Acquisition of Conceptual Data Models from Natural Language Descriptions

William J.Black,
Department of Computation, UMIST,
PO Box 88, Sackville Street,
Manchester, M60 1QD, UK

Abstract

Acquiring information systems specifications from natural language description is presented as a problem class that requires a different treatment of semantics when compared with other applied NL systems such as database and operating system interfaces. Within this problem class, the specific task of obtaining explicit conceptual data models from natural language text or dialogue is being investigated. The knowledge brought to bear on this task is classified into syntactic, semantic and systems analysis knowledge. Investigations with a simple syntactic parse and with a semantic analysis using McCord's Slot Grammar are reported, and the structure of the systems analysis knowledge is considered.

1 Introduction

This section introduces the application of computer-based tools for information systems requirements analysis, design. and implementation, and outlines a motivation for endowing such tools with natural language interfaces. It concludes with the structure of the remainder of the paper.

Information systems development suffers from two widely acknowledged problems:

- an applications backlog, whereby demand for applications exceeds resources available for its satisfaction.
- a requirements analysis problem. This is often manifested as a maintenance problem, whereby resources that could be put into reducing the applications backlog are instead devoted to correcting faults in delivered systems. Most such faults are traceable to erroneous specifications, resulting from a failure to establish user requirements correctly.

The industry has provided solutions to each of these problems:

The problem of productivity has been addressed by the provision of more powerful higher-level languages known as 'application generators' or 'fourth generation languages ' (4GLs), in which the same functionality can be achieved in a tenth or less of the instructions needed in a conventional procedural computer language.

A general feature of such software tools is that they do away with the need for much procedural programming by employing *declarative* notations in which requirements can be expressed in sufficient detail for the software to provide procedures to meet them. (Naturally, the sophistication of these declarative notations varies according to the breadth of their application coverage.)

The requirements analysis problem has been addressed since the mid nineteen-seventies by a range of prescriptive development methods providing working procedures and graphical representation languages, in place of traditional approaches which rely heavily on natural language narrative to specify processing requirements. A typical representation or conceptual modelling language is described in section 2.

A problem with both the high-level application generators and the development methods is that they have been established independently by a variety of manufacturers, software houses, consultants and academics, resulting in a multitude of competing products and methods, with no standard accepted by the industry. Suppliers and users therefore face considerable training and consultancy costs due to staff mobility.

A more recent trend is to combine the two approaches to produce a more powerful software tool environment or analyst's workbench which enables the analyst to edit diagrams that formally represent the requirements, and using these specifications to automatically generate computer programs. Such tools represent an improvement on previous practice in two ways: firstly by bringing forward the use of precise formal languages from the coding to the specification phase in the software life-cycle, and secondly by automating the coding phase. However, they do not similarly *automate* the analysis phase that must precede the formal expression of requirements in a specification, although they may *mechanize* the process of recording and revising a specification.

The nature of such tools is described in section 3.

It is proposed that a natural language interface to tools provided to mechanize such methods would provide several benefits:

- It is possible to develop a specification using a representation language with which the analyst is not

familiar, by *hiding* the representation language from the analyst.

- It can alternatively help the analyst *learn* the specification language by displaying the graphical representation of a given description.

- With a natural language generation facility it is possible for an analyst to informally *verify* that a graphical representation of some aspect of a system expresses the desired meaning.

- A natural language generator can be used to *translate* a specification developed by someone else using a representation that the reader is not familiar with. In addition to facilitating communication between analysts trained on different methods, this technique could facilitate communication between analysts and their users or expert informants.

A further motivation is that increasingly, the 4GLs described above are in the hands of end users who develop applications directly. 4GLs typically make straightforward applications easy to develop, often prompting the users for the parameters that specialize the application as an instance of the stereotype systems that lie within the tool's application bandwidth. However, they are often poor at enabling more elaborate requirements to be met where there are interdependencies between data files and complex integrity and validation rules. A tool built on the lines described below can help end users inexpert in analysis to articulate their own requirements and then to convert those requirements into executable code.

