INTERACTING WITH AN INTELLIGENT PLANNING SYSTEM USING ENGLISH SENTENCES

B Crabtree[1], R S Crouch[2], D C Moffat[3], N Pirie[1], S G Pulman[4], C D Ritchie[3], A Tate[3]

[1]British Telecom Research Laboratories; [2]University of Cambridge; [3]University of Edinburgh;
[4]SRI International, and University of Cambridge, UK .

AIMS OF PROJECT

An intelligent planning system is an example of a software aid which, although developed by specialists in artificial intelligence, and customised for a particular application by knowledge engineers, is intended eventually to be used by non-programmers for a wide variety of tasks. That is, the domain-specific definition of requirements, resources, etc. for a particular application will be specified by people who are trained in their own trade or profession but who are unlikely to be acquainted with the technicalities of automatic planning systems. There is therefore a need for a communication medium which allows the application specialist, and the non-expert user of the eventual domain-tailored system, to specify their needs without knowing any of the low-level details of the planning notation or the actual operations of the planning system.

This kind of system is one where the 'mice and menus' approach is unlikely to be able to provide a very flexible interface, since it is characteristic of the type of interaction that one would like to be able to have with a planner that the range and type of potential queries is not predictable in advance, and thus not reducible to choices within some predetermined set of options. Furthermore, it is often desirable to have the planner try out a range of hypothetical or counterfactual situations that could not be represented in any obvious graphical form: some kind of language, either artificial or natural, is a necessity here.

The aim of this project is to experiment with the use of English language as the medium of communication. The kind of system we have been trying to build during this three year project is one where the user interacts with the planner to plan some type of external activity, perhaps trying out several alternative scenarios based on differing assumptions, and then, during the course of execution of the resulting plans, engages in further interactions making adjustments when parts of the plan turn out for some unforeseen reason to be unachievable.

The following are examples of the kind of query which are successfully handled in the final version of the system. The sample domain that we have been using is one in which telephone field engineers repair equipment faults. The aim of the planner is to allocate engineers to faults in exchanges throughout the relevant region in the way that maximises the use of their skills, minimises travelling, and achieves highest overall productivity.

1. When will the fault at Ipswich be fixed?
2. Will Brown go to Ipswich from Base before anybody does Job2?
3. What jobs will be worked on while Smith is at Ipswich?

Queries like this require that notions of time, in relation to particular points in the current plan, or to the current state of execution of the plan, should be handled correctly.

4. Could Brown repair the fault at Bury?
5. If Smith goes to Ipswich, could Brown go to Martlesham?
6. If Smith had to do Job2, who could do Job5?

Here the interpretation of the queries depends on the ability to consider alternatives to the current plan which still allow the goals to be achieved. Modal notions like 'could' and 'have to' as well as conditionals, cannot be interpreted without implicit or explicit reference to states of affairs that differ in some respects from that currently being contemplated.

SYSTEM DESCRIPTION

The overall system consists of a Natural Language Front End (NLFE), which produces logical forms (LF) representing the meaning of an input sentence; a Plan Query Language Evaluator (PQLE) which accepts inputs in a Plan Query Language and evaluates them against an internal representation of the current plan, or by manipulating the planner itself to to try produce further plans; and the planner: currently the system will run with Tate's NONLIN planner (8) or IPEM, a system developed at the University of Essex. However, since these large general purpose systems can be rather slow, for development and demonstration purposes we use a hand-crafted, domain specific planner written in Prolog at BTRL. Modules exist for translating in both directions betwen the representations used by these different components so that responses from the planner are eventually presented back to the user by the NLFE.

The NLFE, PQLE and Planner are all separate processes, controlled by a 'debugger' interfaced to the SunView Window system, enabling the course of operation of the system to be traced.

THE NATURAL LANGUAGE FRONT END

The NLFE accepts English sentences and produces logical forms representing the literal meaning of the sentences. First, the sentence is parsed using a unification enriched context free grammar, a lexicon and a bottom up, left to right parser originally developed as part of the Alvey Natural Language Tools project (5). This produces the syntactic structure of the sentence, which is used to determine what the initial logical form of the sentence should be.

