From English to Formal Specifications

SUNIL VADERA AND FARID MEZIANE

Department of Mathematics and Computer Science, University of Salford, Salford M5 4WT, UK

Formal methods provide an approach in which design steps can be shown to satisfy a specification. However, if a formal specification is wrong, then although the design steps may satisfy the formal specification, they are unlikely to satisfy the requirements of the system. Since most users are unfamiliar with formal methods, requirements specifications are often written in English. Such requirements, expressed in English, are then somehow translated to formal specifications. This transition has some potential for introducing errors and inconsistencies.

In this paper we propose an interactive approach to proceeding from an informal specification to a formal specification in a systematic manner. The approach uses research in the area of natural language understanding to analyse English specifications in order to detect ambiguities and to generate an entity relationship model. The entity relationship model is then used as a basis for producing VDM data types and the specifications of some common operations.

We illustrate the effectiveness of our approach by applying it to the specification of part of a route planning database system.

Received May 17 1994, revised October 17 1994

1. INTRODUCTION AND MOTIVATION

The use of formal methods in software development has been advocated as a way of improving the reliability of software. A formal development life-cycle begins with a formal specification. Design steps can then be proved with respect to their specification. This verification of design steps against their specification provides the primary benefit of formal methods—namely that design errors are detected earlier in the life cycle and are not propagated further down the life cycle.

Thus specifications provide the foundation upon which a system can be formally developed. However, if a formal specification is wrong, then although the design steps may satisfy the formal specification, they are unlikely to satisfy the requirements of the system.

The specification of a system involves people of different profiles who favour different representations. At the beginning, natural language is used because the specification acts as a contract between the users and the developers. Most of the time, the only representation that users understand and agree on is natural language. At the other end, developers find natural language specifications ambiguous and incomplete and may therefore prefer formal specifications. The transition from informal specifications to formal ones can be an error prone and time consuming process. This transition must therefore be supported to ensure that the formal specifications are consistent with the informal ones.

Hence, we propose an approach that aims to improve the process of producing formal specifications. The approach aims to use research in the area of natural language understanding in order to analyse specifications written in English and then attempts to produce VDM data types. We believe that if successful, this approach will:

- help to ensure some consistency between the informal specifications and their formal versions.
- help with the process of identifying data types from English specifications in a systematic manner. This differs from existing approaches where the process of producing the data types is informal and perhaps even mystical.
- help an analyst identify ambiguities and incompleteness in English specifications.

We do not, however, believe that our approach will somehow enable an analyst unfamiliar with formal methods to produce acceptable formal specifications. Nor do we believe that such an approach can somehow produce complete formal specifications from written specifications without an analyst's involvement. The approach is likely to be most useful in situations where an English specification already exists or is being developed before it is formalised.

2. AN OVERVIEW OF THE APPROACH

Our approach is based on the view that much of the work in the area of natural language understanding can be utilised to aid the production of formal specifications. In particular, much attention has focused on the problem of handling ambiguities and quantification. For example, Warren and Pereira's [19] system for interpreting natural language queries for a geographical database and McCord's [14] work for a student database both tackle the problems of quantification and ambiguities in English text.

For our purposes, we adopt McCord's approach to natural language processing [14] since it extends the approach taken by Colmerauer [5], Pereira [16], and Warren and Pereira [19]. As we illustrate below,

McCord's approach appears to be more suitable for handling the kind of quantification problems one is likely to encounter in producing formal specifications.

McCord's approach proceeds in two phases. First, he produces an analysis based on the grammar rules of the language, and then he uses a semantic analysis phase to produce statements in a meaning representation language called Logical Form Language (LFL). For example, the sentence:

"All companies maintain a stock system."

results in the logical form:

$$all(company(X), ex(stock(system(Y)), maintain(X, Y)))$$

where the first arguments of *all* and *ex* define the kind of objects over which the expression in the second argument is quantified. McCord uses the term *base* to refer to the first argument, and *focus* to refer to the second argument. As this example shows, LFL expressions can be nested in a manner similar to expressions in predicate logic. As a more complex example consider the sentence:

"The company maintains a description for each item of stock."

This produces the logical form:

$$all(item(X, stock), the(company(Y),$$
$$ex(description(Z), for(X, maintain(Y, Z)))))$$

where the indefinite article "a" is translated to the existential quantifier, "each" to the universal quantifier and the form of the definite article is kept.

As this example illustrates, McCord's approach improves upon the use of Definite Clause Grammars (as used in Prolog) since the order in which the words occur in the sentence differs from the order of their quantification in the logical form. Given such a capability for analyzing sentences, we can summarise our approach as in Figure 1. The written specification is taken as an input and logical forms are produced for each sentence. A sentence may be ambiguous and can therefore result in several logical forms. Hence, an analyst would be made aware of any ambiguous sentences and would have to select the logical forms that best represent the intended meaning of the sentences.* Such logical forms then act as a basis for producing the data types. Further, based on the data types there are some common operations (e.g. adding, deleting, updating) whose specifications can be produced automatically by the system.

