles quite naturally some kinds of lexical divergences between languages. For instance, the English sentence *He swam across the river* can be expressed in French as *Il a travers la rivière a la nage* (literally *He crossed the river by swimming*):[6] these two messages are informationally equivalent, but not word-for-word isomorphic. These two sentences can be generated from a common input by a graph-rewriting lexicalization system that has both French and English dictionaries.

Most monolingual NLG systems perform lexical choice using methods which have their origins in the work of Goldman (1975), who used decision trees to encode simple choice rules that are used to vary how concepts are expressed according to various factors. For example, decision trees can be used to perform the following tasks:

- Select different words to realise a domain primitive to add variety to a text. For example, we may want to alternate between *leave* and *depart* as realisations of DEPARTURE. This follows the common writing advice that words should not be overused.
- Select different words to realise a concept in different context. For example, the discourse relation CONTRAST can be expressed with *but* within a sentence, but is better expressed as *however* if it relates messages in different sentences.
- Select different words based on stylistic parameters. For example, the concept MALE-PARENT is probably best realised as *father* in formal contexts, but may be better realised as *dad* in some informal contexts.

6.3 Referring expression generation

Referring expression generation is most commonly viewed as a description task, with the goal of including enough information in the description to enable the hearer to unambiguously identify the target entity. The amount of information needed to

[6] This example is from Stede (1996).

do this will depend on the current discourse context. For example, consider the italicised phrases in the following text:

(16) The next train is *the Caledonian Express*. *It* leaves at 10am. Many tourist guidebooks highly recommend *this train*.

Here, the entity CALEDONIAN-EXPRESS is initially referred to by name (*the Caledonian Express*), which is a standard way of introducing a named object into a discourse. The second reference to CALEDONIAN-EXPRESS uses the pronoun *it*; this is again a standard way of referring to an object which has been mentioned recently. The final reference is a definite description (*this train*), which is a standard way of referring to an entity when it has already been introduced, but where the context rules out the use of a pronoun. Generating each form of reference raises different issues.

Initial introduction: relatively little research has been carried out on the generation of initial references to objects. Two common strategies are to simply give the name of the object (if it has a name), as was done in the CALEDONIAN-EXPRESS example above; and to describe the physical location of an object, as in *the wrench in the blue toolbox*.

Pronouns: pronoun use, and in particular, pronoun interpretation, has been extensively studied in the natural language processing literature. There is less work on pronoun generation. From a practical perspective, a simple algorithm that works surprisingly well in many cases is to use a pronoun to refer to an entity if the entity was mentioned in the previous clause, and there is no other entity in the previous clause that the pronoun could possibly refer to.[7] This is a fairly conservative algorithm, in the sense that it will not generate a pronoun in many cases where one could be used; but it has the advantage that it does not often inappropriately insert a pronoun. Of course, in some cases pronouns must be used for syntactic reasons; this is usually handled by the linguistic realisation module.

Definite descriptions: more research has been done on generating definite descriptions; see, for example, Dale (1992) and Dale and Reiter (1995). From a practical perspective, a simple but useful algorithm is to begin by including in the description a base noun describing the object (for example, *train*), and then (if necessary) add adjectives or other modifiers to distinguish the target object from all other objects mentioned in the discourse. For example, if the discourse has just discussed the Caledonian Express and no other trains, then *the train* can be used to refer to the Caledonian Express. However, if the discourse has also mentioned *the 1015am train from Aberdeen to Edinburgh*, then a definite description for CALEDONIAN-EXPRESS should include extra information to distinguish this from the other train; so, we might build the description *the*

[7] Entities which might otherwise be considered potential referents can often be ruled out on easily-computable grounds: for example, *it* is not generally used to refer to people, and cannot be used to refer to plural entities.

Glasgow train. A corpus analysis can be used to determine which modifiers are most commonly added to definite descriptions in the application domain.

7 Linguistic realisation

7.1 The task

Linguistic realisation is the task of generating grammatically correct sentences to communicate messages. From a knowledge perspective, the realiser is the module where knowledge about the grammar of the natural language is encoded. Some simple examples of this syntactic and morphological knowledge are as follows:

Rules about verb group formation: the earlier stages of the generation process may simply specify a tense (for example, *past*, *present*, or *future*), an overall sentential form (for example, *question* or *imperative*), a polarity (for example, *negated*), and so forth. It is the job of the realiser to construct an appropriate verb group based on all these parameters. For example, if the basic message to be conveyed is concerned with the relationship between the concept NEXT-TRAIN and the domain entity CALEDONIAN-EXPRESS, we might have the following variants:

(17) a. Future Tense:
The next train will be the Caledonian Express.
b. Simple Present Tense, Negated:
The next train is not the Caledonian Express.
c. Past Tense, Question:
Was the [last] train the Caledonian Express?