The architecture of an information systems development workbench with an integrated natural language and graphics interface is described in section 4. An approach to knowledge representation within this system is discussed in section 5. Approaches to natural language analysis and generation and the results of some prototyping work are discussed in section 6.

2 Conceptual Data Models

Within system development methods, there exist several notations for the representation of aspects of the information processing systems. A comprehensive method will be able to represent a conceptual model of the real and abstract objects in the system's environment, the functional requirements of the system, and its detailed design and implementation. Methods differ according to their emphasis on the data the system deals with or its processing requirements. Here, we will only discuss data-oriented approaches, and in particular Entity-Relationship (ER) modelling(Martin 1984, MacDonald 1986), and the Information Structure Diagram within NIAM (Verheijen and van Bekkum, 1982; Blank and Krijger, 1982), an approach to conceptual modelling that claims to be directly informed by considerations of the structure of natural language sentences.

Both approaches started as paper-and-pencil notations for unambiguously recording the systems analyst's understanding of the relationships between objects that are significant to the proposed computer system. Now, however, both feature in interactive computer-based tools.

ER analysis is the cornerstone of several application development toolkits, including the *Information Engineering Workbench*, which uses such models as input to a process that ultimately leads to automatic code generation. Using this software, the analyst creates a model of data objects and relationships, and then specifies how programs will access this data by superimposing access paths and conditions on the ER diagram to produce a 'data navigation diagram' which can in turn be the input to an automatic process of computer program generation.

The NIAM approach is somewhat different. It aims to avoid altogether the need to describe the objectives of a computer program procedurally. The NIAM conception is that if all the relationships and dependencies among data objects are specified rigorously, all events in the world can be recorded in the database by the assertion or retraction of 'sentences' from the information base (database). Further, rules for deriving information, such as that produced in reports, can be included in the conceptual model. Under this approach, the need for application programs is obviated, since 'all computations may be executed under the control of the information base handler instead of an application program' (Blank and Krijger 1982, p 140). NIAM thus belongs to the 'executable specifications' class of approaches to system design. To illustrate NIAM's modelling language, we will use example sentences from a narrative in a database examination paper (1) and (2).

(1) Customers send the company purchase orders for pharmaceutical supplies.

(2) Each order contains requests for quantities of many different products which are all required for one shop.

2.1 NIAM Information Structure Diagrams

In this section, we describe the modelling constructs of the NIAM information structure diagram (ISD). The derivation of the model from natural language text is taken up in section 6.

The NIAM method uses similar constructs to ERA modelling, (employing different terminology) but at a more atomistic level. Its perspective is derived from the structures of natural language, on the justification that a database comprises a set of *sentences* and the purpose of the conceptual data model is to specify a *grammar* of the sentences allowed in a particular database (Blank and Krijger, 1982). Relationships between objects have associated with them two *role names*, one for each related object.

Figure 1 shows how (1) and (2), together with some further information about warehouses and picking lists, can be represented in NIAM notation.

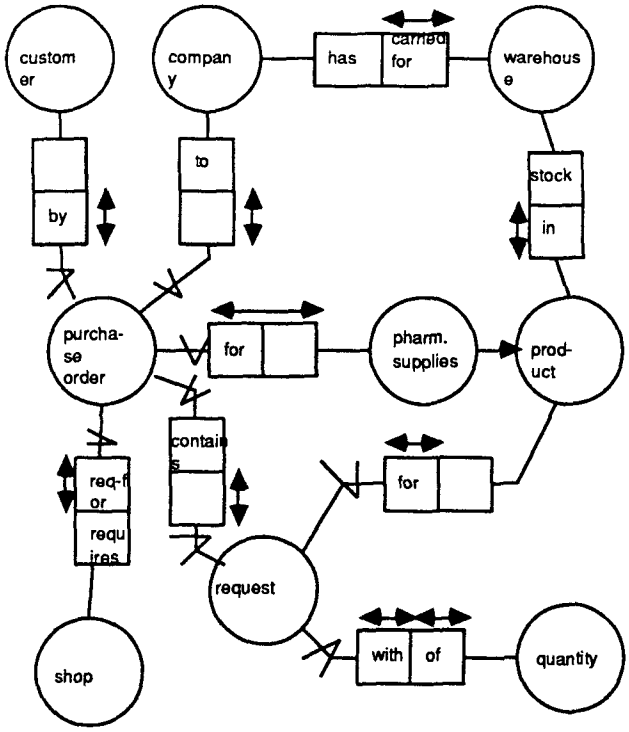


Figure 1 - NIAM Information Struct ure Diagram

NIAM representations are constructed from objects and relationships.