As the diagram suggests, the process of producing a formal specification from an informal one is iterative. We can not assume that the original specification is complete or that it is consistent. If an analyst finds ambiguities, he or she may reword a sentence in the original specification. Likewise, if an analyst believes that the system has not produced a complete entity relationship model, she may add sentences to the original specification.

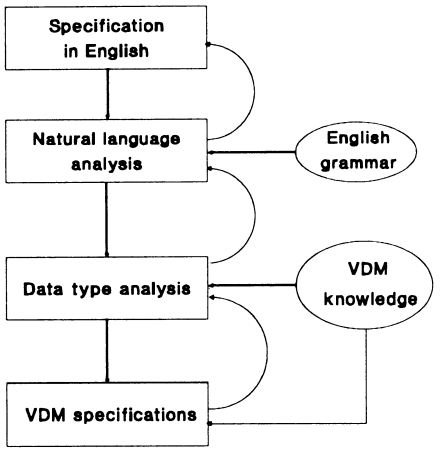

FIGURE 1. Overview of the approach.

The following sections describe each phase of our approach. Section 3 describes how the entity relationship model is identified, and section 4 describes how we obtain the VDM data types. Section 5 shows how we produce specifications of operations in the Vienna Development Method (VDM) [12]. Section 6 illustrates the effectiveness of the approach on an English specification that was written independently of our work. We refer the reader to [15] for a comparison of our approach with related research (e.g. with Balzer's approach [3], and Loucopoulos and Champions' work [13]).

We assume that the reader is familiar with VDM but include those aspects of logical form that are required for understanding this paper. However, the reader can refer to McCord [14] for a tutorial account of logical form.

3. IDENTIFYING THE ENTITY RELATIONSHIP MODEL

The first task in identifying a data type is to obtain the entities in the specification and the relationships between the entities. We base our identification process on the view that the nouns and verbs present in the English document contain the entities and relationships [8, 11]. For example, consider the sentences:

1. The pilot chooses the waypoints from the air.
2. A complex aircraft uses a radar.

The nouns suggest the entities:

$$\{pilot, waypoint, air, complex\ aircraft, radar\}$$

and the verbs the relationships:

$$\{choose, use\}$$

However, such a set of verbs and nouns may contain irrelevant entities and relations. Hence, we allow an analyst to filter such a list of entities and relations before proceeding.

*An approach that concentrates solely on improving English specifications, could reduce the need for familiarity with formal methods by paraphrasing the logical forms.

We can, of course, obtain a list of nouns and verbs by simply scanning the text. However, this approach does not help one to:

1. Find the relationships between entities. For example in the sentence:

   "A company maintains a description for each item of stock."

   although we can list the nouns as "company", "description", "item" and "stock", we do not obtain a relationship between these entities.
2. Extract compound nouns without ambiguity. For example, in the sentence:

   "A computer-assisted flight planning system is used by a complex aircraft."

   "planning" is a part of the compound noun "computer-assisted flight planning system" whose form must be preserved when translated to logical form.
3. Define the degree of the relationships between the identified entities (i.e. whether the relationships are one-to-one, many-to-one, etc.).

The following subsections show how the use of logical form enables us to overcome these problems.

3.1. Identifying the entities

The nouns form the basic list of entities. Below, we see how entities can be extracted from sentences that contain simple and compound nouns.

3.1.1. Simple Nouns

Simple nouns are extracted from noun phrases containing just one noun. For example, the sentence:

"The aircraft may hit an obstacle."

contains two noun phrases: "The aircraft" and "an obstacle". Each noun phrase is composed of a unique noun and each noun is extracted as an entity with its associated quantifier.

Relational nouns are also extracted in a similar fashion. For example the sentence:

"The system of an aircraft can be considered to comprise the plan of the pilot."

results in the entities:

$$\{system, aircraft, plan, pilot\}.$$

Proper nouns identify particular objects and therefore do not normally constitute entities. Hence in a sentence like:

"An example route is planned for a flight from Blackpool to Doncaster."

"Blackpool" and "Doncaster" do not constitute entities.

3.1.2. Compound nouns

Compound nouns are nouns which are composed of two or more nouns or a combination of nouns and adjectives. For example, consider the sentences:

1. A complex aircraft uses a radar.
2. The flight planning software package calculates the route tracks.

In the first sentence, the noun phrase "A complex aircraft" is composed of the adjective "complex" and the noun "aircraft". In the second sentence, the noun phrase "The flight planning software package" is composed of four nouns. In both sentences, the whole noun phrase is extracted as an entity. That is, the entities identified are: "complex aircraft" and "flight planning software package". Identifying entities by using just the head noun may, of course, lead to confusion. For example, in a specification of an aircraft system, both the description of a simple aircraft and a complex aircraft may occur.

3.2. Identifying relations

A natural way of identifying relationships is to use verbs and relational nouns.

3.2.1. Identifying relationships within relational nouns

Relational nouns always define relationships between nouns. Consider the sentences:

1. The company maintains a description for each item of stock.
2. The system of a simple aircraft can be considered to comprise the plan of the pilot.

In the first sentence, there is a relation between "item" and "stock". In the second, two relations are defined: the first between "simple aircraft" and "system"; the second between "plan" and "pilot". The relations extracted are shown in Figure 2.

