

## Chapter 1

# FULL PARAPHRASE GENERATION FOR FRAGMENTS IN DIALOGUE

Christian Ebert

Shalom Lappin

Howard Gregory

*Department of Computer Science*

*King's College London*

{ebert, lappin}@dcs.kcl.ac.uk

howard.gregory@kcl.ac.uk

Nicolas Nicolov

*IBM T. J. Watson Research Center*

*Yorktown, New York*

nicolas@watson.ibm.com

## 1. Introduction

Much previous work on generation has focused on the general problem of producing lexical strings from abstract semantic representations. We consider generation in the context of a particular task, creating full sentential paraphrases of fragments in dialogue. When the syntactic, semantic and phonological information provided by a dialogue fragment resolution system is made accessible to a generation component, much of the indeterminacy of lexical selection is eliminated.

One major challenge for any dialogue interpretation system is the proper treatment of fragments. Examples include bare NP answers (2a), where the NP *a personnel manager* is resolved as the assertion (2b), and sluicing (4a), where the *wh*-phrase is interpreted as the question (4b)<sup>1</sup>.

- (1) Who hired Jones?
- (2) (a) A personnel manager.  
(b) *A personnel manager hired Jones.*
- (3) A personnel manager hired Jones.
- (4) (a) Who?  
(b) *Which personnel manager hired Jones?*

Furthermore the antecedent or the fragment may be embedded, as in the following examples.

- (5) The management asked who hired Jones.
- (6) (a) The personnel department thinks a personnel manager.  
(b) *The personnel department thinks a personnel manager hired Jones.*
- (7) The personnel department thinks a personnel manager hired Jones.
- (8) (a) The CEO wonders who.  
(b) *The CEO wonders which personnel manager hired Jones.*

Generating full paraphrases for interpreted fragments in a dialogue system is an important utility for facilitating human-machine communication. We consider three cases where paraphrase generation performs a useful function.

**Dialogue Systems.** In dialogue systems the purpose of paraphrase generation is twofold. First, it increases the explicitness and transparency of the system, which might be an option that the user wishes to use. Second, it is necessary for the generation of clarification questions, which can be used by the system to resolve ambiguity. This is illustrated in the following dialogue between a user and an information system.

USER: Who did the personnel department hire?  
 SYSTEM: John thinks a student.  
 USER: Who?

At this point *Who?* might ask for more details about *John* (because the user wonders who *John* is) or it might be a question about the hired student. To resolve this ambiguity the system produces the following clarification question by generating the paraphrase of the user's *wh*-sluice:

SYSTEM: *Do you want to know which student John thinks the personnel department hired?*

If the user answers positively the system responds with the fragment answer or the full paraphrase, depending on the parameter which controls the system's level of explicitness:

SYSTEM: Bill Smith.

or

SYSTEM: *John thinks the personnel department hired Bill Smith.*

If the user answers negatively, then the system will treat *Who?* as a clarificatory question on the utterance *John*.

**System Evaluation.** A natural application of paraphrase generation is the monitoring of a system's performance. Full paraphrases can be used to interactively test the system's interpretation of fragments. When the paraphrase is available to a human user, he/she can confirm or revise the paraphrase, and so monitor the performance of the system efficiently. The paraphrase generator that we present in this paper is currently used for monitoring the performance of the SHARDS system – a system for resolving fragments in dialogue – which is introduced in more detail in section 2.

**Machine Translation.** Certain elided structures pose a problem for machine translation. Although the source language might exhibit ellipsis structures of a specific kind, the target language might not allow for these. Therefore these structures will have no direct translation. Two cases in point are VP-ellipsis and pseudo-gapping in English, which have no direct counterparts in languages such as German. A straightforward solution is to use full paraphrases instead of ellipsis as the input to the MT component. Consider the following English dialogue and its translation into German<sup>2</sup>.

- (9) (E) Who submitted a report today?  
 (G) Wer legte heute einen Bericht vor?
- (10) (E) John did to his supervisor.  
 (G) John ?[\*tat es] seinem Betreuer.
- (11) (E) John submitted a report to his supervisor today.  
 (G) *John legte heute einen Bericht seinem Betreuer vor.*

The English answer (10E) exhibits pseudo-gapping, which cannot be translated into a similar structure in German (10G). A translation including an auxiliary corresponding to *did* is ungrammatical. Dropping the auxiliary prior to the translation results in an (at best) odd sentence.

This problem can be circumvented using paraphrase generation. Instead of translating (10E) directly, its paraphrase (11E) is computed. Then the translation can proceed with this paraphrase as the source input to obtain an appropriate German sentence (11G).

Using the fragment interpretation system SHARDS we show how to generate paraphrases for fragments in dialogues like those in (b) in the examples

(1)–(8) above. The generator uses a template-filler approach, and it does not do any deep generation from an underlying semantic representation. Instead it reuses the results of the parse and interpretation process of SHARDS to dynamically compute the templates, and then to update the filler. This recycling of already available syntactic, semantic, and phonological information makes generation efficient because it reduces the operations of the generator to string manipulations.

In Section 2 we give a review of the SHARDS system and the grammatical background. We then explain our proposal for generating fragment paraphrases with templates in Section 3. In Section 4 we describe the implementation of SHARDS and the generation component, which is illustrated with some examples. Section 5 sketches some directions for future work.

## 2. SHARDS

SHARDS (Ginzburg et al., 2001) is a Head Driven Phrase Structure Grammar (HPSG)-based system for the resolution of fragments in dialogue. It is based on a version of HPSG developed in (Ginzburg and Sag, 2000) which integrates the situation semantics-based theory of dialogue context given in the KOS framework (Cooper et al., 1999) into recent work in HPSG (Pollard and Sag, 1994; Sag, 1997).

