Flexible Natural Language Generation in Multiple Contexts

Caroline Cullen, Ian O'Neill, Philip Hanna

The Institute of Electronics, Communications and Information Technology (ECIT),
Queen's University Belfast, Northern Ireland.
{ccullen09, i.oneill, p.hanna}@qub.ac.uk

Abstract. We present a practical approach to Natural Language Generation (NLG) for spoken dialogue systems. The approach is based on small template fragments (mini-templates). The system's object architecture facilitates generation of phrases across pre-defined business domains and registers, as well as into different languages. The architecture simplifies NLG in well-understood application contexts, while providing the flexibility for a developer and for the system, to vary linguistic output according to dialogue context, including any intended affective impact. Mini-templates are used with a suite of domain term objects, resulting in an NLG system (MINTGEN – MINi-Template GENerator) whose extensibility and ease of maintenance is enhanced by the sparsity of information devoted to individual domains. The system also avoids the need for specialist linguistic competence on the part of the system maintainer.

Key words: Natural Language Generation, Spoken Dialogue Systems, Object-Orientation.

1 Introduction

Natural Language Generation (NLG) is concerned with the mapping between the semantic representation of a phrase and its corresponding natural language expression. The NLG component in a multi-domain spoken dialogue system must supply information to the user using the phrasing most appropriate to each of the system's domains. It must also present information in a manner appropriate to the context, taking into consideration the user's experience of working with the system, and the 'personality' that the system wants to adopt.

By abstracting similar concepts between domains and identifying modifications between domains, we created an ordered hierarchy of mini-templates and sparse domain-specific terms that forms the basis of an intuitive object-based toolkit. The toolkit is designed to enhance system usability for developers and end-users alike. Such an approach to NLG, where an output can be closely specified and can draw on ready-made object components, provides a practical solution for application developers who wish to create naturalistic output, but who are not specialist computational linguists and may be unfamiliar with conversational usage in the target application domains. Specialist terminology obtained from domain experts is stored

in such a way that, for a given dialogue context, an automated Dialogue Manager can access the appropriate term, at the correct level of specialisation. In due course we aim to develop an easily navigated GUI which will support NLG system maintenance by non-specialist developers.

This ability to dynamically match NLG capabilities to context becomes a more pressing requirement as speech-based dialogue systems attempt to emulate more closely human-to-human communications. The NLG component must be able to 'change tone', not only as the context changes in a discourse, but also as the relationship with the user evolves, within and between dialogue sessions - cf. GRETA [1] and the broad research into affect in computing being conducted by the EU network of excellence HUMAINE [2].

2 MINTGEN in Context

We have developed a test-bed application, MINTGEN (MINi-Template GENerator) which attempts to address the disadvantages of a number of established approaches to NLG:

1. *Canned Text* produces very natural utterances by printing a string of words 'as is'. These hand-crafted phrases are easy to create, but are inflexible.

2. *Template* or *Frame-Based* approaches slot data into specific locations in a message template. However, it is not easy to re-use templates across domains and applications and they are difficult to maintain as even a slight change to the output requires a large amount of recoding.

3. *Phrase-Based* approaches typically use phrase structure grammar rules to produce "generalized templates" at the sentence level and at the discourse level, resulting in a linguistic phrase-based NLG system which is generally quite adaptable. However, sensitivity towards the grammatical features of the target language is required.

4. *Feature-Based* approaches specify each sentence as a unique set of features e.g. question/imperative/ statement, positive/negative resulting in a system that is easily expanded by the addition of new features. However, the fine-grained linguistic detail required makes such a system difficult to maintain [3].

3 MINTGEN's Approach to NLG

The first three of the existing NLG approaches listed above would not, if used in isolation, provide the flexibility or the fluency required by an NLG module that is intended to cross between domains, registers and languages in real-time. The fourth and (to some extent) the third approach, while highly effective, are beyond the reach of most software engineers who, while sensitive to issues of natural language usage, are not a specialist in computational linguistics. Without the available services of such specialist staff, maintainability and extensibility of the system are problematic.