3.2.2. Identifying relationships within verb phrases

Verbs generally refer to actions, events and processes [11]. In particular, transitive verbs define relationships between two entities. Let us consider the sentences:

1. The pilot chooses the waypoints from the air.
2. The system of a simple aircraft is considered to comprise the plan of the pilot.

In the first sentence, the verb "chooses" relates the entities "pilot" and "waypoints". This information is readily available from the logical form of the first

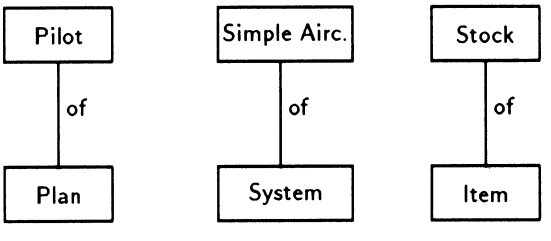

FIGURE 2. Relationships extracted.

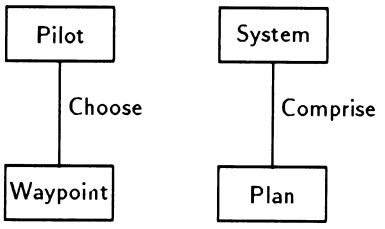

FIGURE 3. Verb relations extracted.

sentence:

$$the(pilot(X), the(waypoint(Y), the(air(Z), from(Z, choose(X, Y)))))$$

where *choose* relates the variables X and Y which are defined to be of type *pilot* and *waypoint*.

The second sentence has two verbs, making it a little more complex to analyse. The verb "comprise" introduces the main action (which is represented in natural language as an infinitive complement of the verb "consider"), and is therefore extracted as the relationship between the "system of a simple aircraft" and the "plan of the pilot". The verb "consider" plays a subsidiary role and does not relate any entities. Once again this information can be extracted from the logical form of this sentence, which takes the form:

$$the(system(X), ex(aircraft(Y)\&simple(Y)\&of(X, Y), the(plan(Z), the(pilot(T)\&of(Z, T), be(X, consider(X, comprise(X, Z)))))))$$

In cases like this, where the logical form contains more than one verb, the inner verb phrase identifies the relationship between the entities. The relations extracted for the above sentences are given in Figure 3.

3.3. Quantification and the determination of the degree

Early attempts at natural language analysis assumed that quantifiers occurred explicitly in the text. Thus it was assumed that the presence of a universal quantifier was always indicated by words like "every" and "all", and the presence of an existential quantifier was indicated by words like "some" [9]. However, many sentences are implicitly quantified by articles.

In this subsection we first examine how such implicit quantifiers can be identified and then show how quantified LFL statements can sometimes aid the identification of the degree of a relation.

3.3.1. Identifying implicit quantifiers

Most studies of quantification identify quantifiers from the articles present in the sentences [1, 9]. Initial studies of quantification regarded both definite and indefinite articles as existential quantifiers. More recent studies have shown various problems with this assumption and have shown how indefinite articles can also lead to universal quantifiers. Below we present our approach to identifying quantifiers from the definite and indefinite articles.

The definite article "the"

McCord translates the definite article "the" into the unique existential quantifier. For example, in the sentence:

"The best student was awarded a prize."

we can interpret "The" as the unique existential quantifier.

However, McCord acknowledges that sometimes "the" should be translated to the usual existential quantifier but does not give any guidance. In the case of obtaining the meaning of sentences in a requirements document, we cannot assume that one of these interpretations holds throughout the application. For instance, consider the sentences:

"The students passed the exams."

"The student passed the exams."

The first sentence does not suggest the unique existential quantifier, while the second does not suggest the normal existential quantifier.

As we will see later, obtaining appropriate quantifiers is an important prerequisite for our approach to identifying the degrees of relationships. Hence, we have attempted to improve upon McCord's approach to this problem.

In our approach, we do not simply translate "the" into the unique existence—instead we use the singularity or plurality of the noun to determine if it should be translated to the unique existence or normal existence. That is, if the quantified noun is singular, we adopt the unique existence, otherwise we interpret it as the usual existential quantifier.

We concede that there remain sentences for which these approaches remain inadequate. For instance, the following example given by Hess [9] is not covered:

"The unicorn is a mythical creature."

In this sentence, we do not presuppose the existence of unicorns, but the sentence nevertheless makes perfect sense.

The indefinite article "a"

The use of the indefinite article as a quantifier is always a source of ambiguity [1]. The indefinite article can sometimes be translated to the existential quantifier and sometimes to a universal quantifier. For our work, we adopt the approach proposed by Hess for interpreting the indefinite article [9]. The primary rule suggested by Hess takes the form:

Rule 1:

The subject of a sentence is existentially quantified if the verb phrase is in the past tense, or in the

progressive aspect, or in the perfective aspect. Otherwise the subject is universally quantified.

For example, consider the following sentences:

1. A text editor *makes* modifications to a text file.
2. A text editor *made* modifications to a text file.

In the first sentence, the "text editor" is universally quantified since the present tense is used but it is existentially quantified in the second sentence since the past tense is used.