### 2.1 Fragment Resolution

Following (Ginzburg and Sag, 2000), two new attributes are defined within the CONTEXT feature structure: the *Maximal Question Under Discussion* (MAX-QUD) and the *Salient Utterance* (SAL-UTT).

The MAX-QUD<sup>3</sup> can be seen as the most salient question that needs to be answered in the course of a dialogue. Its value is of type *question*. In the framework of this system, questions are represented as semantic objects comprising a set of parameters (PARAMS) – that is, restricted indices – and a State of Affairs (SOA).

$$\left[ \begin{array}{l} \textit{question} \\ \text{PARAMS} \quad \{\pi, \dots\} \\ \text{SOA} \quad \left[ \textit{soa}(\dots \pi \dots) \right] \end{array} \right]$$

This is the feature structure counterpart of the  $\lambda$ -abstract  $\lambda\pi.(\dots \pi \dots)$ . In a *wh*-question the PARAMS set represents the abstracted INDEX values associated with the *wh*-phrase(s). For a polar (yes-no) question the PARAMS set is empty. In general a number of such questions may be available in a given dialogue context, of which one is selected as the value of MAX-QUD.

For instance, by uttering (1) or (5) the speaker expects the question *Who hired Jones?* to be answered in the following dialogue. Analogously, by uttering (3) or (7) the speaker expects the dialogue partner to elaborate on the corresponding polar question *Did a personnel manager hire Jones?* (with *a personnel manager* being the salient utterance).

The SAL-UTT represents a distinguished constituent of the utterance whose content is the current value of MAX-QUD. In information structure terms, it can be thought of as specifying a potential parallel element correlated with an element in the antecedent question or assertion. Its value is of type *sign*, enabling the system to encode syntactic categorial parallelism, including case assignment for the fragment. Specifically, SAL-UTT is computed as the constituent associated with the role bearing widest scope within MAX-QUD:

- For *wh*-questions, SAL-UTT is the *wh*-phrase associated with the PARAMS set of the question.
- If MAX-QUD is a question with an empty PARAMS set, the context will be underspecified for SAL-UTT. The possible values for the SAL-UTT feature are either the empty set or the constituent associated with the widest scoping quantifier in MAX-QUD. This is invoked to resolve sluicing<sup>4</sup>. In the case of polar questions, SAL-UTT will be empty.

According to these requirements the SAL-UTT in the before-mentioned MAX-QUD of (1) and (5) is the *wh*-phrase *who*. In case of (3) and (7) the SAL-UTT is the constituent *a personnel manager*, which constitutes the widest scoping quantifier.

Interpreting a fragment in dialogue consists in computing from context (represented as a dialogue record) the MAX-QUD and SAL-UTT features of the assertion or question clause that the fragment expresses, and then using these features to specify the CONTENT feature of the clause. In section 4 we will comment further on this computation.

## 2.2 Grammatical Framework

Bare argument phrases constitute a non-head daughter (the fragment), the remaining information for the interpretation being provided primarily by the contextual features. Following (Sag, 1997; Ginzburg and Sag, 2000), phrases are classified not only in terms of their phrasal type (*headedness*), but also with respect to the further dimension of *clausality*<sup>5</sup>. Figure 1.1 shows a multi-dimensional type hierarchy, where clauses are divided into (among others) declarative clauses (*decl-cl*), which denote SOAs, and interrogative clauses (*int-cl*) denoting questions. Each maximal phrasal type inherits from both the *headedness* and the *clausality* dimension. This classification allows us to specify systematic correlations between clausal construction types and types

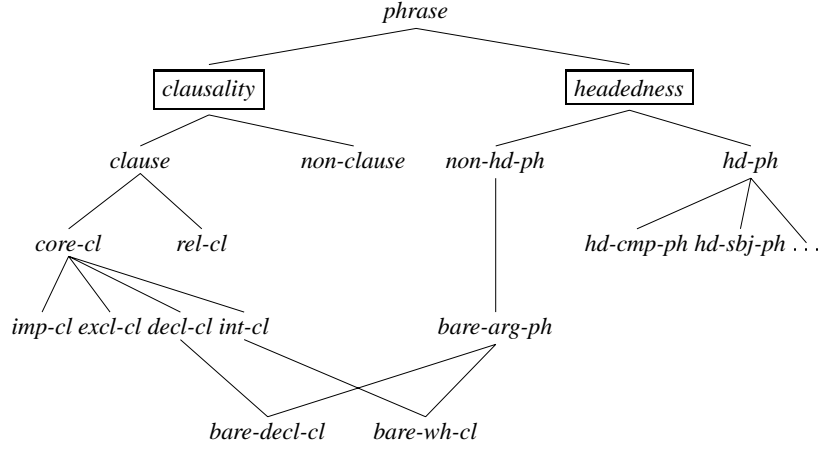


Figure 1.1. Multi-dimensional type hierarchy

of semantic content. Fragments are assigned to a subtype of the type *bare-arg(ument)-ph(rase)*. Bare argument phrases are specified to constitute a non-head daughter (the fragment), the remaining information for the interpretation being provided primarily by the contextual features. The clausal content, notably the nucleus of the SOA, which in a headed clause would be inherited from the head daughter, is here constructed from the MAX-QUD. The constituent in the antecedent picked up by SAL-UTT links the bare phrase to the appropriate argument-role, and enforces categorial identity. The type constraint associated with the supertype *bare-arg-ph* is as follows.