Each syntactic rule in the grammar has one or more corresponding semantic rules saying how the meanings of the constituents should be combined to give the meaning of the whole. (When there are several semantic rules, the instantiations of syntactic features determine which one is applicable). To give a simple example, consider the (simplified) grammar rule

```
syntax:  S[stype statement] -->
           NP[number @num, person @per]
           VP[number @num, person @per, vform finite]

semantics: S' = NP'(VP')
```

This says that a sentence can consist of a noun phrase followed by a verb phrase, provided the NP and VP agree in the values given to their number and person features and that the VP is finite. Agreement between features is enforced by unification: feature values beginning with '@' are variables. The semantic part of the rule says that given a sentence built up from an NP and a VP combined in this way, form the meaning of the sentence (S') by applying the meaning of the NP (NP') to that of the VP (VP'). For a very simple sentence we might have:

```
                    S
           [stype statement]
              /          \
             /            \
            NP             VP
   [num sing, per 3rd]  [num sing, per 3rd,
                         vform finite]
            |              |
           John          snores
```

NP' = [λP. P(john)]
VP' = snore
S' = [λP. P(john)] (snore)
= snore(john)

The meanings of the constituents are expressions in a typed higher order logic based on a simplified form of Montague's intensional logic (see (2)). These meanings are ultimately built up from logical expressions assigned as the meanings of individual words by function application or composition. The expressions produced by this stage of processing are not yet complete representations of the sentence meaning, in several respects:

(i) they contain variables over higher order operators which will later be further instantiated: for example, the meaning of some NPs is translated with a veriable which can be instantiated by one of two operators: 'DIST-NP' and 'COLL-NP'. These operators correspond to the distributive and collective readings of NPs found in 'the engineers slept' and 'the engineers conspired', respectively. The operators are abbreviations for more complex logical expressions. DIST-NP, for instance, is a shorthand for quantification over subparts of the (plural) object denoted by an NP.

Thus at the initial logical form stage, the meaning of some NPs is not completely determined, and it is left to a later stage to instantiate their variable with the appropriate operator. The alternative to this method would be to treat each NP as ambiguous between the two readings and select the appropriate one later. The current method is logically equivalent to this but computationally much more efficient, avoiding an uncomfortable proliferation of 'meanings' for an input sentence which differ only in the possibilities for collective and distributive readings of NPs.

(ii) they contain terms which are proxy for the value of anaphoric expressions like pronouns and definite descriptions. In the current system, all anaphoric expressions are analysed as involving the quantifier 'the' (unique existential), differing only in that whereas a definite description like 'the man' will be:

```
the (x) man(x) & .....
```

a pronoun like 'he' will be:

```
the (x) he(x) & .....
```

Thus pronouns are analysed as predicates which give information only about number and gender of the contextually unique referent. In both cases, these subexpressions may be evaluated against a local context and the result of that evaluation passed on to subsequent processing. Currently, only intra-sentential anaphora is implemented in the system, although the addition of the ability to refer to entities mentioned in previous sentences could easily be added (for simple cases, at least).

(iii) they contain free variables or temporal indices, which will undergo a 'linking' process, representing the establishment of contextually determined relations between them. These relationships are established by rules which are sensitive to the type of verb involved in the sentence, whether or not it is temporally modified, and so on. Thus, for example, in a non-temporally modified, present tense stative sentence, the 'current-time' index will be set to the value of the 'speech-time' index.

(iv) they contain markers corresponding to certain types of elided constituent. For example, a sentence like 'Will Joe go to Ipswich before Fred?' is regarded as elliptical for the full form 'Will Joe go to Ipswich before Fred goes to Ipswich?'. Later on, the appropriate semantic content for the missing constituent is worked out on the basis of the logical form for the main clause, and sortal information about the entities corresponding to the arguments of the verb: this enables the differing types of interpretation of sentences like

'Will Joe go to Ipswich before Norwich?'

which is syntactically identical to the earlier example.

(vi) certain subparts of a logical form are identified as corresponding to presuppositions which must be satisfied before a query can be answered appropriately. In the example just given, for example, it is presupposed that it is in fact true that 'Joe will go to Ipswich'. This information can be made use of in providing a helpful response in the case where a query might be answered in a misleading fashion (i.e. 'no') because the presuppositions are not satisfied.