Hess also suggests more specialised rules which can be summarised as follows:

Rule 2:

A subject is universally quantified if the past tense is used and the verb requires a spatial or a temporal postmodifier. A subject is also universally quantified if the verb is in progressive form and is modified by expressions such as "always", "in general" and "regularly".

Rule 3:

In a restrictive noun phrase, those arguments that are referred to by the main verb are universally quantified and those that are not referred to by the main verb are existentially quantified.

For example, in the sentences:

1. A man who loves a woman is happy.
2. A man that loves a woman respects her.

the third rule suggests that "woman" should be considered to be existentially quantified in the first sentence and universally quantified in the second sentence.

3.3.2. *Obtaining the degree from the quantifiers*

In this subsection we show how the quantifiers associated with each entity can be used to determine the degree of a relationship. It is not always possible to determine the degree of a relation from the quantifiers. However, we describe how our system gives a default degree for some cases. Of course, the user is allowed to override the system determined degrees.

Consider the following examples and their logical forms:

1. A complex aircraft uses a radar.

$$all(aircraft(X)\&complex(X), ex(radar(Y), use(X, Y)))$$

2. The company maintains a unit cost for each item of stock.

$$all(item(X, stock), the(company(Y), ex(unit(cost(Z)), for(X, maintain(Y, Z)))))$$

3. The students passed the exam.

$$all(student(X), the(exam(Y), pass(X, Y)))$$

In the first two examples, the first entity in the relation is quantified by the universal quantifier and the second by the existential quantifier. In the examples we have encountered, usually only one occurrence of the variable quantified by the existential quantifier is involved in the relation. Hence, based on our current experience, in such cases we interpret the logical form quantifier "ex" as referring to the quantifier $\exists!$ which denotes the unique existence. That is, many occurrences of the first variable are related to only one occurrence of the second variable. Then, by definition, we have a many-to-one relationship from the first entity to the second. In the third example, our interpretation of "the" results in a many-to-one relation between "student" and "exam".

Some one-to-many relationships can also be detected. For example, consider the following sentences and their logical forms:

1. The company maintains a description for each item of stock.

$$all(item(X, stock), the(company(Y), ex(description(Z), for(X, maintain(Y, Z)))))$$

2. The student passed all exams.

$$the(student(X), all(exam(Y), pass(X, Y)))$$

In the first sentence, the phrase "each item of stock" suggests that we are talking about one stock system that contains many items (i.e., a one-to-many relation between the entities "stock" and "item").

Sentences where the first entity is singular and quantified by "the" define one-to-many relationships. The second sentence is a typical example. An exception to this rule occurs when the second entity is also quantified by "the" and is singular. In this case, we infer a one-to-one relationship between the entities.

We have now given several cases in which we can identify the degree of a relationship. In other cases, when it is difficult to predict the degree of a relation, we let the analyst identify the degree.

At this stage we should have a complete list of entities and relationships, and therefore the entity relationship model. The next section shows how we can obtain VDM data types from such models.

4. PRODUCTION OF VDM DATA TYPES

In general, the entity relationship model produced as a result of the above process will be quite complex. As an example, appendix B contains the diagrams obtained for a problem that we illustrate in section 6. As the diagrams show, we may have several sub-models. The diagrams may contain many-to-many relationships as well as one-to-one, many-to-one, and one-to-many relationships.

The process of translating such diagrams to VDM and Z has already been studied by a number of authors (e.g. [6, 17]). Since our approach is similar to these studies, we summarise our translation and refer the reader to these

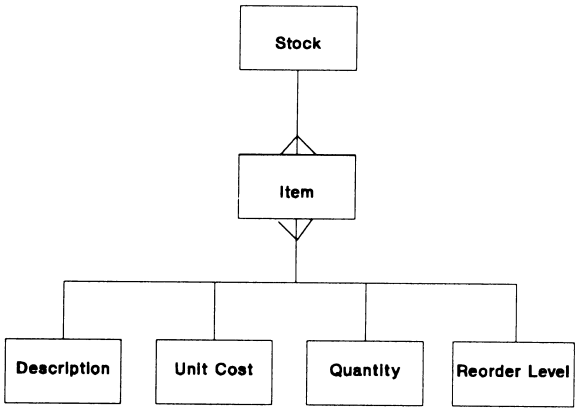

FIGURE 4. The stock problem.

papers for a more detailed account of the problems involved.[†]

We can model many-to-many relationships as sets of pairs. However, as with SSADM [2] we require the user to have resolved many-to-many relationships so that our entity relationship diagram only contains one-to-one, one-to-many and many-to-one relatiqnships. We model these relationships as follows:

- One-to-one relationships are modelled as one-to-one maps in VDM.
- The many sides of a one-to-many relationship can be modelled by using a map, a sequence or a set. Based on the examples in the literature (e.g. [12, 18]), we prefer the first of these approaches, since VDM's map data type appears to be more natural for modelling this kind of relationship. However, when the user specifies that the order of the items is important, the second option is adopted.
- A many-to-one relationship is modelled by a map from the many side to the one side.