*bare-arg-ph*  $\Rightarrow$

$$\left[ \begin{array}{l} \text{STORE} \quad \{ \} \\ \text{CONT} \mid \text{SOA} \mid \text{NUCL} \quad \boxed{1} \\ \text{CTXT} \quad \left[ \begin{array}{l} \text{MAX-QUD} \mid \text{SOA} \mid \text{NUCL} \quad \boxed{1} \\ \text{SAL-UTT} \quad \left[ \begin{array}{l} \text{CAT} \quad \boxed{2} \\ \text{CONT} \mid \text{INDEX} \quad \boxed{3} \end{array} \right] \end{array} \right] \\ \text{NHD-DTRS} \quad \left\langle \left[ \begin{array}{l} \text{CAT} \quad \boxed{2} \\ \text{CONT} \mid \text{INDEX} \quad \boxed{3} \end{array} \right] \right\rangle \end{array} \right]$$

It requires unification of the NUCLEUS (propositional core) value of the MAX-QUD with the NUCLEUS value of the CONTENT feature of the proposition or question of which the bare argument is the non-head daughter. Similarly, the

CAT feature of the *SAL-UTT* and the non-head daughter features of the bare phrase are unified to insure syntactic categorial parallelism.

We posit two subtypes of *bare-arg-ph*: *bare-decl-cl* for ‘short answers’ and *bare-wh-cl* for sluices. These subtypes are also subtypes of the *clausality* types *decl-cl* and *int-cl*, respectively. By type inheritance it follows that *bare-decl-cl* denotes a SOA and that the information in the constraint for *bare-arg-ph* holds. The only information, beyond that inherited from *bare-arg-ph* and *decl-cl*, which remains to be specified concerns the scoping of quantifiers and the amalgamation of semantic restrictions.

- If the bare phrase is (or contains) a quantifier  $Q$ , then  $Q$  gets scoped in wider than the (optionally existing, already scoped in) quantifiers of the contextually salient question (MAX-QUD).
- The set of (semantic) restrictions on the index  $I$  of the bare clause is the union of the restrictions on  $I$  in MAX-QUD and those contributed by the bare phrase.

The constraint particular to *bare-decl-cl* is, hence, the following:

*bare-decl-cl*  $\Rightarrow$

$$\left[ \begin{array}{l} \text{CONT} \mid \text{SOA} \quad \left[ \begin{array}{l} \text{QUANTS} \quad \boxed{1} \oplus \boxed{2} \\ \text{RESTR} \quad \boxed{3} \cup \boxed{4} \end{array} \right] \\ \\ \text{CTXT} \quad \left[ \begin{array}{l} \text{MAX-QUD} \quad \left[ \begin{array}{l} \text{PARAMS} \quad \left\{ \left[ \text{INDEX} \quad \boxed{5} \right] \right\} \\ \text{SOA} \quad \left[ \begin{array}{l} \text{QUANTS} \quad \boxed{2} \\ \text{RESTR} \quad \boxed{3} \end{array} \right] \end{array} \right] \end{array} \right] \\ \\ \text{NHD-DTRS} \quad \left\langle \left[ \begin{array}{l} \text{CONT} \quad \left[ \begin{array}{l} \text{INDEX} \quad \boxed{5} \\ \text{RESTR} \quad \boxed{4} \end{array} \right] \\ \text{STORE} \quad \boxed{1} \text{ set}(qf) \end{array} \right] \right\rangle \end{array} \right]$$

As with *bare-decl-cl*, the type *bare-wh-cl* inherits a significant part of its specification through being a subtype of *bare-arg-ph* and *int-cl*. The conditions that are specific to *bare-wh-cl* pertain to quantifiers and restrictions.

- The widest scoping quantifier  $Q$  in MAX-QUD’s QUANTS list is removed from the QUANTS list of the content of a *bare-wh-cl*. Thus, the widest scoping quantifier, if any, in the open proposition of the question after resolution will be whichever quantifier, if any, was previously scoped just narrower than  $Q$ .

- The set of (semantic) restrictions on the index  $I$  of the bare  $wh$ -phrase is the union of the restrictions on  $I$  in MAX-QUD with the restrictions on  $I$  contributed by the bare phrase.

The constraint particular to *bare-wh-cl* is, hence, the following:

*bare-wh-cl*  $\Rightarrow$

$$\left[ \begin{array}{l} \text{CONT} \\ \text{CTXT} \\ \text{NHD-DTRS} \end{array} \left[ \begin{array}{l} \text{PARAMS} \left\{ \begin{array}{l} \text{INDEX} \quad \boxed{1} \\ \text{RESTR} \quad \boxed{2} \cup \boxed{3} \end{array} \right\} \\ \text{SOA} \mid \text{QUANTS} \quad \boxed{4} \\ \\ \text{MAX-QUD} \left[ \begin{array}{l} \text{PARAMS} \quad \{\} \\ \text{SOA} \mid \text{QUANTS} \left\langle \begin{array}{l} \boxed{5} \left[ \begin{array}{l} \text{non-neg-}qf\text{-rel} \\ \text{INDEX} \quad \boxed{1} \\ \text{RESTR} \quad \boxed{3} \end{array} \right] \right\rangle \\ \oplus \boxed{4} \text{ list}(qf) \end{array} \right. \\ \text{SAL-UTT} \mid \text{STORE} \quad \boxed{5} \end{array} \right] \\ \\ \text{NHD-DTRS} \left\langle \left[ \text{STORE} \left[ \begin{array}{l} \text{INDEX} \quad \boxed{1} \\ \text{RESTR} \quad \boxed{2} \end{array} \right] \right] \right\rangle \end{array} \right. \right]$$

We give the result of resolving fragment (2a). After parsing the antecedent (1) and the fragment phrase, the resolution procedure yields the Attribute-Value-Matrix (AVM) in figure 1.2. The AVM satisfies the constraints mentioned above, as the NUCLEUS value of the entire clause for which the bare phrase is a non-head daughter is structure shared with MAX-QUD's NUCLEUS via  $\boxed{1}$ , categorial parallelism is ensured via  $\boxed{6}$ , and the semantic restrictions  $\boxed{2}$  and  $\boxed{3}$  have been amalgamated into the RESTR value of the phrase. The generator uses AVMs like this to set up its templates and fillers.