The logical form that results from 'fleshing out' the initial, syntactically generated logical form may contain higher order constructions like predicate modifiers, as well as various tense operators. Such expressions are relatively hard to deal with inferentially, and since the translation from LF to PQL involves inference based on the contents of the LF, these LFs are converted to a more tractable form.

In the case of the tense operators, conversion amounts to replacing the operators by equivalent quantifications over times. In fact we go slightly further than this. Our tense operators are defined using time periods rather than time points. But time periods are more easily reasoned about if we represent them in terms of their start and end points. Thus each operator is converted into a quantification over start and end points of periods. For example `(PRES (faulty ipswich))` is said to be true at the triple of time $<s,t,l>$ if l is a period which has s as an initial period, t is contained in l, and `(faulty ipswich)` is true at time e. (Intuitively, s is the speech time, t is the 'event' time, and l is a localisation time, a time period within which all the events being talked about must occur). Assuming that s has the value 'now' (for speech time), the expression is converted into something that quantifies over times as follows:

```
(some (l)
 (and (time l) (T= (start l) (start now)))
                ;; now starts l
 (some (t) (and (time t)
                 (and (T>= (start t) (start l))
                ;; t contained in l
                          (T>= (end l) (end t))))
            (faulty ipswich t)
```

A sentence like 'Fred works in Ipswich' will contain a verbal predicate modifier, and (ignoring tense) will be expressed as:

```
[(in Ipswich) works] (Fred)
```

where 'in' applies to 'Ipswich' to form a predicate modifier that modifies the predicate 'works'. In this case it will cut down the set of people who work to the set of people who work in Ipswich. The resulting predicate is applied to Fred. In general it is not possible to convert predicate modifications like these into a logically equivalent and more tractable form, as we can with tense operators. We therefore introduce events during the conversion, to give us

```
(some (e) (event e)
        (and (work Fred e) (in e ipswich)))
```

'There is a working event by Fred, and the event is in Ipswich'. It should be pointed out that events are merely introduced as a device for making inference easier.

To illustrate some of these features, here is the initial logical form for the query 'Who does job3?':

```
((S-tense _te1 _tl1)
   ;; tense
 (((WHdet _wh1) person)
   ;; Wh-phrase
   (lambda (_x1) (PRES (do _x1 job3)))))
   ;; Verb phrase
```

When the wh-phrase is expanded out:

```
((S-tense _te1 _tl1)
 (some (_wh1) (person _wh1) (PRES (do _wh1 job3))))
```

In this case there are no pronouns. After the rules have expanded out the tense operator, and linked temporal indices, the final first order logical form is:

```
(some (_tl1)
  (time _tl1)
  (some (_te1)
    (time _te1)
    (some (_wh1)
      (person _wh1 _te1)
      (and (T>= (start _tl1) (start now))
           (T>= (start _te1) (start _tl1))
           (T>= (finish _tl1) (finish _te1)))
     (some (_e1) (event _e1) (do _wh1 job3 _e1 _te1]
```

As will be evident, quite complex temporal relations are expressed via predications involving the start and end points of events, and their relations to the time of utterance of the sentence (i.e. 'now'). Notice also that it is strictly speaking inaccurate to regard this process of fleshing out the compositionally derived logical form as one of macro expansion: actually, we are translating from expressions in one language to expressions of another with a systematic semantic relationship to it (much as skolemisation does in transforming to normal form for theorem proving purposes). For example, the two place logical constant 'do' given as the original translation of English 'do' is eventually mapped into another which is 4 place, relating agents, entities, events and times.

The final translation into PQL is made easier by the fact that we now have a fully first order expression (though this is desirable on other grounds also):

```
exists(type(??wh1, person),
  exists(type(?e1, done(job3, ??wh1, _, _)),
        =<(now, start(?e1)))).
```

Notice that the temporal information supplied by the linguistic analysis is much simplified in PQL: this is a reflection of the more general point, amplified below, that the information derivable from the English input is much richer than can be made use of in the planner.

PLAN QUERY LANGUAGE

Our original intention was to develop a formal language in which to talk about the structures and operations of a planning system in a way which was independent of any particular application, and which did not rely on any idiosyncrasies of a particular planner - a type of SQL for planners. We would like the language to provide in a precise sense a definition of the range of possible interactions it is possible to have with a planning system. PQL would have a clear denotational or operational semantics, which any 'implementation' of it in terms of a particular planner would have to respect. Then the process of moving our system from one planner to another operating of the same factual domain would be *simply that of implementing* PQL *using the basic mechanisms* of the planner in question. (There are of course a different set of problems arising when the domain is different too).