In addition to these simple transformations, we also need to consider situations where there is a many-to-one mapping to leaf entities. Benyon [4] argues that such leaf entities should be attributes of the parent entity. Hence, in such situations we use a composite object to model the relationships.

As an example, the following data type is obtained when the above transformations are applied to Figure 4:

Stock_t = *Item-ID* → *Item_t*
Item_t :: *description* : *Description_t*
unit-cost : *Cost_t*
quantity : *Quantity_t*
reorder-level : *Level_t*

5. THE SPECIFICATION OF COMMON OPERATIONS

The development of specifications in VDM can be very complex. It is not our intention to develop a tool that produces all possible specifications. However, there are several specifications that are common across applications. These include the specification of operations that add items, delete items and list items that satisfy requirements. In this section we describe how such specifications are generated once the preceding step has identified the data types.

The general format of an operation specification in VDM is as follows:

OPER (input:In_t) output:Out_t
`ext` *...state...*
`pre` *...input...state...*
`post` *...input...output...state...*

The first line is called the signature of the operation. The signature is composed of the name of the operation (*OPER*), a list of input parameters and their types and a result and its type. The second line records those state variables to which an operation has external access. These state variables can be read only (rd) or read and write (wr) and the name of each variable is followed by its type. The pre-condition of an operation records assumptions about the arguments and state variables to which it is to be applied. The post-condition relates the before and after states and is an assertion that is required to hold after the operation is applied.

We can view this format as a template that needs to be filled to obtain a specification. In general, the template used is dependent on the operation required and the data type identified. Thus to add an item to a map we provide a template which specifies that:

- there is one input argument (the item to be added), and one output argument (the identifier of the item added),
- a state variable with write access (the map),
- no precondition,
- a postcondition that records the requirement that the identifier of the added item is new and that the map has been updated appropriately.

The information required for naming the arguments and the types is readily available as a result of the previous phase that identifies the types. Specifying the same operation for a sequence or a set is not substantially different from the specification for a map. We also adopt a similar template based approach to obtain specifications for deleting and updating maps and sequences (see [15] for details). The next section shows how our system works in practice.

6. A CASE STUDY: A FLIGHT PLANNING DATA BASE

The approach described in this paper has been implemented in Prolog-2 [7]. This section illustrates the use of the implementation to develop a specification of a database for a flight planning system from an English

[†]A related aspect, that of producing data type invariants, is mentioned in the section on future work.

specification. The English specification was written without prior knowledge of our work and was written independently of us.

Our current implementation of LFL does not handle conjunctions or pronoun references. Hence sentences that contain conjunctions in the original text are transformed so that individual conjuncts form separate sentences, and pronoun references are identified explicitly. In addition, the original text contained a table which provides structural information. This information is represented by adding three sentences. Appendix A contains the transformed text together with an identification of the kind of transformation.

This transformed text can now be analysed by the system. For each sentence, the system attempts to produce a logical form. When a sentence is ambiguous, the system displays the different logical forms and requires the user to select the logical form that corresponds to the intended meaning. For example, when analysing the sentence:

"The pilot draws the tracks of the route on the map."

the system displays three different logical forms that correspond to the following meanings:

1. "The pilot draws (the tracks of the route) on the map". That is, the information is drawn on the map.
2. "The pilot draws (the tracks of the route on the map)". That is, the tracks are already on the map and the drawing is done somewhere else.
3. "The pilot draws the tracks of (the route on the map)". That is, the route is given on the map and its tracks are drawn somewhere.

The user can then select the intended meaning. In this case study, out of the 41 sentences analysed:

- Twenty one sentences are not ambiguous.
- Eleven sentences produce two interpretations each.
- Nine sentences produce more than two interpretations.

Of the nine sentences that have more than two interpretations, the system produces reasonable interpretations for seven of them. However, two of the sentences result in an unexpected number of interpretations. These sentences are:

1. The planned tracks will assure the safe arrival of the aircraft over Doncaster when they are flown in correct order by the aircraft.
2. The pilot chooses the waypoints from Blackpool to Doncaster in a complex aircraft.

The first sentence results in 32 interpretations and the second sentence results in 12 interpretations. If we analyse the first sentence, we find that it is composed of two conjuncts and four prepositional phrases. Our parser produces 32 different syntax trees and each syntax tree results in a logical form. Likewise, in the second sentence, the presence of three prepositional phrases produces 12 syntax trees and each syntax tree is translated to a logical form. These two examples clearly demonstrate that the use of multiple prepositions can lead to sentences that are difficult to interpret by a natural language understanding system. Not surprisingly, such sentences are also difficult for a human reader. Indeed, style checkers, like the one employed by Grammatik [10], discourage the use of multiple prepositions. In our approach we could encourage the user to reformulate those sentences that have more than five interpretations.

In the next stage the system suggests 55 entities and 52 relations to the user. The user is allowed to remove any unexpected entities and relations. In this case study, the only suggested entity that we remove is "Whole range". We remove this since "Whole range" is a pre-modifier of the noun phrase "Electronic equipment" in the text (19th sentence in appendix A). These combined relations result in the entity relationship model given in appendix B.