### 3. Generation of Fragment Paraphrases

Template-based approaches to NL generation have proved useful in various systems (see e.g. (Reiter, 1995; Becker and Busemann, 1999)). These approaches are particularly appropriate in systems where large parts of the text to be generated remain fixed in some way, or are partially determined prior to generation. An inspection of the following dialogue indicates that this is true for the generation of fragment paraphrases:

(12) The personnel department wonders who the CEO hired.

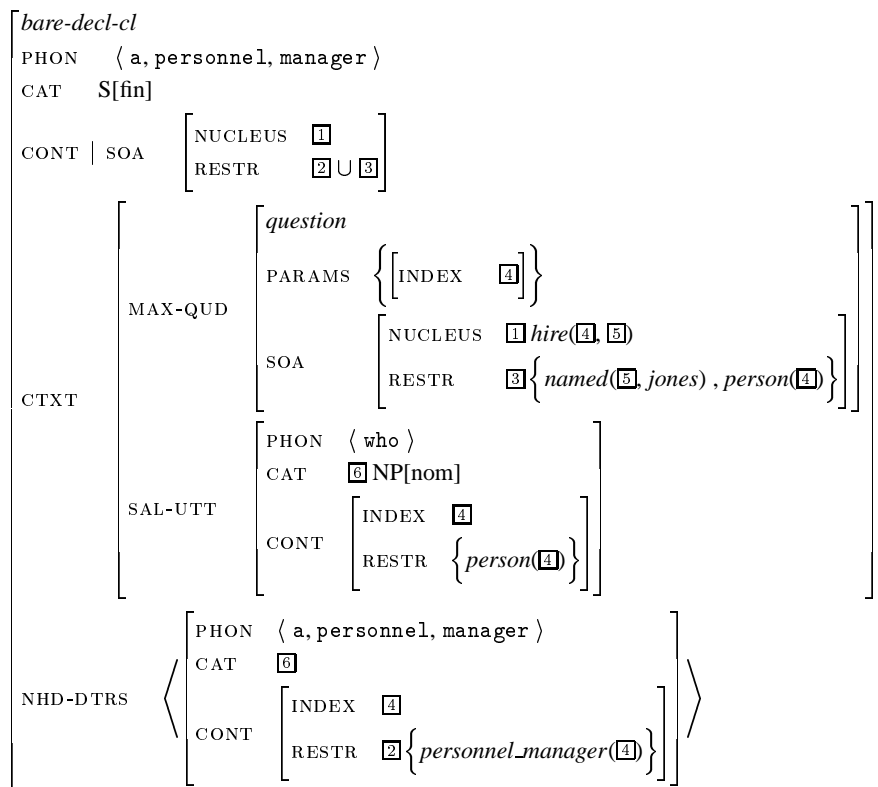


Figure 1.2. Resolution of (2a): A personnel manager hired Jones.

- (13) (a) John thinks a student.  
 (b) *John thinks the CEO hired a student.*
- (14) (a) Who?  
 (b) *Which student did the CEO hire?*
- (15) (a) The student who passed all tests.  
 (b) *The CEO hired the student who passed all tests.*

The paraphrase (13b) is a combination of parts of the fragment answer (13a) and the antecedent clause (12). It is worth noting at this point that dialogues with consecutive fragment answers or questions (such as the one above) are recursive in nature. The paraphrase (15b) for instance contains material from the initial clause of the discourse. Just considering the antecedent *Who?* and the fragment *the student who passed all tests* itself is not sufficient to generate the fragment's paraphrase. It is necessary to insure that some parts of the dialogue (in this case *the CEO hired ...*) are carried over from one paraphrase to the other, while other parts have to be modified with respect to the fragment (such as *which student* and *the student who passed all tests*). This suggests the use of templates for the fixed parts that have to be carried over, and of fillers for the parts that might require modification within the templates.

Unlike template-based generation systems where the templates are defined in advance, our algorithm uses dynamically created templates. The templates are not fixed in advance, but must be constructed dynamically from the dialogue. In a dialogue such as (12)–(15) the templates are built once for a specified MAX-QUD at the very beginning and remain specified until the dialogue ends. More precisely, every change in the MAX-QUD forces a recomputation of the templates. Another dynamic feature concerns the filler. After each utterance, the filler is updated and combined with one of the templates to yield the intended paraphrase. This enables us to update the paraphrases in a sequence of questions and fragment answers.

The main point in using this template-filler mechanism is that it permits the reuse of constituents/syntactic structure that are already available. Since the surface strings are accessible from the user input, and the syntactic and semantic information is available from the parse and fragment interpretation, it is useful to exploit these resources. Our generator performs *shallow generation* by recycling already given data without using deep semantic knowledge.

We now consider the dynamic construction of the templates and fillers in greater detail. Essentially, the templates correspond to different paraphrase schema of the MAX-QUD, while the fillers correspond to the fragments.

### 3.1 Identifying the Templates

In (12)–(15) the SHARDS system identifies the question *Who did the CEO hire?* as the maximal question under discussion and *who* as the salient utterance.