Objects are of two kinds: NOLOTs or NOn-Lexical Object Types are concrete or abstract objects of reality, and LOTs or Lexical Object Types are objects of which occurrences have values, i.e. they are names. NOLOTs and LOTs are synonyms for entities and attributes, and are shown by unbroken and broken circles respectively.

Relationships are associations between objects: either between two NOLOTs or between a NOLOT and a LOT. They are shown by lines connecting two circles. Semantically, the relationship is the Cartesian product of the two related object types. On the line are two rectangular boxes bearing the name of the role each object plays in the relationship. The concept of a role is similar to that of a case role in linguistics (cf Fillmore 1968).

In addition to portraying objects and their relationships, NIAM also explicitly represents constraints between these objects.

Relationship degree is shown by double headed arrows beside one or both roles. The relationship between customer and purchase order in Figure 1 is a one to many (1:N) relationship in that a customer may send many purchase orders, but each order is only sent by one customer. A many to many (M:N) relationship, such as that between purchase orders and pharmaceutical supplies, is shown by the arrow's spanning both roles. A one to one (1:1) relationship has separate arrows alongside each role.

Whether a relationship is obligatory or not is also shown in the diagram. The 'V' across the line between purchase order and the 'sent-by' role indicates that a purchase order cannot exist without being related in this way to a customer, but the absence of such a symbol at the opposite end of the relationship line shows that a customer can exist without having any (current) purchase orders.

Additional constraints, such as subset and set inequality constraints between objects, relationships or roles can also be modelled on the NIAM ISD. For example, the arrow linking pharmaceutical supplies to product indicates a subset relationship.

Often, M:N relationships are indicative that further analysis is required. Where such a relationship conveys genuine information, it is usually helpful to resolve the relationship into two 1:N relationships, with a new entity type between.

The M:N relationship in figure 1 between purchase order and pharmaceutical supplies was derived from sentence (1). In sentence (2), further information about orders was supplied. All the information conveyed by the M:N relationship is represented by the chain of 1:N relationships linking purchase order, request, product and pharmaceutical supplies.

3 Tools for Conceptual Modelling

Many proprietary tools exist for editing conceptual data models, e.g. *Excellerator*, *Information Engineering Workbench*, and *Blues*. The system enables the user to draw diagrams using a mouse input device. The user selects from the symbols in the notation by clicking the mouse button, moving the cursor to a desired location and clicking again. Lines connecting symbols can be selected in the same way and placed by clicking twice, to indicate the two symbols the line connects.

Violations of the 'syntax' of the notation are policed by the software.

Modifications to both the content and layout of a diagram can be made by cutting and pasting. Annotating components with their names and other attributes is done by clicking on existing symbols to open a dialogue window.

As the diagram is thus created and edited, the information expressed in it is stored in a data dictionary (or 'encyclopaedia').

It can be argued that such an interface is so user-friendly that no case could be made for a natural language alternative. However, it is emphasised that a tool as described above is entirely passive. It simply records the information fed into it, and can give no guidance as to the correct way to represent a given state of affairs. It can only be used by an expert in the method of analysis it documents. For such an expert, it is probably an optimal

tool. However, we have noted in section 1 that owing to the babel of alternative notations, there are circumstances in which experienced analysts are required to use methods they are not familiar with. This is the premise of the AMADEUS project (Loucopoulos *et al* 1986, Black *et al* 1987) which seeks to provide a facility for translating between alternative method notations. Briefly, the requirement to use unfamiliar methods can arise because of job mobility, organizational take-overs, customers dictating the method to be used by those who tender for their contracts, and in the course of training.