Register, *domain* and *language* were selected as vehicles through which to investigate our practical object-based approach to NLG. We provide hierarchies of

generic and domain-, register- and language-specific components for the NLG task – the more specialized components making use of existing generic functionality wherever possible. This middle-ground approach uses easily configurable components, sparsely populated tables and mini-templates that are small enough to be rearranged so that they can generate from the numerous semantic constructs that the Dialogue Manager (DM) may present. This form of generation avoids the need for a highly abstracted linguistic representation such as the Sentence Planning Language (SPF) [4].

Elements of the approaches outlined in Section 2 benefit the implemented NLG module: since the object-based generation module comprises a hierarchical, extensible set of components tailored to known domains, any new element to be added is typically a further specialization of an existing element in the basic 'toolkit'.

3.1 Multi-Context Capabilities

MINTGEN was designed to work flexibly in a variety of application contexts which include changes in domain, register and language, examples of which are given in Section 4.5. With the possible exception of language, these changes are likely to occur dynamically mid-dialogue. MINTGEN is not designed as a translating tool, although it could be manipulated for that purpose.

MINTGEN is intended to work with an updated version of the *Queen's Communicator* (QC) [5], but will work with any dialogue system whose Dialogue Manager conforms to the NLG's input specification. The QC already encompasses three application domains: Accommodation, Theatre and Cinema booking.

MINTGEN is a *cross-domain* system. By decomposing a domain into speech acts (e.g. requesting information, providing information etc.) and related domain terms (pertaining to hotel, airplane, cinema etc.), commonality and differences between domains are apparent, thus indicating which utterances are generic and which are domain-specific. Items within domains are grouped together by their characteristics in order to take advantage of their common attributes so that duplicate information is not repeated.

MINTGEN is capable of *multi-register* generation. As Poggi et al. [1] remark, "Words have formal and informal variants, connotations, positive or negative, tender or insulting nuances". As an exploration of how register is incorporated into the NLG input constructs and the subsequent effect on the generated phrase, normal, formal and informal registers are considered in the current implementation. We have made provision for the use of APML [6] to tag utterances that are intended to convey particular affective or emotional colouring.

Finally MINTGEN also has a *multi-lingual* capability. We currently generate from the abstracted semantic constructs used by the system's DM into two target languages: English and Irish, whose sentence structure and vocabulary are markedly different from each other [7] – a good test of the abstracted representations.

4 System Architecture

4.1 Description of Mini-Templates

When the conversational context changes, the NLG component has to provide the building blocks to cope with the change. Templates are broken down into fine-grained structures or 'mini-templates'. These mini-templates are references to phrasal fragments consisting of one or more words and come from either the Domain Terms Class hierarchy (4.2) where they realise a concept associated with the domain or the Speech Act hierarchy (4.4) in which case they are classified as being a phrase, statement, verb or question. See 4.5 for an example of the mini-templates selected by the sentence planning part of the module and a step through of how they are realised based on the request from the Dialogue Manager.

By providing slots for specific data input from the user, these reusable mini-templates provide a more adaptable and precise sentence than would be possible with conventional larger templates. If a new form of utterance is to be added, the system is extended with new mini-templates.

The template required to present *n* options to a user, and to ask them to choose one, is defined by the generic mini-templates [QuestionDesire] from the Speech Act hierarchy and [DomainVerb] from the Domain Terms Class hierarchy, taking the form:

$\{QDesire.register\}\{domain.Verb\}O_1 \ldots O_n$

where each option (O_x) enters the NLG module as an input parameter passed from the DM. If, for example, the choice is between some hotels, then the options will be hotel names and specialized domain terms relating to hotels (e.g. 'stay at' being the verb for the hotel domain) will be used in the mini-templates. This eventually results in a generated phrase such as: 'Do you want to stay at The Europa, The Hilton or The Ramada?'

4.2. Domain Terms Class Hierarchy

For the cross-domain NLG task, we have identified five domains in the context of a notional City Information Service. The domains in this instance are Accommodation, Transport, Entertainment, Places to Visit and Places to Eat. Inheritance enables us to capture cross-domain similarities, and divide high-level domains into more specialized sub-domains, as shown in the inheritance fragment in Figure 1. Each of these domains and sub-domains comprises sets of domain terms, in the appropriate language and register, which can be used with the corresponding domain-specific mini-templates.

A top-level Domain Terms class encapsulates the high level terminology necessary to describe concepts – e.g. cost, start time, end time, duration, location, number, name, etc. – within each of the modelled domains.