However, the analogy is not one which holds as firmly as we might like. For one thing, database theory and practice is much more advanced in standardisation than is the case with planning. Only if some reasonable convergence on the range of operations and type of structures that planners deal with emerges will it be possible to provide a definition which has a chance of being fairly generally applicable. As it is, there is a wide variety of types of system being used for planning: deductive, procedural, agentless, multi-agent, those that include plan execution and monitoring, or on-the-fly replanning, etc. etc. There seems to be *no great measure of agreement (at any useful level of detail)* about the primitive concepts of planning systems in general, other than those we have already tried to incorporate. Thus any proposed version of PQL is quite likely to be overtaken by events in the world of planning research.

The language that we actually use for communicating with planners and reasoning about plans is, in essence, a sorted first order logic with equality, with the addition of three modal-like operators which can only be used in queries.

To give an example of a traditional type of planning operator formalised in PQL, consider:

```
needs (move(X,Y,Z),
          [block(X), on(X,Y), clear(top(Z))])
makes (move(X,Y,Z),
          [on(X,Z), clear(top(Y)), ~clear(top(Z))])
```

The 'needs' predicate expresses the relation between an action and its preconditions, and 'makes' expresses the relation between the action and any postconditions. The list appearing as a second argument is a shorthand form for a series of individual predications each relating the event to a single precondition. The sortal structure that we assume distinguishes between events, times, propositions, and other individuals, so that 'move' is a function from entities to events, 'on' and 'clear' etc. are functions from entities to things that we might as well call propositions. We also have relations between propositions or events, and times, expressed by 'holdsat', e.g. 'holdsat(t3, on(block3,block4))'.

This language is then suitable for formalising the concepts known and used by the planner, the plans that are actually delivered, as well as domain specific information which may not be needed to form plans, but could be relevant to answering questions.

Queries posed against a PQL 'database' are answered by a procedural interpreter. In the case of the three extra modal operators, the process of answering the query also involves invoking the planner. The modal operators aim to capture notions of necessity, possibility and conditionality. Given a specification of a task we are trying to perform, MUST(P) is true with respect to the task if P is true in all the plans solving the task. MAY(P) is true with respect to the task if P is true in at least one of the plans solving the task. Conditionals of the form 'If P then Q' involve modifying the task specification to include the information contained in the antecedent, P, and evaluating the query Q by generating plans from the new specification.

It is tempting to look at the modalities in terms of possible worlds (4), whereby the sentence MAY(P) is true in the current world if there is some possible world accesible from the current one in which P is true. Here we can regard plans as possible worlds. The current world is the plan currently under consideration, and accessible worlds are those plans that achieve the same goals as the current plan. But this view faces certain difficulties in terms of the procedural implementation of possibility and necessity, different 'grades' of modality that arise, and the treatment of conditionals.

The procedural interpretation of possibilty and necessity as true in some and true in all plans runs into problems with the fact that planners are carefully designed *not* to search a full space of plans. In a route planning domain, we would not want the planner to return a route from London to Cambridge that went via Dumfries. This is a possible route, but it involves going at least 10 times as far as any of the more obvious routes. If you asked a question like 'Can I go from London to Cambridge via Dumfries?' and answered it by looking at the London-Cambridge routes the planner returned, the answer would be therefore be 'No'. But there is a sense in which the answer should be 'Yes', as such a route is possible, if ill-advised. To get a 'Yes' answer, we must contrive to alter the space of plans that the planner searches over. This can be done by making it a goal that the route passes through Dumfries. This motivates the following procedural treatment of possibilities, MAY(P).

> If P is true in the current plan, then TRUE. Otherwise add P to the set of goals, and see if a plan can be generated that meets these goals. If one can, then TRUE, else FALSE.

This treatment of possibility makes it appear as though the task specification is altered rather than held constant. However, it is possible to avoid this consequence if we distinguish between goal conditions that are genuinely part of the task specification, and those that are merely added to alter the planner's search space.