It is also envisaged that the system will be used by end users to develop applications without professional support. Figure 1 illustrates, by the variety of special symbols used and their connectivity, that for end-user application development, notations like the NIAM ISD would require an explanation facility to support comprehension. For a non-expert to use such a notation constructively to develop a specification also requires some form of expert assistance.

A final motivation for building the system is that as an integrated natural language and graphics interface, it provides a context in which the relative merits of the two interface styles can be compared. As Thompson (1983) has noted, almost no empirical work has ever been carried out into the relative merit of natural language and graphic interfaces.

4 Architecture of an integrated NL and graphics environment.

4.1 Dialogue Structure

A natural interface using both text and graphics requires a large bit-mapped screen and both keyboard and pointing input devices. An Apollo DN3000 running Quintus Prolog under UNIX has been selected as an environment for development of the system. The intended dialogue structure employs two windows, one for text and one for graphics. In both cases, highlighting is used for attention focussing and establishing correspondence between a diagram and natural language narrative.

Text to graphics. Appendix A shows an hypothetical dialogue where the user input is in the text window.

This dialogue owes much to the style of dialogue employed in *Nanoklaus* (Haas and Hendrix 1983), and would suit a very inexperienced or casual user. Someone more used to expressing rules in unambiguous English might be able to say most of the above in one sentence:

"A paper is written by one or more authors, one of which must also be its presenter, and any of whom may be the authors of other papers."

For this reason, the interface must have good syntactic coverage and a formal semantic component that deals with quantifier scoping.

Graphics to text. A dialogue where the input takes place in the graphics window proceeds as follows: The user

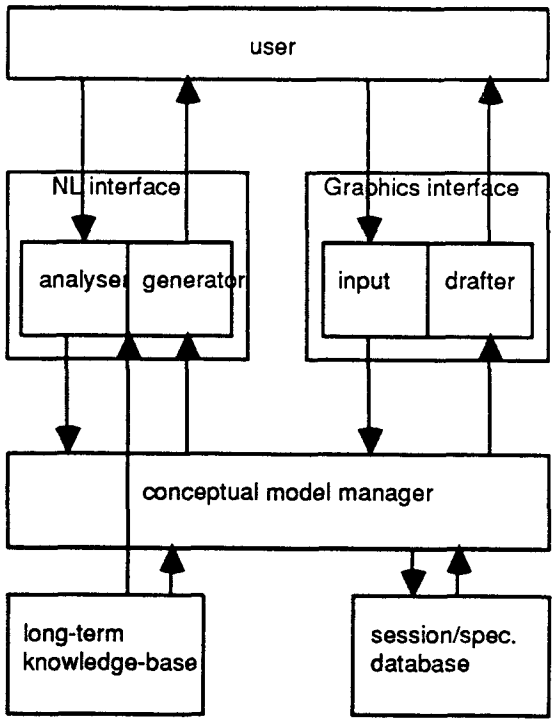


Figure 2 - System components and d ata flows

selects and places new symbols in the graphics window. For each symbol added, the change is recorded in the session database, and its internal representation is passed to the language generation component, which produces an English description of the effect of the changes. Suppose for example, that the graphics window contains the first drawing shown in Appendix A. The user then adds the V symbol to produce the next drawing shown. In response, the following text is produced:

"A paper must be written_by at least one author. (Previously it could apparently exist without being written by an author.)".

Alternative uses of the natural language generation facility exist. For example, a user could highlight a part of the diagram and request a translation into English, or could enter changes in a "what if" mode and have their consequences explained.

4.2 System structure

To produce a dialogue such as that shown in Appendix A or as described above, a system organization such as that shown in figure 2 is required.

Both user interfaces must use the same internal representation for the aspects of systems described alternatively in text or graphics. This is discussed below. The session/specification database is the counterpart of the data dictionary in individual proprietary tools. In such a system, the graphics interface is such an integral part of

the system that it along with the natural language interface requires to be re-implemented.

5 Knowledge representation framework

It has been established in the separate AMADEUS project (Black *et al* 1987) that a frame representation based on FRL (Roberts and Goldstein, 1977) is capable of representing all the modelling constructs used in a range of requirements specification notations. Specifically, in the case of NIAM, objects (lexical and non-lexical) and relationships are represented by frames, and roles by slots. Constraints of relationship degree and optionality are represented together by *facets* of role slots.