Rules about agreement: English requires certain words to agree in grammatical number (*singular* or *plural*); for example, we say *A cat sleeps* but *Two cats sleep.* The realiser can automatically enforce these rules as it constructs a sentence.

Rules about syntactically required pronominalisation: in some cases, syntactic rules require pronouns to be used in sentences. For example, we say *John saw himself in the mirror* rather than *John saw John in the mirror* if the person John saw was John himself.

The above are a small sample of some of the 'peculiarities' of English which a realiser can take care of automatically. This enables the rest of the NLG system to work with a much simpler and cleaner model of the language, unencumbered by these details.

In the rest of this section, we will describe a few of the more popular approaches to linguistic realisation. Space prohibits an exhaustive review of all approaches to this task.

7.2 Realisation as the inverse of parsing

The process of linguistic realisation is sometimes viewed as the inverse of the parsing process found in natural language analysis systems. In a parsing system, a grammar is used to map from a surface sentence to a representation of the semantic content of that sentence. Under the inverse parsing model, the task of the realiser is to go in the opposite direction; that is, the realiser takes as input a representation of the semantic content of a sentence that is similar to the representations produced as output by parsers, and produces from this as output a surface sentence that expresses this semantic content.[8] A number of algorithms have been proposed for this task, of which the best known is the semantic-head-driven algorithm (Shieber *et al.*, 1990).

The inverse parsing approach in principle allows *bi-directional grammars* to be constructed. These grammars are declarative formulations of the correspondences between semantic structures and syntactic structures, which can be used to produce mappings in either direction. Bi-directional grammars are theoretically very elegant. They also have the practical advantage of making it easier to build dialogue systems which can understand the same range of syntactic and lexical phenomena that they are able to produce, which is likely to help users of such systems.

However, there are problems with making the inverse parsing approach work in practice. In particular, the approach assumes that the input to an NLG realiser is similar to the output of a parser in a natural language analysis system. However, the representations that are naturally produced by the pre-realisation stages of NLG (that is, text planning and sentence planning), such as SPL, are in fact quite different from the representations currently required by the post-parser components of most natural language analysis systems. Busemann (1996) discusses this issue and other problems with inverse parsing.

7.3 Systemic grammars

A popular approach to linguistic realisation that does have its basis in natural language generation research is motivated by Systemic Functional Linguistics (SFL) (Halliday, 1985). As suggested by its name, SFL is primarily concerned with the functions of language; a systemic functional grammar describes how the functions may be mapped into or expressed by surface forms. This view is, not surprisingly, very appropriate in the context of NLG.

Systemic grammar emphasises choice-making: the central task is not viewed as the finding of a chain of grammar rules which convert an input structure into a sentence, but rather that of making a series of increasingly fine-grained choices which taken together determine the syntactic characteristics of the sentence being constructed. In a linguistic realisation component based on systemic grammar, these choices are often characterised as queries posed to the intended semantic content and the wider environment in order to determine what function words should be added, how words should be ordered, and so on. For a detailed exposition of the use of systemic

[8] A common terminological misperception is that *generation* is the inverse of parsing, but as we have seen in this paper, generation involves very much more than this. If we are to seek ways of comparing work in language generation and work in language analysis, then NLG as a whole corresponds to a multi-level language analysis model that culminates in plan recognition and the incorporation of the results into a world model.

grammar in natural language generation, see Matthiessen and Bateman (1991). Bateman (1996) describes KPML, a linguistic realiser based on systemic grammar which has been used in several NLG projects.

An alternative representation of the ideas in SFL can be found in Elhadad and Robin's (1996) SURGE, a unification-based systemically-oriented grammar of English that uses the Functional Unification Formalism (FUF) as its underlying mechanism.

Both FUF/SURGE and KPML are in the public domain.[9] These are complicated and sophisticated general purpose systems, with a correspondingly steep learning curve required in order to use them successfully. For simple applications, simpler approaches are often more appropriate.