We can see that a certain constituent of the MAX-QUD – namely *did the CEO hire* – reappears in modified forms in the paraphrases (13)–(15)(b): The paraphrases (13b) and (15b) contain this component in Subject-Verb-Object (SVO) order (*the CEO hired ...*). The paraphrase (14b) however contains this component in Subject-Auxiliary-Inversion (SAI) form (*... did the CEO hire*). To handle these phenomena our algorithm constructs two different templates: the template  $T_{svo}$  for the SVO case and  $T_{sai}$  for the SAI case.

Both templates contain a slot (represented by  $\diamond$ ) which determines the position that the filler will occupy. In the example above we indicated this position with three points. For the dialogue in (12)–(15) the templates are

$$\begin{aligned} T_{svo} &= \text{the CEO hired } \diamond \\ T_{sai} &= \diamond \text{ did the CEO hire} \end{aligned}$$

### 3.2 Identifying the Filler

The dialogue contains the following progression from the SAL-UTT *who* in (12) to the final bare NP answer *the student who passed all tests* in (15):

$$\begin{array}{c} F_1 : \textit{who} \\ \downarrow \\ F_2 : \textit{a student} \\ \downarrow \\ F_3 : \textit{which student} \\ \downarrow \\ F_4 : \textit{the student who passed all tests} \end{array}$$

We generate the paraphrases of the corresponding fragments by substituting the fillers  $F_1$  to  $F_4$  for the slots in the corresponding templates. For instance substituting  $F_3$  for the slot in  $T_{sai}$  (which we write  $T_{sai}(F_3)$ ) yields the intended paraphrase (14b).

Each fragment produces an update of the preceding filler. E.g. the filler  $F_2$  is updated by the fragment *who* of (14a) to  $F_3$ . The final selection of the template and its composition with the filler depends on the entire fragment clause and not on the fragment alone. In (13), for instance, we have to prefix *John thinks* to  $T_{svo}(F_2)$  to get the intended result.

## 4. An Implemented System for Fragment Resolution and Paraphrase Generation

One of the system's main tasks is the computation of the MAX-QUD. This is accomplished by computing a list of possible MAX-QUD candidates from the parsed utterances. After the system has identified a fragment, it is matched with the MAX-QUD candidates. From the matching candidates one is selected and the final fragment resolution can take place according to the constraints given

in section 2. This resembles algorithms for anaphora resolution, where a set of possible antecedents is computed and each of them is matched (i.e. checked for agreement) with the anaphor under consideration. As in the case of anaphora resolution, where ambiguity arises due to several possible antecedents, there may be several MAX-QUD candidates which can be matched with a fragment. Add present SHARDS selects the most recent MAX-QUD candidate of appropriate semantic type for the fragment. However, as the example in (16) indicates this strategy will not always succeed.

- (16) (a) Why did Mary arrive early?  
 (b) I can't tell you!  
 (c) Why can't you tell me?  
 (d) Ok, if you must know – to surprise you.

The utterance of (16c) adds the new MAX-QUD candidate corresponding to *Why can't you tell me?* to the candidate list. However, the next utterance (16d) contains a fragment answer which has to be resolved with respect to the MAX-QUD introduced by (16a). We are in the process of developing a more refined procedure for ranking MAX-QUD candidates and selecting the highest valued of the candidate list.

The entire system of fragment resolution and generation is implemented in Prolog using ProFIT (Erbach, 1996) as a formalism to handle typed feature structures. The complete system consists of five main components. The components I-IV constitute the SHARDS system (Ginzburg et al., 2001). V is the paraphrase generator.

**I. HPSG Grammar.** This is a substantially modified version of the grammar employed by (Gregory and Lappin, 1999), but using the types and features mentioned in Section 2.

**II. Dialogue Record.** When a clause has been parsed its AVM is converted into a transitive network of Mother-Daughter-Relations (the *MDR list*) and then stored in a dialogue record. If the clause contains a fragment, component III (CONTEXT Resolution Procedure) is invoked to resolve the fragment. Otherwise a list of MAX-QUD candidates is computed as follows: At first, all subclauses of the clause are identified. Then all CONTENT features of each of these subclauses are extracted and added to the MAX-QUD candidate list as a further component of the dialogue record. By adding new candidates to the list we are able to keep track of MAX-QUDs from preceding utterances.

**III. CONTEXT Resolution Procedure.** This assigns values from the dialogue record to the MAX-QUD and SAL-UTT features of the current clause as follows: For each element in the MAX-QUD candidate list the SAL-UTT is

computed from the sign whose content provides MAX-QUD by applying the conditions given in section 2. The most recent MAX-QUD element, which yields a SAL-UTT that is compatible with the fragment is selected as the value of the MAX-QUD feature.

The following short dialogue shows how the check for compatibility of SAL-UTT and fragment enables us to select the appropriate MAX-QUD.

- (17) A board member asks who hired Jones.  
 (18) (a) Peter wonders who.  
       (b) *Peter wonders which board member asks who hired Jones.*  
 (19) (a) Peter thinks the CEO.  
       (b) *Peter thinks the CEO hired Jones.*

In component II two MAX-QUD candidates are computed, corresponding to *Does a board member ask who hired Jones?* and *Who hired Jones?*. In component III the two SAL-UTTs *a board member* and *who* are computed from the MAX-QUD candidates, respectively. As the first one (being a proper noun phrase) can only be matched with the fragment *who* in (18a) and the second one (being a *wh*-phrase) can only be matched with the fragment *the CEO* in (19a), the appropriate MAX-QUD with respect to the fragment is selected. Thus the final paraphrases will be as shown in (18b) and (19b).

**IV. Fragment Resolution Procedure.** This computes the semantics (i.e. the CONTENT) of the current fragment clause from its MAX-QUD and SAL-UTT values according to the type constraints specified in section 2.