Necessity on the other hand is treated by simply looking at all the plans generated from an unmodified set of goals. This means that in dealing with possibility we are searching over a wider range of plans than we are with necessity. This can lead to the following paradoxical situation. Suppose we ask 'Must Smith do Job 3?', and in all the plans that the planner can currently search over, Smith does indeed do Job 3. Thus the answer is 'Yes'. Now suppose that we ask 'Could Jones do Job 3?'. In answering this, we change the range of plans that the planner will search over, and it possible that Jones does do Job 3 in one of these previously unconsidered plans, in which case the answer would be 'Yes'. On the assumption that only one person can do a job, the two answers are contradictory. The problem could be avoided if we could add negative conditions to the set of goals. In this case, we would treat MUST(P) by adding the negation of P to the set of goals. Assuming that there was at least one plan for the original set of goals, if no plans can be generated once we ensure that P is false, then P must necessarily be true. If a plan can be produced where P is false, P need not be true. This treatment would make necessity the dual of possibility. Unfortunately, there is no general way of imposing negative goals on planners.

In practice, the paradox is not a serious problem. Instances of it do not frequently arise. Also, most users are likely to have a fairly accurate idea of the space of sensible plans, and are unlikely to be posing questions of the type designed to prise out these little inconsistences. The query will not be answered in too misleading a way provided that the idea that these modalities are interpreted with respect to a set of particular desired conditions, and not some more general background, is kept in mind.

Another difficulty for the possible worlds/plans view of modality is the existence of different 'grades' of modal. A question like 'Can Jones repair the fault at the TXE exchange in Cambridge?' can be interpreted in two ways. It could be a question about valid alternatives to the current plan. Or it could be a more general question about Jones's ability as engineer: whether Jones knows how to repair TXE exchanges. The second interpretation of the question makes no appeal to a current plan being the current 'world'. The association of worlds with plans is too rigid.

In general, possibility and necessity are evaluated with respect to a set of background assumptions (3). In many cases the background assumptions will be exactly the specification of the task to be solved. In other cases we might remove some of the goals from the assumptions, leaving perhaps just a specification of engineers' abilities and the locations and types of faults. Alternatively, we might add the description of a plan to the assumptions [1].

In theory the range of background assumptions form a continuous scale, and necessity and possibility can be viewed in terms of entailment by or consistency with the assumptions. In practice it difficult to identify what background is being assumed, and it is helpful to distinguish different types of background. It is also undesirable to appeal directly to logical entailment and/or consistency with respect to a set of assumptions, since planners do not search a logically complete space. PQL categorises three grades of modal: those taken against a task specification, those taken against a particular plan, and those taken against a more general background that assumes no particular task but which do assume a certain amount of knowledge about the domain. The different grades of modal are evaluated in different ways. Task modals are evaluated in the way described above, by using the planner to generate plans. The planner can be seen as a kind of theorem prover, albeit logically incomplete. Plan modals are evaluated by carrying out genuine inference on the description of a plan. This inference is done within the PQLE, and does not invoke the planner. Non-task modals are not implemented yet. There are two ways of dealing with them, depending on whether the planner is used or not. Without using the planner, the PQLE could use inference to determine entailment or consistency by the non-task oriented assumptions. Alternatively the assumptions could be used to construct a new mini-task, and the planner employed to see if such a task can be solved. Thus, for a non-task interpretation

[1] We can also identify two different interpretations for questions of the form 'Could A happen after B?', since typically plans returned by a planner give only a partial specification of the order of events. If the ordering of A and B is left unspecified, the question could be about possible orderings within the current plan, or about completely different plans

of a query like 'Can Jones repair the fault at the TXE exchange in Cambridge?' we set up a task with just one goal (that Jones repair the fault at Cambridge) and initial conditions involving Jones's abilities etc., and see if a plan can be produced.