7.4 Meaning-text grammars

Another popular approach to linguistic realisation in applied NLG systems is the use of grammars based on Meaning-Text Theory (MTT) (Mel'čuk, 1988). MTT makes use of a type of dependency grammar, and it divides the realisation process up into several stages: a full implementation of MTT would have seven distinct levels of representation, but most applied systems based on MTT use fewer. The REALPRO system (Lavoie and Rambow, 1997), for example, takes as input a deep syntactic structure (which is similar in content to an SPL expression) and converts this into text in the following stages:

- a *deep syntactic component* adds function words, and specifies syntactic relations between words;
- a *surface syntactic component* linearises the words;
- a *morphological component* inflects the words;
- a *graphical component* adds punctuation and formatting; and
- a *formatter* converts the result into HTML, RTF or ASCII.

MTT realisers have been used by many applied NLG projects, including FoG (Goldberg *et al.,* 1994), AlethGen (Coch, 1996) and GhostWriter (Marchant *et al.,* 1996). The REALPRO system described above is available as a commercial product, and may be licensed free of charge to qualified academic institutions.[10] Like FUF/SURGE and KPML, REALPRO is a complex system; making effective use of the system will require some effort being expended on learning about the system and the relevant aspects of MTT.

7.5 Templates

Some applied natural language generation systems do not perform syntactic realisation at all; instead, the content determination process directly specifies messages

[9] For information on obtaining FUF/SURGE, see `http://www.cs.bgu.ac.il/surge`. For information on obtaining KPML, see `http://www.darmstadt.gmd.de/publish/komet/kpml.htm`.

[10] For more information on REALPRO, see email `realpro@cogentex.com`.

as text templates. This has an impact on a number of the tasks we have described so far:

- Content determination and discourse planning proceed as described above in Section 5, but the end result is a text plan whose leaves are templates (which may include linguistic annotations, as mentioned in Section 4.3.2). For example, a text plan might have the following as a leaf node:

(18) $\langle x$ [root=depart, number=number(x)] at $y\rangle$
$\langle x$=CALEDONIAN-EXPRESS, y=1000$\rangle$

- Sentence aggregation basically works as described in Section 6.1, but it may not be able to use certain types of aggregation such as ellipsis. This is because templates do not specify the sentential content at a sufficiently abstract level to permit the appropriate manipulations. Some types of aggregation are still possible with templates, however, as discussed in Geldof and Van de Velde (1997).
- Lexicalization is usually not performed, and instead the content determination system chooses templates which contain appropriate words to describe domain concepts. If it is important to not overuse words so as to maintain variety in a text, it may be necessary to have several templates for the same basic message and to put in place some mechanism for choosing between them.
- Referring expression generation generates slot fillers for the template. In the above example, for instance, the referring expression generation mechanism might determine that the concept CALEDONIAN-EXPRESS should be realised by the pronoun *it*. It can be difficult to generate good referring expressions in template systems because the NLG system does not know what entities are mentioned in the canned portion of the template.
- Realisation may take care of agreement (e.g. *departs* instead of *depart*) and other aspects of morphology and orthography, but it does not perform any syntactic processing.

The template approach often makes sense when only limited syntactic variability is required in the output texts. For example, if all the sentences that are to be generated are in the simple present tense, then there is no need for a complex realisation mechanism that can generate sentences in other tenses. Another advantage of the template approach is that, for domain experts, templates are usually easier to understand than mechanisms that manipulate more complex syntactic structures. This may assist in the knowledge acquisition task.

8 Conclusions

In this paper, we have:

- discussed the pros and cons of building natural language generation systems;
- discussed some of the techniques that can be used in determining the requirements to be met by such systems;

- presented an overview of the tasks that natural language generation systems need to attend to; and
- described in some detail one particular architectural model that accommodates these tasks.

There has been insufficient space here to describe all the design issues involved in building NLG systems, and all the technical problems that arise. For further information on these issues, the reader may wish to look at the wider NLG research literature. Much of this is to be found in books arising from various international and European workshops on Natural Language Generation that have been held over the last ten years: see Kempen (1987), McDonald and Bolc (1988), Zock and Sabah (1988), Dale *et al.* (1990), Paris *et al.* (1991), Dale *et al.* (1992), Horacek and Zock (1993) and Adorni and Zock (1996). Unfortunately, there are many workshops which have not resulted in subsequent publications. Some of the papers presented at these workshops are available over the World Wide Web from the *Computation and Language* server (`http://xxx.lanl.gov/archive/cmp-lg`); otherwise, obtaining information about papers presented at workshops may require contacting the author or someone else who was present at the workshop. Joining the Association for Computational Linguistics' Special Interest Group in Natural Language Generation (SIGGEN) is a good way to keep in touch with upcoming events in the field. The SIGGEN Web site (`http://www.cs.bgu.ac.il/siggen`) gives general information about SIGGEN, plus conference announcements, papers, jobs and other news of interest to the NLG community.