**V. Paraphrase Generator.** After the resolution procedure has been applied, the generator is called with the antecedent clause in the form of the MDR list and the resolved fragment clause in the form of an AVM. It computes the templates, updates the fillers, and returns the paraphrase of the fragment clause.

#### 4.1 Implementation of the Generation Algorithm

The generator takes AVMs as arguments (a list of AVMs in from of the MDR list corresponding to the antecedent and the AVM of the resolved fragment clause) and returns the paraphrase of the fragment clause in the form of a word string. The MDR list argument enables the generator to search the parse of the antecedent clause efficiently, e.g. for the auxiliary in SAI clauses or for some verbal head, etc. The operations (such as deletion or substitution of an element) that the generator performs are carried out on the surface strings, i.e. on the values of the PHON features. The generator extracts the PHON values of

the AVMs for the antecedent clause and the fragment clause in the initial phase of generation and performs its operations not on their AVMs but just on these surface strings. All operations – though guided by the information in the full AVMs – are simple string manipulations.

If the generator is called for the first time (or when the MAX-QUD changes, e.g. when a new dialogue starts), it computes the two templates and the first filler  $F_1$ , using the antecedent clause and the fragment, according to the following algorithm:

**Initialize.**

- 1 Identify the SAL-UTT and store the AVM as first filler  $F_1$ .
- 2 Identify the MAX-QUD and delete the SAL-UTT to get the AVM  $MQ$ .
- 3 Compute the templates  $T_{sai}$  and  $T_{svo}$  from  $MQ$  and store them:
  - (a) If  $MQ$  is in SVO order, store it as  $T_{svo}$ , compute  $T_{sai}$ , and insert the slots; else
  - (b)  $MQ$  is in SAI order, store it as  $T_{sai}$ , compute  $T_{svo}$ , and insert the slots.

The setup of the templates requires a conversion from a clause in SVO order to one in SAI order or vice versa. If the clause is in SVO order, the verbal head has to be searched, an appropriate auxiliary has to be inserted into the string, and the base form of the verb has to be substituted for its inflected form. If the conversion has to be done in the other direction, then the auxiliary is identified and deleted, and the inflected form of the verb is substituted for its base form.

The insertion of the slot at the appropriate position in the SVO template requires the identification of the verbal head which subcategorizes for the SAL-UTT. After this verb has been found, the slot is inserted at the position that the SAL-UTT would occupy. E.g. if the SAL-UTT is the subject, then the slot is inserted immediately in front of this verb. In the case of the SAI template the slot is inserted at the beginning of the template just in front of the auxiliary. As mentioned above, the search for daughters such as the verbal heads can be done efficiently by going through the MDR list, which has already been computed for the SHARDS dialogue record (cf. component II above).

Once the templates are set up, the generator updates the filler  $F_i$  to  $F_{i+1}$  according to the type of the fragment. Below we refer to the clause which contains the fragment as  $FC$ .

**Update Filler.**

- 1 If the fragment of  $FC$  is a *wh*-phrase, substitute *which* for the determiner in the filler  $F_i$ ; else,

- 2 substitute the fragment for  $F_i$ .

To accomplish the first case of filler update the AVM corresponding to the stored filler  $F_i$  is searched for its determiner. Then *which* is substituted for this determiner. The second case is straightforward.

After the filler has been updated the generator is ready to compute the complete paraphrase  $P_{i+1}$ . Let  $\circ$  stand for the concatenation of strings and  $\varepsilon$  for the empty string. We write  $T(F)$  for the result of substituting  $F$  for the slot  $\diamond$  in the template  $T$ . Thus  $T(\varepsilon)$  is the result of deleting the slot from  $T$ . The composition of template and filler depends on the clause  $FC$ , which contains the fragment.

#### Composition of template and filler.

- 1 If  $F_{i+1}$  is a *wh*-phrase
  - (a) If  $FC$  consists just of the fragment, then  $P_{i+1} = T_{sai}(F_{i+1})$ ; else,
  - (b) substitute  $F_{i+1} \circ T_{svo}(\varepsilon)$  for the fragment in  $FC$  to get  $P_{i+1}$ ;
 else,
- 2 substitute  $T_{svo}(F_{i+1})$  for the fragment in  $FC$  to get  $P_{i+1}$ .

The if-then cascade takes care of the possible word orders and specifically of the fronting of *wh*-phrases. 1(a) handles fronted *wh*-phrases in SAI constructions of *wh*-questions such as *Which student did the CEO hire?*. Case 1(b) handles cases where the *wh*-question is embedded, as in *The personnel department wonders which student the CEO hired*. Case 2 generates the SVO order in embedded and non-embedded cases (*[John thinks] the CEO hired a student*).

Once the filler and the template have been set up, the composition of these two components is achieved by simple string concatenation. We insure agreement of the filler and the template by supplying the slot with the corresponding agreement features of the template and checking them when the filler is inserted<sup>6</sup>.

## 4.2 The Generator at work

**Basic Examples.** We will now illustrate the generation algorithm with three basic examples, which correspond to the three cases of the composition procedure. The following dialogue yields an instance of a non-embedded *wh*-sluice and thus corresponds to case 1(a) of the composition procedure.

- (20) Mary likes a student.
- (21) Who?