The third kind of modal operator is the conditional. This too is evaluated relative to a set of background assumptions and is inspired by the Ramsey treatment of conditionals (1,6). The information in the antecedent of the conditional is added to the set of background assumptions, and the consequent evaluated relative to the updated set of assumptions just as if it was a non-conditional query. Again, we have three grades of conditional: plan, task and non-task (with non-task conditionals unimplemented). The modification of the assumptions may be counterfactual, in that the conditional's antecedent contains information that is inconsistent with those assumptions. Normally this means that the background assumptions have to be adjusted, causing 'minimal' change, so that the antecedent can be consistently added. Bringing about such an adjustment is a non trivial matter. However, with the classification of background assumptions that we employ this process can be made easier. The assumptions form a hierarchy, starting with non-task assumptions that simply describe the domain and perhaps the initial state of the world, followed by task assumptions that add a set of goals to the non-task assumptions, and finished by plan assumptions that add a description of a plan to the task assumptions. An antecedent may be counterfactual with respect to a particular plan without being counterfactual to the task specification used to produce it. In such cases, the antecedent is added non-counterfactually to the task assumptions rather than counterfactually to the plan assumptions. There are cases where dealing with counterfactual antecedents by ascending a hierarchy will not work, e.g. if something is counterfactual against non-task assumptions. In these cases there are some very simple heuristics for attempting to add the antecedent consistently, and if these fail, the conditional is not evaluated.

The procedural implementation of conditionals taken relative to the task specification is as follows. The antecedent is added to the task specification, a plan is generated, and the consequent is treated as a normal query. If the consequent is non-modal, as in 'If Smith did Job 3, who would do the job in Ipswich?' the consequent, 'who would do the job in Ipswich' is evaluated against the first plan generated. This contrasts with conditionals with modal consequents, like 'If Smith did Job 3, who could do the job in Ipswich?'. Here we have to generate a range of plans to determine all the people who could do the job rather than the person who would in fact be assigned to do it were Smith to do Job 3. This has a passing, though misleading, resemblance to the Stalnaker (7) treatment of conditionals. On this account, conditionals are handled by going to the 'nearest' possible world in which the antecedent is true, and evaluating the consequent from there. Finding the nearest world involves the use of a similarity relation, ordering the worlds. The order in which a planner generates plans from a task specification could be seen as a kind of similarity relation. However, it functions in a different way from the Stalnaker similarity relation. For them, the similarity relation is a way of incorporating counterfactual information. For us, this is already done by other means, and the order of plans is only used for conditionals with non-modal antecedents.

One interesting thing to note about our implementation is the treatment of conditionals of the form 'If P then May(Q)'. First P is added to the task specification. Then to evaluate May(Q), Q is added to the specification, and plans are then generated. *This means that the query becomes equivalent to* 'May(P and Q)' and also 'If Q then May(P)'. Such equivalences do not hold with conditionals like 'If P then MUST(Q)', and they are a bye-product of the incomplete search space of planners. However, these equivalences seem to be harmless, and this suggests that the 'directionality' of conditionals is a matter of implicature rather than of truth conditions. In many cases, the antecedent is taken to precede the consequent (e.g. 'If he comes here, I will leave'), but in other cases, especially in planning domains, there is no such directionality (e.g. 'If Smith does X, Jones will do Y' where Y may be done before X).

It is also important to note that to handle the treatment of modals and conditionals given here, the planner must be able to accept partial plans. This is because the task specification, as well as giving initial and goal conditions, may also specify certain operations that must be used in any resultant plan at a particular place. To allow this, the planner must be able to build up complete plans on the basis of a partial description of a plan. Most planners do allow this.

GENERAL INTERFACING ISSUES

There are some general lessons that we feel can be learned from our experiences with this attempt to provide a natural language interface to a planning system. The first concerns the limitations of the system that we are interfacing to. The general point is that a system which was not designed with a view to having access via natural language is unlikely to include facilities which such an interface would find useful. In this respect planners, and other types of problem solving systems, are different from databases, which are the most usual system to provide NL interfaces to. Whereas with database systems the range of possible (sensible) queries is more or less limited to the contents, and to some degree, the organisation of the database, with a planner there are far more things that could sensibly and usefully be required.

For some of these type of interactions, there may be little or nothing inside the actual planner for the front end system to use. In presenting our earlier example of successive translations from logical form, we mentioned that much temporal detail is stripped out of the the sentence meaning when it is translated into PQL. This is because the temporal structure in a planner is invariably much simpler than that presupposed by the tense and aspect system of a language.