If this article has been successful in its goal, it will have provided the reader with a better appreciation of what is involved in building an applied natural language generation system. We would be very interested to receive feedback from readers who try to make use of the work presented here.

Acknowledgements

The authors would like to thank the three anonymous reviewers of this article, the many other people who have commented on drafts of the article, and the attendees of the tutorial on *Building Applied Natural Language Generation Systems* presented at the 1997 Applied Natural Language Processing Conference in Washington DC, for comments and suggestions which have led to improvements in the material presented here.

References

Adorni, G. and Zock, M. (eds.) (1996) *Trends in Natural Language Generation: Lecture Notes in Artificial Intelligence*. Berlin: Springer.

Allen, J. and Perrault, C. (1980) Analyzing intention in utterances. *Artificial Intelligence* **15**: 143–178.

Appelt, D. (1985) Planning English referring expressions. *Artificial Intelligence* **26**: 1–33.

Aust, H., Oerder, M., Seide, F. and Steinbiss, V. (1995) The Philips automatic train timetable information system. *Speech Communication* **17**: 249–262.

Bateman, J. (1996) KPML Development Environment. *Technical Report*, IPSI, GMD, Darmstadt, Germany.

Buchanan, B., Moore, J., Forsythe, D., Carenini, G., Banks, G. and Ohlsson, S. (1995) An intelligent interactive system for delivering individualized information to patients. *Artificial Intelligence in Medicine* **7**: 117–154.

Busemann, S. (1996) Best-first surface realization. In *Proceedings of the 8th International Workshop on Natural Language Generation*, pp. 101–110.

Caldwell, D. and Korelsky, T. (1994) Bilingual generation of job descriptions from quasi-conceptual forms. In *Proceedings of the Fourth Conference on Applied Natural-Language Processing*, pp. 1–6. Association for Computational Linguistics.

Cawsey, A., Binsted, K. and Jones, R. (1995) Personalised explanations for patient education. In *Proceedings of the 5th European Workshop on Natural Language Generation*, pp. 59–74.

Coch, J. (1996) Evaluating and comparing three text production techniques. In *Proceedings of COLING-1996*.

Dale, R. (1992) *Generating Referring Expressions: Building Descriptions in a Domain of Objects and Processes*. Cambridge: MIT Press.

Dale, R., Mellish, C. and Zock, M., editors (1990) *Current Research in Natural Language Generation*. London: Academic Press.

Dale, R., Hovy, E., Rösner, D. and Stock, O., editors (1992) *Aspects of Automated Natural Language Generation: Lecture Notes in Artificial Intelligence*. Berlin: Springer.

Dale, R. and Reiter, E. (1995) Computational interpretations of the Gricean maxims in the generation of referring expressions. *Cognitive Science* **19**: 233–263.

Dalianis, H. and Hovy, E. (1996) Aggregation in natural language generation. In: G. Adorni and M. Zock, editors, *Trends in Natural Language Generation*, pp. 88–105. Berlin: Springer.

Elhadad, M. and Robin, J. (1996) An overview of SURGE: a reusable comprehensive syntactic realisation component. In *Proceedings of the 8th International Workshop on Natural Language Generation (Demos and Posters)*, pp. 1–4.

Geldof, S. and Van de Velde, W. (1997) An architecture for template based (hyper)text generation. In *Proceedings of the 6th European Workshop on Natural Language Generation*, pp. 28–37.

Goldberg, E., Driedgar, N. and Kittredge, R. (1994) Using natural-language processing to produce weather forecasts. *IEEE Expert* **9**: 45–53.

Goldman, N. (1975) Conceptual generation. In R. Schank (ed.), *Conceptual Information Processing*. New York: Elsevier.

Halliday, M. (1985) *An Introduction to Functional Grammar*. London: Edward Arnold.

Horacek, H. and Zock, M., editors (1993) *New Concepts in Natural Language Generation*. London: Pinter.

Hovy, E. (1993) Automated discourse generation using discourse structure relations. *Artificial Intelligence* **63**: 341–386.

Iordanskaja, L., Kim, M., Kittredge, R., Lavoie, B. and Polguère, A. (1992) Generation of extended bilingual statistical reports. *Proceedings of the Fifteenth International Conference on Computational Linguistics (COLING-92)*, volume 3, pp. 1019–1023.