The first time the generator is called the arguments are a parse (in the form of the MDR list) of the antecedent clause (20) and an AVM of the parsed

and interpreted fragment clause (21). As this is the first call, the templates  $T_{svo}$  and  $T_{sai}$  are generated in the initialization procedure in the way described above. SHARDS identifies *Mary likes a student* as the maximal question under discussion and *a student* as the salient utterance. Thus the first filler  $F_1$  is set to *a student* and the templates are computed from *Mary likes*, which is the maximal question under discussion after deletion of the salient utterance. After initialization the situation is as follows:

$$\begin{aligned} F_1 &= \text{a student} \\ T_{svo} &= \text{Mary likes } \diamond \\ T_{sai} &= \diamond \text{ does Mary like} \end{aligned}$$

Immediately after this initial computation the filler is updated to

$$F_2 = \text{which student}$$

in case 1 of the update procedure, because the fragment is the *wh*-phrase *who*. As the filler is a *wh*-phrase and the fragment clause consists just of the fragment, case 1(a) of the composition procedure is invoked and the final paraphrase is computed as

$$P_2 = \text{Which student does Mary like?}$$

In the following dialogue the *wh*-sluice is embedded.

- (22) Mary likes a student.  
 (23) John wonders who.

As the MAX-QUD, the SAL-UTT, and the fragment are the same as in the previous example, the results after the initialization and update of the filler are the same. However, as the fragment is embedded, case 1(b) of the composition is invoked.  $F_2 \circ T_{svo}(\varepsilon) = \text{which student Mary likes}$  is substituted for the fragment *who* in the fragment clause *John wonders who*. The final result is

$$P_2 = \text{John wonders which student Mary likes.}$$

The following dialogue illustrates the application of case 2 of the composition procedure. As this case works for non-embedded as well as for embedded fragments, we indicate *John thinks* as being optional.

- (24) Who does Mary like?  
 (25) [John thinks] a student.

In (24) *does Mary like* is identified as the MAX-QUD and *who* as the SAL-UTT. Therefore the initialization procedure yields the following results.

$$\begin{aligned} F_1 &= \text{who} \\ T_{svo} &= \text{Mary likes } \diamond \\ T_{sai} &= \diamond \text{ does Mary like} \end{aligned}$$

The update of the filler yields

$$F_2 = \text{a student}$$

according to case 2 of the update procedure. Finally, this filler and the SVO template are composed to

$$P_2 = [\text{John thinks}] \text{ Mary likes a student}$$

in case 2 of the composition procedure, because the filler is not a *wh*-phrase.

**Consecutive Paraphrases and Multiple Embeddings.** The algorithm is capable of generating paraphrases of consecutive fragments as shown in dialogue (12)–(15). After initialization the situation is as follows:

$$\begin{aligned} F_1 &= \text{who} \\ T_{svo} &= \text{the CEO hired } \diamond \\ T_{sai} &= \diamond \text{ did the CEO hire} \end{aligned}$$

The update of the filler yields

$$F_2 = \text{a student.}$$

As the filler is not a *wh*-phrase, case 2 of the composition procedure applies and  $T_{svo}(F_2) = \text{the CEO hired a student}$  is substituted for *a student* in (13). The final paraphrase is

$$P_2 = \text{John thinks the CEO hired a student}$$

The next call of the generator with (13) as the antecedent clause and (14) as the fragment clause leads to an update of  $F_2$  to

$$F_3 = \text{which student}$$

as the fragment is *who* and *which* is substituted for the determiner *a* in  $F_2$ . As  $F_3$  is a *wh*-phrase and the fragment clause consists just of the fragment *who*, case 1(a) of the composition procedure is considered.

$$P_3 = \text{Which student did the CEO hire?}$$

After the last call of the generator, the filler  $F_3$  is updated to

$$F_4 = \text{the student who passed all tests}$$

and the final paraphrase is computed as

$$P_4 = \text{The CEO hired the student who passed all tests}$$

This algorithm works properly with examples that contain multiple embeddings, as in the case of the following antecedent clause, where the *wh*-phrase occurs outside of the matrix clause *the system administrator thinks...*

- (26) Peter wonders who the system administrator thinks deleted the files.
- (27) (a) The webmaster believes a student.  
 (b) *The webmaster believes the system administrator thinks a student deleted the files.*
- (28) (a) Who?  
 (b) *Which student does the system administrator think deleted the files?*

SHARDS identifies the question *Who does the system administrator think deleted the files* as the maximal question under discussion and *who* as the salient utterance (and the first filler  $F_1$ ). Therefore the antecedent clause (26) gives rise to the two templates

$$T_{svo} = \begin{array}{l} \text{the system administrator} \\ \text{thinks } \diamond \text{ deleted the files} \end{array}$$

$$T_{sai} = \begin{array}{l} \diamond \text{ does the system administra-} \\ \text{tor think deleted the files} \end{array}$$

The first update of the filler yields  $F_2 = \text{a student}$ , and the paraphrase (27b) is generated on case 2 of the composition procedure. In the next turn  $F_2$  is updated to  $F_3 = \text{which student}$ , and case 1(a) of the composition procedure leads to (28b).

**Polar Questions.** The algorithm is able to produce paraphrases of answers to polar questions quite straightforwardly.

- (29) Does Peter think the CEO hired a student?
- (30) (a) Yes.  
 (b) *Peter thinks the CEO hired a student.*  
 (c) No.  
 (d) *Peter does not think the CEO hired a student.*

As the polar question (29) is an SAI construction we can just run the procedure for setting up the templates on it. This will compute  $T_{svo}$  and since the SAL-UTT is empty in the case of polar questions, nothing will be deleted, and no slot will be inserted. The result will be the paraphrase (30b). A small addition to this procedure makes it possible to generate the paraphrases of negative answers such as (30c) as well.

The generator can produce paraphrases for all phenomena that the parser and the resolution procedure of SHARDS currently handle. These are bare NP fragments and bare *wh*-sluices, embedded fragments and polar questions of the kind illustrated in the preceding sections.