As an example, Figure 3 shows a set of frames representing some of the information about paper authorship shown in Appendix A.

It is intended that a uniform knowledge representation structure such as that shown in Figure 3 will be used throughout the system, both for storing the facts gathered in a session, and for representing the stored knowledge in the system, including the dictionary.

paper	(ako,**value**,object)
	(written_by,**value**,authorship)
author	(ako,**value**,object)
	(writer_of,**value**,authorship)
authorship	(ako,**value**,association)
	(written_by,**domain**,paper)
	(written_by,**min_card**,1)
	(written_by,**max_card**,1)
	(writer_of,**domain**,author)
	(writer_of,**min_card**,0)
	(writer_of,**max_card**,N)

Figure 3 Internal representation of objects and relationships.

6 Approaches to NL analysis and generation

Haas and Hendrix (1983) describe a system where a semantic network model of object classes, instances and properties is constructed through a co-operative natural language dialogue. In the early version, *Nanoklaus*, the syntactic coverage is restricted to simple sentences in which the user may assert propositions about the set membership and other properties of objects.

(Enomoto *et al* 1984) describes a system in which an unambiguous fragment of English (based on Montague's PTQ) can be used in a highly constrained way to describe the desired behaviour of a system.

Other work on natural language understanding of descriptive text has tended to use ad-hoc semantic grammars specialized to the application domain. Norton (1982) describes a program that acquires knowledge of the BASIC programming language's syntax and semantics from a textbook and uses this to generate an interpreter for part of the language. In some respects, the goals are similar to our own, but the semantic grammar approach used means that little of that apporach is re-usable.

Less directly related to the system specification domain is (Mellish 1985) which describes a system for the semantic interpretation of mechanics problems expressed in English. The program made use of the given/new distinction in establishing the co-reference of definite and indefinite descriptions, incrementally constructing extensional semantic interpretations using intermediate intensional reference entities.

Earlier work on text comprehension (e.g. de Jong, 1979) concentrated on skimming techniques to match text content against sketchy scripts. Such a grain of analysis is inappropriate for present purposes.

6.1 Conceptual Modelling from NL Text.

The goal of conceptual modelling is to identify the significant objects and relationships in the application universe of discourse. As with other NLU tasks, this requires knowledge of three sorts: syntax, semantics and real-world knowledge. In this section, we discuss the separate contribution each source of knowledge makes in conceptual modelling.

Syntax. Martin (1984) has observed that there is a simple mapping of surface syntactic categories onto the components of ER modelling. Nouns correspond to entities (objects), and verbs correspond to relationships (or in the case of NIAM, with role names). On this basis, sentences (3) and (4) would receive different analyses, as shown below.

(3) Customers send orders for products.
(4) Customers *order* products.

The English description in (2) is much less directly helpful in identifying relationships. The attachment of the relative clause *which are all required for one shop* to *order* rather than *product, request* or *quantity* cannot be decided on purely syntactic grounds. Further, that *quantity* is an attribute of *request* rather than an entity in its own right cannot be determined without extra-linguistic knowledge.

The requirements for a linguistic approach are that either is is constructed in the same manner as Nanoklaus, to employ simple input phrase structures, but embedded in a co-operative dialogue, or else it should have sufficient linguistic coverage to handle the complex sentence structures exhibited in (1) and (2). Most importantly in the latter respect, it should have a reasonable treatment of the variety of natural language quantifiers and relative clauses. Many database interfaces have such capabilities, McCord (1982), Dahl (1982) and Warren and Pereira (1982) *inter alia*.

Semantics. Charniak (1983) makes a distinction between inferential and non-inferential semantics. The former is concerned with establishing the logical form corresponding to a syntactic analysis of a sentence, whereas the latter is concerned with co-occurrence restrictions between phrases which may be stated in terms of lexical subcategories such as human, mass, machine, etc.