Kasper, R. (1989) A flexible interface for linking applications to Penman's sentence generator. *Proceedings of the 1989 DARPA Speech and Natural Language Workshop*, pp. 153–158.

Kempen, G., editor (1987) *Natural Language Generation*. Martinus Nijhoff.

Kittredge, R., Korelsky, T. and Rambow, O. (1991) On the need for domain communication knowledge. *Computational Intelligence* **7**: 305–314.

Kosslyn, S. (1994) *Elements of Graphic Design*. New York: W. H. Freeman.

Lavioe, B. and Rambow, O. (1997) A Fast and Portable Realizer for Text Generation Systems. *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pp. 265-268.

Maier, E. and Hovy, E. (1993) Organising discourse structure relations using metafunctions. In: H. Horacek and M. Zock, editors, *New Concepts in Natural Language Generation*, pp. 69–86. London: Pinter.

Mann, W. and Thompson, S. (1988) Rhetorical structure theory: toward a functional theory of text organization. *Text* **3**: 243–281.

Marchant, B., Cerbah, F. and Mellish, C. (1996) The GhostWriter Project: A demonstration of the use of AI techniques in the production of technical publications. *Proceedings of Expert Systems 1996: Applications Stream*, pp. 9-25.

Matthiessen, C. (1991) Lexico(grammatical) choice in text generation. In: C. Paris, W. Swartout and W. Mann, editors, *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, pp. 249–292. Kluwer Academic.

Matthiessen, C. and Bateman, J. (1991) *Systemic Linguistics and Text Generation: Experiences from Japanese and English*. London: Pinter.

Maybury, M. (ed.) (1993) *Intelligent Multimedia Interfaces*. AAAI Press.

McDonald, D. and Bolc, L., editors (1988) *Natural Language Generation Systems*. Berlin: Springer.

McKeown, K. (1985) Discourse strategies for generating natural-language text. *Artificial Intelligence* **27**: 1–42.

McKeown, K., Kukich, K. and Shaw, J. (1994) Practical issues in automatic document generation. *Proceedings of the Fourth Conference on Applied Natural-Language Processing*, pp. 7–14.

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

Moore, J. and Paris, C. (1993) Planning text for advisory dialogues: capturing intentional and rhetorical information. *Computational Linguistics* **19**: 651–694.

Nogier, J-F. and Zock, M. (1992) Lexical choice as pattern-matching. *Knowledge-Based Systems* **5**: 200-212.

Paris, C., Vander Linden, K., Fischer, M., Hartley, A., Pemberton, L., Power, R. and Scott, D. (1995) A support tool for writing multilingual instructions. *Proceedings of Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1398–1404.

Paris, C., Swartout, W. and Mann, W., editors (1991) *Natural Language Generation in Artificial Intelligence and Computational Linguistics*. Kluwer Academic.

Pressman, R. (1994) *Software Engineering: A Practitioner's Approach*. McGraw-Hill.

Quirk, R. and Greenbaum, S. (1972) *A University Grammar of English*. Longman.

Reiter, E. (1994) Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? *Proceedings of the 7th International Workshop on Natural Language Generation*, pp. 163–170.

Reiter, E., Cawsey, A., Osman, L. and Roff, Y. (1997) Knowledge acquisition for content selection. *Proceedings of the 6th European Workshop on Natural Language Generation*, pp. 117–126.

Reiter, E., Mellish, C. and Levine, J. (1995) Automatic generation of technical documentation. *Applied Artificial Intelligence* **9**: 259–287.

Scott, A., Clayton, S. and Gibson, F. (1991) *A Practical Guide to Knowledge Acquisition*. Addison-Wesley.

Shieber, S.,van Noord, G., Pereira, F. and Moore, R. (1990) Semantic head-driven generation. *Computational Linguistics* **16**: 30–42.

Springer, S., Buta, P. and Wolf, T. (1991) Automatic letter composition for customer service. In *Proceedings of the Innovative Applications of Artificial Intelligence Conference (CAIA-1991)*, pp. 67–83.

Stede, M. (1996) Lexical options in multilingual generation from a knowledge base. In: G. Adorni and M. Zock, editors, *Trends in Natural Language Generation*, pp. 222–237. Berlin: Springer.

Swartout. W. (1983) XPLAIN: a system for creating and explaining expert consulting systems. *Artificial Intelligence* **21**: 285–325.

Zock, M. and Sabah, G., editors (1988) *Advances in Natural Language Generation: An Interdisciplinary Perspective* (two volumes). London: Pinter.