## 5. Conclusion and Future Research

Most work on NL generation such as (Shieber et al., 1990; Kay, 1996; McKeown, 1985; Nicolov and Mellish, 2000) has formulated the problem in abstract terms as the production of a lexical string to encode a semantic representation. We have situated generation within the context of dialogue interpretation, specifically fragment resolution. In doing so, we have been able to eliminate much of the indeterminacy which characterizes classical generation systems by exploiting the rich syntactic and phonological information produced in the course of dialogue interpretation.

While there are undoubtedly generation problems to which this approach does not apply, the work described here does suggest the possibility of efficient generation through the exploitation of the results of dialogue interpretation in an important class of NL applications.

The research on the system we introduced in the preceding sections is part of a larger project of dialogue management at King's College, London. We are extending our system in several directions. We plan to incorporate the handling of ellipsis structures (VP-ellipsis and gapping) into the SHARDS system, and the generation component will be developed to deal with these constructions. In addition, we are currently doing corpus work with the British National Corpus, which will eventually result in a typology of fragment types. We will use the examples of ellipsis that we find in this corpus to evaluate and improve our generation system. We are also in the process of extending the lexicon to achieve broader coverage for our parser and fragment interpretation components (Purver, 2001).

## Acknowledgments

We are grateful to an anonymous reviewer and the participants of the 2nd SIGDial Workshop in Aalborg for helpful comments and suggestions. The dialogue project of which the work described here is a part is funded by grant number R00022269 of the Economic and Social Research Council of the United Kingdom. Some of the research presented in this paper was done in the summer of 2000, when the second author was a Visiting Academic at the IBM T.J. Watson Research Center in Hawthorne, NY. During this time he worked with the fourth author on the design and implementation of an initial version of the generation algorithm.

## Notes

1. In cases like this one, *Who?* may as well be a clarificatory question on *Jones*, meaning *Who is Jones?*. See (Ginzburg, 2001) for an account of clarificatory ellipsis within the general HPSG framework assumed here. A procedure for recognizing clarificatory question fragments is being developed on the basis of this account.
2. Currently our system is not capable of handling ellipsis structures. But as we point out in the last section, we are extending it to deal with these cases.
3. In the following we will use 'MAX-QUD' as an abbreviation for 'maximal question under discussion' as well as for the corresponding HPSG feature
4. SAL-UTT can also be a set containing more than one element in contexts where MAX-QUD is a multiple question as in A: *Who arrived when?* B: *Jo at 5, Mustafa at 7.*
5. Ginzburg and Sag (2000) actually treat bare fragments as head daughters. For simplicity of exposition we retain the typed feature system which we employ in our current implementation of SHARDS and the paraphrase generator. It would not be difficult to revise this system in accordance with the analysis of bare phrases presented in (Ginzburg and Sag, 2000).
6. It will be necessary to refine our agreement checking procedure to deal with mismatches in number and tense, as in (i) and (ii) respectively.
  - (i) (a) Who is presenting the report?  
(b) John and Mary.
  - (ii) (a) Who has written the program?  
(b) Mary will.

## References

- Becker, T. and Busemann, S., editors (1999). *May I Speak Freely? Between Templates and Free Choice in Natural Language Generation. Workshop at the 23rd German Annual Conference for Artificial Intelligence (KI '99)*, Saarbrücken. DFKI.
- Cooper, R., Larsson, S., Poesio, M., Traum, D., and Matheson, C. (1999). Coding instructional dialogue for information states. In *Task Oriented Instructional Dialogue (TRINDI): Deliverable 1.1*. University of Gothenburg, Gothenburg.
- Erbach, G. (1996). ProFIT: Prolog with features, inheritance and templates. In *Proceedings of the 7th European Conference of the Association for Computational Linguistics*, pages 180–187.
- Ginzburg, J. (2001). Clarification ellipsis and nominal anaphora. In Bunt, H., editor, *Computing meaning*, volume 2. Kluwer, Dordrecht.
- Ginzburg, J., Gregory, H., and Lappin, S. (2001). SHARDS: Fragment resolution in dialogue. In Bunt, H., van der Sluis, I., and Thijse, E., editors, *Proceedings of the 4th International Workshop on Computational Semantics (IWCS-4)*, pages 156–172, Tilburg.
- Ginzburg, J. and Sag, I. (2000). *English Interrogative Constructions. Studies in Constraint-based Lexicalism*. CSLI Publications, Stanford, California.
- Gregory, H. and Lappin, S. (1999). Antecedent contained ellipsis in HPSG. In Weibelhuth, G., Koenig, J. P., and Kathol, A., editors, *Lexical and Construc-*

- tional Aspects of Linguistic Explanation*, pages 331–356. CSLI Publications, Stanford.
- Kay, M. (1996). Chart generation. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 200–204.
- McKeown, K. R. (1985). *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*. Cambridge University Press, Cambridge.
- Nicolov, N. and Mellish, C. (2000). PROTECTOR: Efficient Generation with Lexicalized Grammars. In *Recent Advances in Natural Language Processing*, Current Issues in Linguistic Theory (CILT 189), pages 221–243. John Benjamins, Amsterdam & Philadelphia.
- Pollard, C. and Sag, I. (1994). *Head Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications, Chicago.
- Purver, M. (2001). Adding a realistic lexicon to SHARDS. Technical report, Department of Computer Science, King's College London.
- Reiter, E. (1995). NLG vs. templates. In *Proceedings of the Fifth European Workshop on Natural-Language Generation (ENLGW-1995)*, Leiden, The Netherlands.
- Sag, I. (1997). English relative clause constructions. *Journal of Linguistics*, 33:431–484.
- Shieber, S., Pereira, F., van Noord, G., and Moore, R. (1990). Semantic-head-driven generation. *Computational Linguistics*, 16:30–42.