In fact, NONLIN in our present system provides an example of this: there is actually *no* explicit temporal information at all in a NONLIN plan. The partial ordering of nodes in a plan places some constraints on how the plan could be temporally executed, but all of this is neutral as to particular approaches to the representation of time. Nevertheless, we still want to be able to ask and answer questions about temporal ordering of jobs to be done, and so on. In our current system, this has either to be inferred from domain information, or (as is actually done) treated in an oversimplified manner by associating nodes in a plan with temporally located events. Whereas in this application this is fairly satisfactory, it is not difficult to imagine other applications in which consumption of time resources were important, where this kind of temporal information would have to be superimposed on the actual planner in a way fairly unconnected with how the plans were actually arrived at.

If the designers knew that the planner was going to be used with an NL interface, then it might well be the case that the system could be built in such a way that the relevant information could be more easily superimposed on the basic planning system.

Many interactions do not require invoking the application so much as the domain model, or a declarative representation of the current state of the system. In some more extreme cases, it might be information straightforwardly about the nature of the domain which is needed to answer the question, information that is totally irrelevant for the operation of the planner itself and thus not represented in it. In our current system we would have to represent this knowledge inside PQL, for the not very good reason that there is nowhere else for it to go. But enriching PQL so as to encompass the ability to answer such questions would not, we feel, be a good idea: a general purpose knowledge representation formalism should not be confused with a planning-specific language. In retrospect, we should have devoted much more attention to modelling the domain of application: we tried to avoid this, motivated partly by the practical necessities of building a system within limited resources in terms of time and effort, and partly by the feeling that it was theoretically undesirable to have to emulate some of the knowledge and output of the planner in another part of the system. However, this intuition was, we now feel, wrong: it is better to bring as much as possible of the back-end application, whether a planner, expert system, or whatever, within the purview of the front end processing system. To give a concrete example, much of the information that is necessary for the disambiguation of the Natural Language input is also necessary for an informed translation into the language of the planner, and can in many cases be used to answer questions without consulting the planner or the plan at all.

We should also have paid more attention to issues arising out of interfacing an NL front end to a back end (in this case the PQLE) with its own query language. We had assumed that we could simply translate from logical forms to expressions in PQL, so that a question would produce a single LF query which would get translated into a single PQL query. We now suspect that this is the wrong approach. A particular problem we encountered in the translation was simplifying the complex temporal information in LFs to a less complex and non-redundant form in PQL. The complexity in the LFs arise as a result of giving a compositional semantics for tense that covers a wide range of cases, though in any one instance much of the information is will in fact be redundant. The translation thus involved an element of generating logically equivalent but non redundant expressions. Such expressions are not easy to simplify or optimise, and in the general case there may not even be a decision procedure for doing this. It would be much easier if the inference required for this simplification were done in the question answering process rather than in the question forming process. This suggests a different way of using PQL and the PQLE. PQL could be used to provide a language in which to express the semantic clauses that provide an interpretation for LF expressions, much as the language of set theory is normally used for this purpose. Thus instead of producing a single PQL expression for a single LF query, a procedure generates smaller PQL expressions in an attempt to interpret the LF. Many problems need to be sorted out with this approach, and it may turn out to be as difficult to implement as direct translation, but it is an approach we would like to study.

ACKNOWLEDGEMENTS

This project is supported by the Alvey Directorate, British Telecom, and the Science and Engineering Research Council (Alvey IKBS 179, SERC GR/D/83507).

REFERENCES

1. van Benthem, J. (1985) 'A Manual of Intensional Logic', CSLI Lecture Notes No 1.
2. Dowty, D., Wall, R., and Peters, S. (1981) 'An Introduction to Montague Semantics', Dordrecht: D. Reidel Publishing.
3. Kratzer, A. (1977) 'What 'Must' and 'Can' Must and Can Mean', *Linguistics and Philosophy*, **1**, 337–355
4. Kripke, S. (1963) 'Semantical Analysis of Modal Logics', I, *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik*, *9*, 67-96.
5. Phillips, J. D. (1986) A Simple, Efficient Parser for Phrase-Structure Grammars. AISB Quarterly 59, pp.14-18.
6. Ramsey, F. P. (1950) 'General Propositions and Causality', in *Foundations of Mathematics and other Logical Essays*, New York.
7. Stalnaker, R. 1968 'A Theory of Conditionals', in N. Rescher, ed. *Studies in Logical Theory*, Oxford: Basil Blackwell.
8. Tate, A. (1977) 'Generating Project Networks', IJCAI-77, Cambridge, Ma. USA.