Database interfaces are the most common instances of complete natural language interfaces which comprise both syntactic and semantic components. As such they are potential models for the development of interfaces to new types of software systems. However, their approach to semantics cannot be imported wholesale. They avoid the general theoretical problem of what a semantics of natural language should consist of by an operational approach in which the propositional content of a sentence is represented by a database tuple, and lexical subcategorization is implemented in application-specific categories. The following dictionary entries, for 'order' both as a noun and a verb have been encoded in the notation used by (McCord 1982).

```
noun  (order,
       ord(O_no,Cust,Supp,C)
                  &item(O_no,Prod,Qty)
       nil,
       O_no:transaction,
       [npobj(for):Prod:goods]).

verb  (order,
       ord(O_no,Cust,Supp,C)
                  &item(O_no,Prod,Qty)
       C,
       Cust:prsn,
       [obj:Prod:goods,npobj(from):Supp:prsn]).
```

Each of these dictionary entries has five components. The first is the name of the word, the second is the propositional meaning, the third a variable denoting time, the fourth specifies the semantic subcategorization of the word (in the case of nouns) or its subject (in the case of verbs), and the last subcategorizes the objects or other postmodifiers the word may take.

One danger with application-specific lexical subcategorization is that it may be applied too restrictively. For example, in the lexicon published in (McCord 1982), subclasses are specifically restricted to the database entities that can be expected in a query. For example, the semantics of *take* are specified to expect a student as subject and a course as an object. Such restrictions are fine for database queries, such as (5) but a question such as (6) cannot even be asked.

(5) Which students took Logic?
(6) Do lecturers ever take courses?

Real world knowledge. It is not possible to produce an analysis such as that shown in Figure 1 without 'real-world' knowledge in addition to a grammar and dictionary. For example, the knowledge that pharmaceutical supplies are a subset of products is required to link the information acquired from the analyses of (1) and (2). The full extent to which real-world knowledge will be required in the system is not known, but it is assumed that the sort of notation shown in Figure 3 can be employed to encode arbitrary real-world knowledge for the system.

The boundary between what is linguistic knowledge and what is real-world knowledge is not a clear one. In the sample dictionary entries for order, we have shown that corresponding to an order, there is also an item. This was necessary so that the type of object can be linked to an argument place in the predication. It can be argued that this amounts to non-linguistic knowledge that orders typically comprise several distinct items.

Adapting a database interface. An initial prototype system for inferring the existence of entities and relationships from natural language descriptions is being constructed using McCord's Slot Grammar (McCord 1982), selected for its syntactic coverage and treatment of a variety of natural language quantifiers.

To adapt the form of lexical entries in the McCord parser from the database query task to the present one, generic definitions of word meanings have been provided, allowing a wider range of assertions to be made.

Results. With these definitions it has been possible in a rudimentary way to determine the existence of some relationship types between entities to build simple ER models. This is done by examining the attributes of the relational database predicates in the parse tree. The existence of a relationship between two database relations, is indicated by the sharing of attributes. If the identifier of one relation occurs as a non-identifying attribute in another relation, we may infer a 1:N relationship between them. For example, in the following parse of the sentence "customers order products" the variable _133 is common to both order and customer:

```
[customers,order,products,.]
[s,dcl] main ord(_195,_133,_197,_198)
  [np,pl] indef:_133:prsn cust(_133,_134,_135)
  [advc] conjunct present(_198)
  [np,pl] indef:_199:goods prod(_199,_244,_245)
```

This occurs precisely because the dictionary entry for "order" explicitly provided for the identier of the subject to be an argument of the predication.) The sharing of the arguments tells us that a relationship exists between the entity order and the entity customer, and furthermore, it is a 1:N relationship from customer to order, since the shared argument is the whole key of customer, and either a non-key or part key in order.

Current status of project. The prototyping activity described above is ongoing, but in parallel, the overall design is being elaborated, and a purpose-built parser based on LFG is being implemented in Prolog. Work on the

generation component has not yet commenced.