Given an entity relationship model, a user may notice incomplete parts. For example, expected relations may not be identified by the system. In this case study the initial entity relationship model produced does not show a relation between the entity route and the entity waypoint. Although we understand that a route is composed of waypoints, there is no explicit sentence giving this relationship. Hence, we could add the following sentence:

"The route is composed of waypoints."

which would reduce the incompleteness of the specification as well as the entity relationship model.

The next stage aims to produce a VDM data type from the entity relationship model. In this example, suppose that we are interested in modelling the *Route* and therefore select the sub-model given in Figure 5. Further, we prune this sub-model by removing the *Number* and the *Track* entities that were obtained from the English document but which are not relevant in this case.

The model is now clearly defined. The user is asked about the importance of the order in the definition of the *Route*. In this case, suppose we think that the order is important, then the system produces the data type:

$$Route\_t = Waypoint\_t^*$$

The last stage is the specification of the operations. The pre-defined templates enable the generation of the specifications given in appendix C.

7. CONCLUSION

We have proposed an approach to the development of formal specifications from English specifications. The approach takes advantage of research in the area of natural language understanding in order to identify ambiguities and incompleteness in written specifications. The logical forms of the English sentences are used to obtain an entity relationship model which is then used to produce VDM data types. Although the process of

producing formal data types from entity relationship models has been widely studied, we believe that our process of producing the entity relationship models is new.

We have implemented our approach on a restricted grammar and have used it to specify part of a flight planning database system. The system manages to highlight potentially syntactic and semantic ambiguities. The entities and relationships produced appear to be appropriate. Most of the degrees are identified automatically by the system. The entity relationship diagram produced is broader than the part of the system we were seeking to model. Such broader diagrams would encourage analysts to gain a better understanding of the environment in which the specified component would operate. The VDM data types and the specifications produced are also appropriate for this example.

To conclude, our initial case studies suggest that the approach we have proposed is promising. In particular, we believe that the approach suggested could improve the process of producing specifications.

8. FUTURE WORK

At present there are several weaknesses of the approach that we are addressing. These include:

- The production of data type invariants. One approach that we are currently attempting is to translate an invariant written in English into logical form and then to VDM. This approach, described in [15], involves relating the logical form of the invariant to the identified data type. For the case study discussed in this paper, it is able to translate the sentence:

  "All adjacent waypoints are different."

  into the following invariant:

  $$Route\_t = Waypoint\_t^*$$

  $$inv\text{-}Route\_t(route) \triangleq \forall i1 \in \texttt{inds}\ route \cdot \forall i2 \in \texttt{inds}\ route \cdot adjacent(i1,i2) \Rightarrow route(i1) \neq route(i2)$$

  The analyst then has to define *adjacent*:

  $$adjacent: \texttt{N} \times \texttt{N} \rightarrow \texttt{B}$$

  $$adjacent(m,n) \triangleq m = n + 1 \vee n = m + 1$$

- The production of a wider range of specifications. At present the system produces a small predefined range of specifications and the user is left to develop his or her own specifications given the data types. The template based approach given in this paper is limited and one could attempt to translate given English pre and post conditions to their logical versions.

9. ACKNOWLEDGMENTS

We would like to thank Colin Parker of British Aerospace for contributing the case study by Brian Hepworth which we used to test our approach. We would also like to thank the three anonymous referees whose comments have improved the presentation of this paper.

REFERENCES

[1] J. Allen. *Natural Language Understanding*. The Benjamin/Cummings Publishing Company, Inc, 1987.
[2] C. Ashworth and M. Goodland. *SSADM: A Practical Approach*. McGraw-Hill Book company, 1990.
[3] R. Balzer, N. Goldman, and D. Wile. Informality in program specification. *IEEE Transactions on Software Engineering*, SE-4(2):94–103, 1978.
[4] D. Benyon. *Information and Data Modelling*. Blackwell Scientific Publications, 1990.
[5] A. Colmerauer. Metamorphosis grammars. In L. Bolc, editor, *Natural Language Communication with Computers*. New York Springer-Verlag, 1978.
[6] J. Dick and J. Loubersac. Integrating structured and formal methods: A visual approach to VDM. In *Third European Software Engineering Conference*, LNCS 550, pages 37–59, 1991.
[7] T. Dodd. *An Advanced Logic Programming Language-Prolog-2 User Guide*, volume 1. Intellect, 1990.
[8] C. Gane and T. Sarson. *Structured Systems Analysis*. Prentice-Hall, 1979.
[9] M. Hess. How does natural language quantify? In *Second Conference of the European Chapter of the association for Computational Linguistics*, pages 8–15, 1985.
[10] Reference Software International. Grammatik 5, 1992.
[11] H. Jackson. *Analysing English*. Pergman Press, 1982.
[12] C.B. Jones. *Systematic Software Development using VDM*. Prentice Hall International, 1990.
[13] P. Loucopoulos and R.E.M. Champion. Concept acquisition and analysis for requirement specification. *Software Engineering Journal*, 5(2):116–124, March 1990.
[14] M. McCord. Natural language processing in Prolog. In Walker Adrian, editor, *Knowledge Systems and Prolog*, pages 391–402. Addison-Wesley, 1990.
[15] F. Meziane. *From English to Formal Specifications*. PhD thesis, University of Salford, 1994.
[16] F.C.N. Pereira. Extraposition grammars. *American Journal of Computational Linguistics*, 7(4):243–256, 1980.
[17] F. Polack. Integrating formal notations and systems analysis: Using entity relationship diagrams. *Software Engineering Journal*, 7(5):363–371, September 1992.
[18] A. Walshe. NDB: The formal specification and rigorous design of a single-user database system. In Prentice Hall International Series in Computer Science, editor, *Case Studies in Systematic Software Development*, pages 11–45. Jones Cliff B and Shaw R.C., 1990.
[19] D.H.D. Warren and F.C.N. Pereira. An efficient easily adaptable system for interpreting natural language queries. *American Journal of Computational Linguistics*, 8(3-4):110–122, 1982.