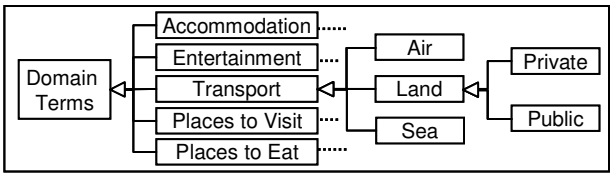

Fig. 1. Domain Terms Class Hierarchy.

Similarly, properties of the Transport Class, a subclass of Domain Terms, are inherited by each more specialized Transport class. However, some of the properties are overridden. Consider the generic, top-level concept of cost. This becomes cost/stay when in the Accommodation domain, or cost/journey in the Transport domain. Cost/stay can be further specialized to the domain-specific concept of price/night for a Hotel or price/week for an Apartment. This hierarchy of Domain Terms classes along with their associated properties and inheritance relationships determines the structure of the Domain Terms Store (DTS - See 4.5.f) The DTS contains the lexical information that enables the system to realize utterances in each of its domains. The DTS is stored as a database with each domain being a record which comprises its superdomain (the domain one level hierarchically above) and the potential to represent all generally applicable concepts. A blank entry denotes that the value is the same as that of its superdomain, which itself is also a record.

This lack of certain entries results in sparse database tables, a distinct advantage for system maintenance. A measure of this 'sparsity' has been taken (See 4.3). Domain-specific tables with additional concepts also exist and a similar process is in operation.

4.3 Domain Terms Store Sparsity

There are 9 main concepts in the Domain Terms Store which are applicable to all domains. There are 54 records in the store representing all of the domains and superdomains. This would result in 486 concept values in a densely populated store. However, by taking advantage of the repeated entries in the store which indicate the generic concepts that are inheritable down through the domains, we created a sparsely populated store, where values only need to be introduced at the level at which they are specialized – i.e. the presence of a value denotes that the corresponding value from its superdomain is overwritten. Table 1 lists for each of the 9 concepts the number of values which are present in the sparse store (maximum of 54), the number of those values that are unique and a percentage measurement of how populated the sparse store is in comparison to the densely populated one.

For the English Domain Term Store, there are 146 values represented out of a potential 486. This represents a 30% populated table i.e. - 70% of the values are inherited and thus do not need to be duplicated. Since the concept of location is almost always unique (41 times out of 54), if it is ignored, the sparsity improves, as there are 105 values out of a potential 432, resulting in a density of 24%, a great reduction in the information required.

Table 1. Number of Concept Values in Sparse Table.

Concept	No. of Values	No. of Unique Values	Sparsity (No.Values/54)
People	10	6	19%
Start Time	14	8	26%
End Time	12	8	22%
Location	41	39	76%
Duration Concept	19	15	35%
Duration Unit	10	5	19%
Cost Concept	20	15	37%
Cost Unit	10	9	19%
Domain Verb	10	10	19%
TOTAL	146	115	30 %

This sparse approach to representing concepts that apply to the Domain Terms means that if a new domain is added to the system, it needs to know what its superdomain is (i.e. the category into which it falls and whose properties it shares with other members of that domain). The new domain automatically inherits these properties unless the developer specifically overrides them. This shows the commonality between the domains and demonstrates that an object-based approach using generic objects is an appropriate model for the NLG module.

4.4 Speech Act Definition and Hierarchy

The speech act hierarchy defines the interface for the NLG module by describing the semantic input constructs. It defines the range of utterances that the system is expected to realize.

The NLG system makes use of three main *Speech Acts* that are basic to the City Information task:

- *Statements*: conversational system utterances not requiring a response e.g. greeting, thanks, goodbye …
- *Information Request*: the system elicits information from the user e.g. desire, number, location, duration …
- *Information Provision*: the system provides the user with information e.g. existence, distance, closing time …

The speech act hierarchy is refined to a level of specialization that corresponds with the top-level concepts in the Domain Terms hierarchy. Once the speech act is formulated, the appropriate specialized term is selected from the Domain Terms hierarchy – e.g. a 'start-time' Information Request causes a phrase to be generated that asks about arrival time in the Accommodation domain, departure time in the Transport domain and opening time in the Entertainment domain. See Figure 2 for a snapshot of the speech act hierarchy.