7 Summary and Conclusions

This paper has outlined an application area that can serve as a test-bed for work on the processing of natural language text for the purpose of knowledge acquisistion, a problem that is much wider than the specific case of information systems design. The role of natural language analysis and generation within an environment that also supports a WIMPs (Windows, Icons, Mice and Pointers) interface was defended, and the architecture of the software outlined. Finally, the relative place of syntactic, semantic and real-world knowledge in conceptual modellingwas discussed, and the adaptation of a database interface to the analysis component of the system was described.

Acknowledgements

Edem Williams carried out some lexicographic work and implemented a version of the McCord Slot Grammar parser, and carried out some preliminary work on identifying objects and relationships under this approach. I am grateful to Donal Flynn for discussions about NIAM.

References

Black, W.J, Sutcliffe, A.G., Loucopoulos, P and Layzell, P.J. (1987) Translation between pragmatic software development methods. Submitted to ESEC87, Strasbourg.

Blank and Krijger (eds) (1982). Evaluation of methods and techniques for the analysis, design and implementation of information systems., Academic Service, pp 137-156.

Dahl,V (1982) On database systems development through logic, *ACM Transactions on Database Systems* **7**, 102-123

Enomoto, H, Yonezaki,N, Saeki, M and Aramata, H (1984) Natural Language Based Software Development System TELL, ECAI-84.

Fillmore, C J (1968) 'The Case for Case' in Bach and Harms (eds) Universals in Linguistic Theory, Holt, Reinhart and Winston.

Haas, N and Hendrix, G.G. (1983) Knowledge acquisition from Interactive Dialogue, in Michalski, Mitchell and Carbonell (eds) Machine Learning 1, Tioga.

Loucopoulos, P; Black, W.J; Layzell, P.J; Sutcliffe, A.G. (1986) AMADEUS : A multi-method approach to developing universal specifications. Paper presented to ESPRIT Technical Week, September 1986.

MacDonald, I.G. (1986) Information Engineering - An Improved, Automatable Methodology for the design of Data Sharing Systems, In Olle, T.W *et al* (eds) Information Systems Design Methodologies: Improving the Practice, North-Holland.

Martin, J. (1984) Information Engineering, Savant Institute.

McCord M.G. (1982) Slots and Modifiers in Logic Grammars, *Artificial Intelligence* **18**, 327-367.

Mellish, C (1985) Computer Interpretation of Natural Language Descriptions, Ellis Horwood.

Norton, L.M. (1982) Automatic Analysis of Instructional Text, *Artificial Intelligence* **20**, 307-344.

Thompson, H (1983) Natural language processing: a critical analysis of the field, with some implications for parsing. in Sparck-Jones and Wilks (eds) Automated Natural Language Parsing, Ellis Horwood, Chichester.

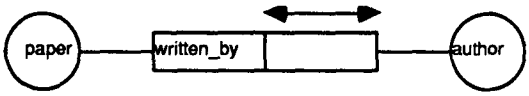
Verheien, G.M.A and van Bekkum, J (1982) NIAM: An Information Analysis Method, In Olle, T.W *et al* (eds) Information Systems Design Methodologies: A Feature Analysis, North-Holland.

Warren, D.H.D and Pereira, F.C.N (1982) An efficient easily adaptable system for interpreting natural language queries *AJCL* **8**, 110-122.

Appendix A

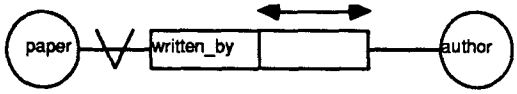
Sample dialogue

user A paper can be written by several authors



sys Must a paper be written by at least one author?

user yes

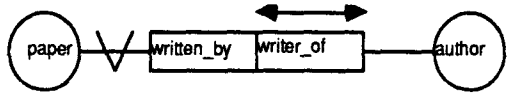


sys How would you describe the role of the author with respect to the paper, e.g.

The author _____s the paper, or

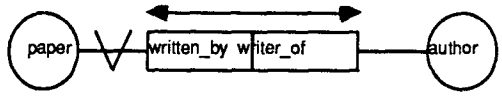
The author is _____ of the paper?

user writer_of



sys Can the author be the writer_of other papers?

user yes



sys ok

user The paper must be presented by one of the authors.