APPENDICES
A THE AIRCRAFT PROBLEM

```
An example route is planned for a flight from
blackpool to doncaster.
The route is planned as a number of the discrete
tracks between the intermediate waypoints.
The planned tracks will assure the safe arrival of
the aircraft over doncaster when they are flown in
correct order by the aircraft.(R)
The pilot may have unnecessarily flown through a
storm.(C1)
```

The pilot may have unnecessarily flown through a controlled airspace. (C1)
The aircraft may hit an obstacle. (C2)
The aircraft may hit another aircraft. (C2)
The pilot of a simple aircraft without a sophisticated electronic navigation system would be cleared to undertake a risky flight.
The pilot chooses the visible waypoints from the air. (R)
The pilot draws the tracks of the route on the map.
The pilot steers a heading giving the required tracks along the ground.
The pilot scans the ground for the visible features. (PR)
The pilot verifies the visible features against the map.
The system of a simple aircraft can be considered to comprise the map of the pilot. (C3)
The system of a simple aircraft can be considered to comprise the plan of the pilot. (C3)
The system of a simple aircraft can be considered to comprise a heading indicator. (C3)
The system of a simple aircraft can be considered to comprise the visual sens of the pilot. (C3)
The system of a simple aircraft contrasts with the system of a complex aircraft.
A complex aircraft uses a whole range of the electronic equipment to support the navigation.
A complex aircraft uses a computer-assisted flight planning. (C4)
A complex aircraft uses an inertial navigation system. (C4)
A complex aircraft uses a radar. (C4)
A complex aircraft uses a moving map display. (C4)
A complex aircraft uses a route display. (C4)
A complex aircraft uses a waypoint display. (C4)
A complex aircraft uses an autopilot. (C4)
The pilot chooses the waypoints from blackpool to doncaster in a complex aircraft. (R)
The route is composed of waypoints. (T)
The pilot identifies each waypoint with a number. (C5)
The pilot identifies each waypoint with a grid reference. (C5)
The grid reference contains the latitude. (T)
The grid reference contains the longitude. (T)
The information is used as input to a flight planning software package.
The flight planning software package calculates the route tracks. (C6)
The flight planning software package calculates the distance between waypoints. (C6)
The flight planning software package calculates the heading for the wind conditions. (C6)
The flight planning software package calculates the non-violation of a controlled airspace. (C6)
The derived information may be listed for the pilot to record on the map. (C7)
The derived information may be transferred to a cassette tape. (C7)
The cassette tape is used to load the navigation database on the aircraft.
The autopilot uses the data to fly the aircraft according to the plan of the pilot.

Ci : All the sentences referred by Ci are the result of the split of the same conjunct.
T : The sentence is added to replace a table.
PR : A pronoun reference is encountered and resolved.

B E-R DIAGRAMS FOR THE AIRCRAFT PROBLEM

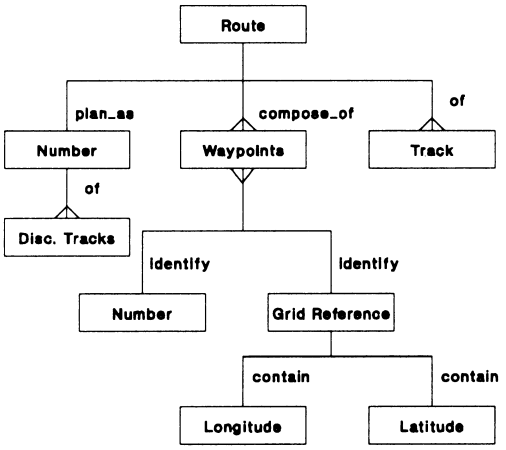

FIGURE 5. The ER diagram of the route planing system.

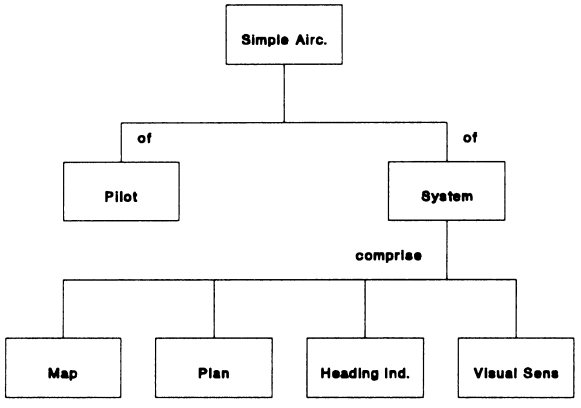

FIGURE 6. The ER diagram of a simple aircraft.