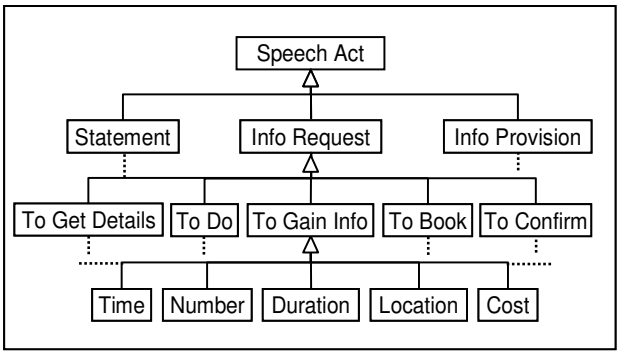

Fig. 2. Speech Act Hierarchy.

The speech act hierarchy influences the basic structure of the Template Object Store (TOS – see 4.5.d) by providing a framework around which various generalised and more specific mini-templates can be defined. Sibling templates in alternative registers and languages are also stored in the TOS. The Sentence Planner (4.5.c) selects the appropriate templates from the TOS according to the speech act, domain, register and language specified by the DM.

4.5 Components

The language generation module itself consists of seven units which are supported by two stores of information. This section outlines realisation of the module output for a given Dialogue Manager request.

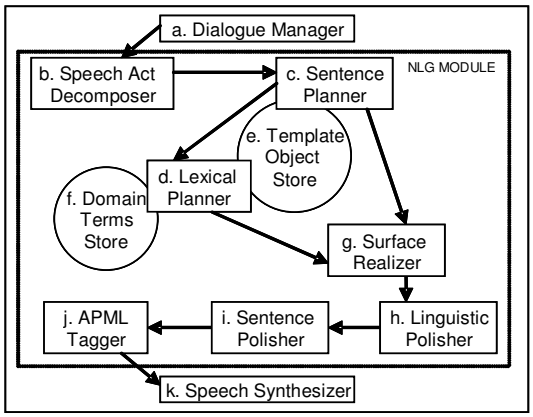

Fig.3. Components of NLG System.

a. **Dialogue Manager (DM**) – decides on abstracted semantic input constructs and passes them on to the NLG module in the form of:

`[speechAct][purpose][actName]`
`[specialism][domain][parameters]`
`[register][language]`

e.g. if the system wants to ask the how long a user wishes to stay at a hotel in a normal English register, the DM passes the following constructs to the NLG module:

DM: `[info_request][gain_info][duration][-][hotel]`
`[-][normal][•nglish]`

b. **Speech Act Decomposer (D)** – extracts the required information from the DM input constructs and stores it according to the specification in (a) above.

c. **Sentence Planner (SP)** – decides on the mini-templates (represented by entries within {}) that are to be selected from the TOS and the order in which they will be realised, e.g. For the DM's request in *a* above the SP decides that the following sequence of generic mini-templates is sufficient to realize the speech act:

SP: `Eng[{Qnumber.register}{domain.durationUnit(plural)}`
`{Qdesire.register}{domain.domainVerb(infinitive)}`
`{domain.definite}]`

SP passes any parameters to the SR along with the mini- template sequence. Any components that the SR cannot realize are passed first to the LP by the SP.

d. **Lexical Planner (LP)** – decides upon and retrieves the actual words to be generated from the DTS and TOS e.g.

LP retrieves `{Qnumber.english.normal}` and `{Qdesire.english.normal}` from the TOS, and `{hotel.english.durationUnit(plural)}` and `{hotel.english.domainVerb (infinitive)}` from the DTS.

e. **Template Object Store (TOS)** – a collection of speech act objects in the form of mini-templates arranged in hierarchical structure to support the SP.

TOS stores `{how many}` and `{do you want}` for the English, normal versions of `{Qnumber}` and `{Qdesire}`.

f. **Domain Terms Store (DTS)** – a collection of instantiated domain objects with the appropriately refined terms for each domain member. The DTS supports the Sentence Planner, Lexical Planner and the Surface Realizer. For `{hotel.english.durationUnit(plural)}` and `{hotel.english.domainVerb (infinitive)}`, the DTS returns `{night}` and `{stay at}` respectively. It is the LIP that later deals with the plural and infinitive versions of these phrases.

g. **Surface Realizer (SR) –** realizes NL form of templates and words by combining output from SP and LP

SR: `How many night (plural) do you want`
`stay at (infinitive) hotel (definite)?`

h. **Linguistic Polisher (LIP)** – deals with language specific elements e.g. plurals, agreements, masc./fem. Etc., so `{night (plural)}` becomes 'nights', `{stay at (infinitive)}` becomes `'to stay at'` and `{hotel (definite)}` becomes `'the hotel'`, thus resulting in the desired sentence:

```
LIP: How many nights do you want to stay at the hotel?
```

i. **Sentence Polisher (SEP)** – searches for and rectifies any grammatical issues which affect the whole sentence e.g. certain word combinations. SEP does not detect any further polishing to be done with this particular sentence.

j. **APML Tagger** – has the ability to apply affective mark-up to the sentence to be uttered based on speech acts and concepts in the input specification. After several misunderstood repetitions, the APML tagging can change to emphasise an element.

k. **Speech Synthesizer** – takes the well-formed sentence output from the NLG module and the APML tagging along with any other realisation mark-up and voices the utterance.

In order to demonstrate the multi-context capabilities of MINTGEN, Table 2 parts (i) to (iv) in the Appendix to this paper give examples of the output of each system component – from the DM's request constructs through to full sentence realization. Changing just one of the DM's request constructs (e.g. the construct representing register or language or domain) significantly alters the resulting generation, thus demonstrating the multi-register, multi-lingual and cross-domain capabilities of the system.

5 Conclusion

The NLG module MINTGEN as described in this paper has been designed and implemented using an object-based approach. Examining the commonality and differences in the objects included in the system has allowed us to take advantage of object-orientation to develop a sparse repository of well-defined components, representing terms specific to each domain and terms required for more general speech acts. Identifying commonality amongst terms relating to domains has led to a 70% reduction in the number of values required by the system (Table 1). By standardising the descriptors for the semantic input constructs, we have removed part of the complexity of NLG. Already, using easily changed descriptor values, the NLG module can output in different registers and languages and across different domains (Table 2). This flexibility of output was a primary motivation of our research.

A recently conducted evaluation of the system, involving non-specialist end users and developer users, has proved very encouraging. We hope to collate and publish our results in the near future. Evaluation of the system has focussed on real-world usability, and two aspects of this in particular. Firstly, in the case of non-specialist

end users, we have assessed the perceived appropriateness and quality of the words used in the system utterances. Secondly, in order to explore issues of maintainability and extensibility, we have tested the ability of novice developers to work with our utterance descriptors, including their ability to introduce new types of utterance, which, while exploiting some existing domain terms, involve the creation of new mini-templates.

Taking into account the results of evaluation and testing of the current MINTGEN system, future plans include a broadening of the system's vocabulary alongside an optimized range of mini-templates and Domain Terms objects for selected application domains.

Object-orientation is a well-established software design technique. However, the degree to which it may assist development of a cross-domain, multi-register, multi-lingual NLG system is, we believe, worthy of on-going investigation.

References

1. Poggi, I. et al.: Greta. A Believable Embodied Conversational Agent. In: Intelligent Information Presentation, Kluwer Academic (2003)
2. HUMAINE website, http://emotion-research.net/
3. Cole, R.A. et al.: Survey of the State of the Art in Human Language Technology, Chapter 4, Language Generation. CSLU, Oregon (1995)
4. Kasper, R.T.: A Flexible Interface for Linking Applications to Penman's Sentence Generator. In: Proceedings of the DARPA Speech and Natural Language Workshop, Philadelphia (1989)
5. O'Neill, I.M. et al.: The Queen's Communicator: An Object-Oriented Dialogue Manager. In: Proceedings of Eurospeech 2003, pp. 593-596, Geneva (2003)
6. De Carolis, B. et al.: APML, A Mark-Up Language for Believable Behaviour Generation. In: Proceedings of AAMAS Workshop, Bologna (2002)
7. Carnie A.: Flat Structure, Phrasal Variability and Non-Verbal Predication in Irish. In: Celtic Linguistics, Galway (2005)

Appendix: Tables of Sample Outputs from the System Components

Table 2(i). English *normal* generation for 'How many nights do you want to stay at the hotel?' This table corresponds to the DM request processing that was examined step-by-step in Section 4.5.