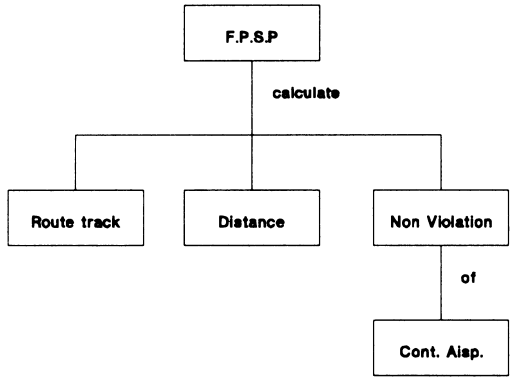

FIGURE 7. The ER diagram of the flight planning package.

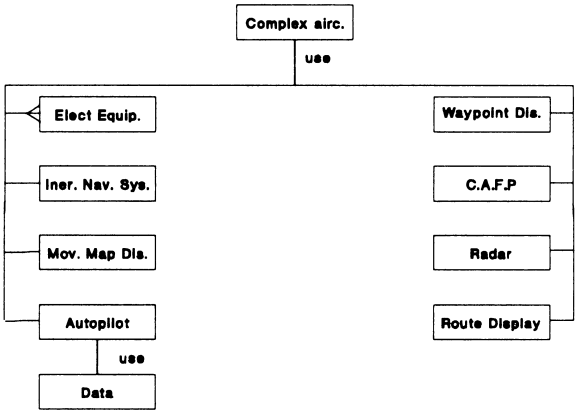

FIGURE 8. The ER diagram of a complex aircraft.

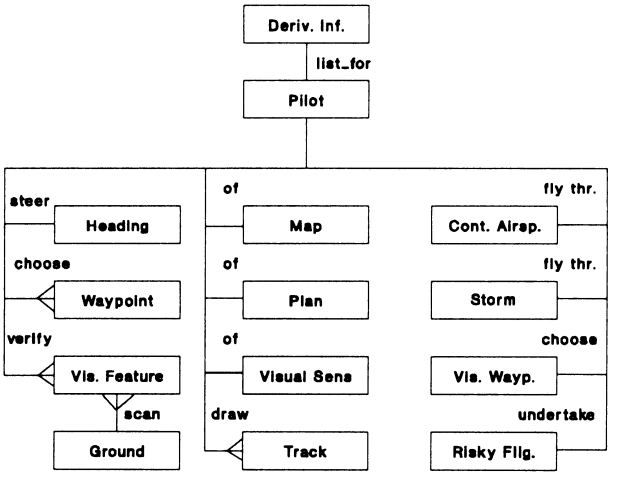

FIGURE 9. The ER diagram of the pilot.

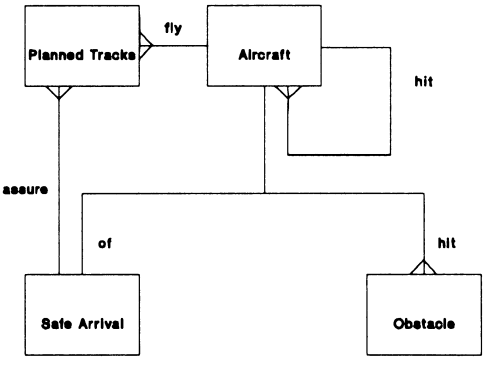

FIGURE 10. The ER diagram of the planned tracks.

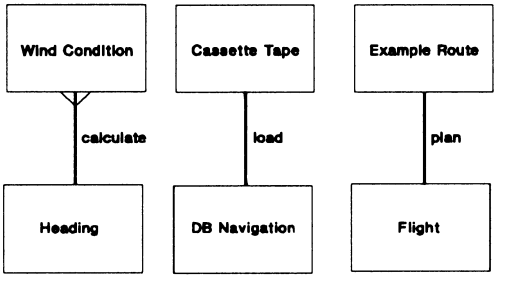

FIGURE 11. The remaining diagrams.

C SPECIFICATION OF THE OPERATIONS

(a) Add a waypoint

ADD-WAYPOINT (*way:Waypoint_t*)
ext wr *route* : *Route_t*
pre true
post $\overleftarrow{route} \frown [way]$

(b) Deleting a waypoint

DELETE-WAYPOINT (*i*: N)
ext wr *route* : *Route_t*
pre i ∈ inds(*route*)
post $\exists route1{:}Route\_t \cdot \exists route2{:}Route\_t \cdot$
$\overleftarrow{route} = route1 \frown [\overleftarrow{route}(i)] \frown route2 \wedge route = route1 \frown route2$

(c) Updating a waypoint

UPDATE-WAYPOINT (*way:Waypoint_t*,*i*: N)
ext wr *route*:*Route_t*
pre $i \in$ inds(*route*)
post $route(i) = way \wedge \forall j \in \text{inds}(route)\ i \neq j \Rightarrow route(j) = \overleftarrow{route}(j)$