DM	`[info_request][gain_info][duration][-] [hotel][-][normal][•nglish]`
SP	`Eng[{Qnumber.register}{domain.durationUnit (plural)}{Qdesire.register}{domain.domain Verb(infinitive)}{domain.definite}]`
LP	`EngNorm[{QNumber}{hotel.durationUnit (plural)}{QDesire}{hotel.domainVerb (infinitive)}{hotel.definite}}]`

Table 2(i) continued

TOS	{how many} for {QNumber.english.normal} {do you want} for {QDesire.english.normal}
DTS	{night} for {hotel.english.durationUnit} {stay at} for {hotel.english.normal. domainVerb}
SR	{how many}{night (plural)}{do you want}{stay at (infinitive)}{hotel (definite)}
LIP	{nights} for {night (plural)}{to stay at} for {stay at (infinitive)}{the hotel} for {hotel (definite)}
SEP	–
NLG	how many nights do you want to stay at the hotel

Table 2(ii). English *formal* generation of the 'hotel' request – a *register* change from the request in the previous table.

DM	[info_request][gain_info][duration] [–][hotel][–]**[formal]**[english]
SP	Eng[{QNumber.register}{domain.durationUnit (plural)}{QDesire.register}{domain.domain Verb(infinitive)}{domain.definite}]
LP	EngForm[{QNumber}{hotel.durationUnit(plura l)}{QDesire}{hotel.domainVerb(infinitive)} {hotel.definite}}]
TOS	{for how many}for{QNumber.english.formal} {does one wish}for{QDesire.english. formal}
DTS	{night} for{hotel.english.durationUnit} {be accommodated at} for {hotel.english.formal.domainVerb}
SR	{for how many}{night (plural)}{does one wish}{be accommodated at (infinitive)}{hotel (definite)}
LIP	{nights} for {night (plural)}{to be accommodated at} for {be accommodated at (infinitive)}{the hotel} for {hotel (definite)}
SEP	–
NLG	for how many nights does one wish to be accommodated at the hotel

Table 2 (iii). *Irish* normal generation of the 'hotel' request – a *language* change.

DM	`[info_request][gain_info][duration][-][hotel][-][normal]` **`[irish]`**
SP	`Ir[{QNumber.register}{domain.durationUnit}{Qdesire.register}{domain.domainVerb(infinitive)}{domain.verbLinkWord}{domain.definite}]`
LP	`IrishNorm[{Qnumber}{hotel.durationUnit}{Qdesire}{hotel.domainVerb(infinitive)}{hotel.verbLinkWord}{hotel.definite}}]`
TOS	`{cé mhéad} for {Qnumber.irish.normal}{ar mhaith leat} for {Qdesire.irish.normal}`
DTS	`{oíche} for {hotel.irish.durationUnit}{stop} for {hotel.irish.normal.domainVerb}{i} for {hotel.irish.normal.verbLinkWord}`
SR	`{cé mhéad}{oíche}{ar mhaith leat}{stop(infinitive)}{i}{ostán(definite)}`
LIP	`{stopadh} for {stop (infinitive)}{an ostán} for { ostán (definite)}`
SEP	`{san} for {i}{an}`
NLG	`Cé mhéad oíche ar mhaith leat stopadh san ostán`

Table 2 (iv). English normal generation for a '*car*' request, which other than the domain change from hotel to car, is equivalent to the initial 'hotel' request.

DM	`[info_request][gain_info][duration][-]` **`[car]`**`[-][normal][English]`
SP	`Eng[{QNumber.register}{domain.durationUnit(pl)}{QDesire.register}{domain.domainVerb(infinitive)}{domain.definite}]`
LP	`EngNorm[{QNumber}{car.durationUnit(plural)}{QDesire}{car.domainVerb(infinitive)}{car.definite}}]`
TOS	`{how many} for {QNumber.english.normal}{do you want} for {QDesire.english.normal}`
DTS	`{day} for {car.english.durationUnit}{hire} for {car.english.normal.domainVerb}`
SR	`{how many}{day (plural)}{do you want}{hire(infinitive)}{car(definite)}`
LIP	`{days} for {day (plural)}{to hire} for {hire(infinitive)}{the car} for {car (definite)}`
SEP	–
NLG	`how many days do you want to hire the car`