Declarative Interaction through Interactive Planners

Conn Copas and Ernest Edmonds

Human-Systems Integration Group, Information Technology Division,
Defence Science & Technology Organisation,
P.O. Box 1500, Salisbury, SA 5108, Australia
Phone: +61-(0)-8-25-95349 - Fax: +61-(0)-8-25-95980 -
E-mail: cvc@itd.dsto.gov.au
Loughborough University of Technology, Computer Human Interaction Research Centre (LUTCHI), Leics, LE11 3TU, United Kingdom
Phone: +44-(0)509-22-2691 – Fax: +44-(0)509-61-0815
E-mail: E.A.Edmonds@lut.ac.uk

Abstract

Recent progress in planning has enabled this technique to be applied to some significant real-world problems, including the construction of intelligent user interfaces. Previous research in interactive planners has emphasised their dynamism and maintenance advantages. This paper adopts a user-interaction perspective, and explores the theme that a paradigm shift in human-computer interaction is now a prospect: away from the requirement to instruct machines towards a more declarative, goal-based form of interaction. This initiative necessarily involves consideration of the design of goal description languages, and some alternatives are analysed. Some implementation issues involved with embedding planners within a user interface management system are examined. The general planning strategy of constructing executable models of causality within some domain is discussed in the context of human-computer interaction specification methods. Some advantages of planners in contrast to process algebras are described, and it is also shown how Petri nets could usefully incorporate some initiatives from planning research.

Keywords

Intelligent user interfaces, model-based systems, user interface management systems, formal specifications, executable specifications, task analysis, planning, geographic information systems, Petri nets, declarative interaction, goal description languages.

Introduction

Planning techniques have long been considered to hold potential for injecting intelligence into interactive systems. The general principle is that interactive planners are the recipients of goals which describe some desired state(s) of a computer-based system. These planners possess knowledge about various actions (typically corresponding to user-level commands), including in particular the preconditions and effects of these actions. The planning task is to search (nondeterministically)

for command combinations which will achieve the goal. At that point, the planner may either recommend a course of action to the user, or automatically execute the script which has been generated.

Planners have historically been hampered by problems of poor expressiveness, poor performance and, to a lesser extent, ambiguous completeness, but recent research progress suggests their potential may be closer to realisation.

For example, expressiveness has improved with the advent of algorithms for accommodating conditional action effects [Pednault88], disjunctive preconditions, and quantification over dynamic object universes [Weld94]. Performance has simultaneously improved to the point where quasi real-time, interactive planners are being reported in domains such as network searching within the Unix operating system [Etzioni94a] and image processing [Chien94].

One of the aims of this paper is to report on the feasibility of employing interactive planners within another domain: that of user interaction with a *geographic information system* (GIS). These systems, along with many other so-called high-functionality systems [Fischer91] have a poor reputation for usability. As discussed in section 1, conventional engineering solutions to this problem, such as the construction of graphical user interfaces, suffer from inherent limitations which planners may overcome.

More significantly, the little existing work on interactive planners has tended to emphasise the maintenance and dynamism advantages which these possess in comparison to systems which operate in a more procedural fashion, and has only addressed end-user concerns indirectly. A further aim of this paper is thus to investigate in section 2 some HCI issues, with particular reference to the design of goal description languages.

Planners have typically been built by *Artificial Intelligence* (AI) workers for the purpose of implementing some problem-solving system. However, the underlying knowledge representation (including operators, preconditions and effects) is itself of HCI relevance, given the interest in appropriate specification techniques within fields such as UIMSs, CSCW and TA.

Planners necessarily involve an executable model of causality within some domain, which aligns them with model-based approaches to software development in general, and which gives them a close correspondence in particular with techniques which specify the semantics of state transitions, such as high-level *Petri nets* (PNs).

A subsidiary aim of this paper is to compare and contrast developments in planning with executable specification practices in HCI, in section 3. It is contended that planning offers a number of features which could profitably be incorporated, including a more expressive formalism in many cases, and the possibility of more dynamic run-time control.

1 GIS User Interfaces

Consider the following simple visualisation task facing some GIS users, which will be used for illustration throughout the remainder of this paper. The system includes a number of data themes, representing roads, elevation, population, etc, with the display currently being blank. The users' desire could be paraphrased as follows: "I would like to see the roads map in plan view, superimposed upon a white background, containing a legend in the bottom right corner and a scale-bar in the top centre". The expected output of the system is depicted in figure 1.

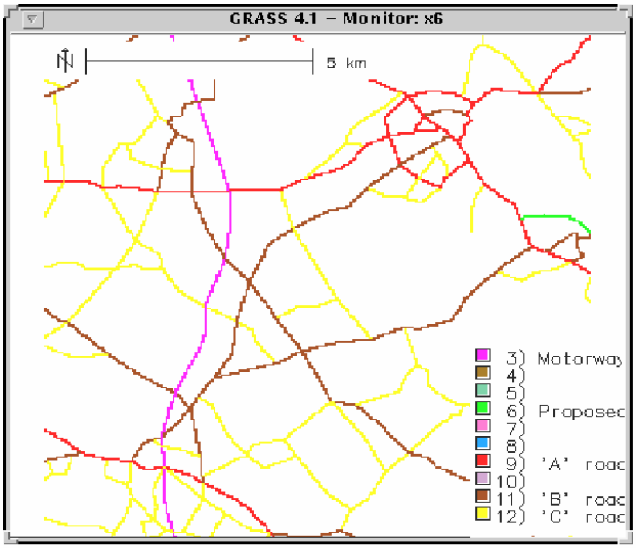

Figure 1. the output of a GIS visualisation goal

It may be objected that this task is undemanding, as it does not involve any particular sophistication in spatial analysis on the part of the user. However, it is a good example for precisely that reason, because even users who have a clear idea of their goals must still translate those goals into a sequence of GIS instructions which is both syntactically correct and semantically coherent. Employing the command-driven interface of the public-domain GIS Grass4.1 [CERL93], seven instructions are necessary for achieving the goal, as depicted in figure 2.

```
d.mon start=x0
d.erase color=white
d.rast -o map=roads
d.scale at=0,0
d.frame frame=frame0 at=0,40,75,100
d.erase color=black
d.legend map=roads
```

Figure 2. A typical GIS command sequence, or plan

As may be inferred from figure 2, GIS tend to possess a large, relatively primitive command-set out of a concern for general-purpose capability and thus resemble the Unix operating system, or a (spatial) statistics package. A typical response to this usability problem is the construction of menu-driven, graphical interfaces; an example of which is shown in figure 3.

These have the obvious advantage of eliminating errors of command retrieval and construction, but are not themselves beyond criticism. One feature of menus is that, linguistically, the items are usually imperatives and, in the simplest case, correspond to application commands. Thus, the influence of the command-line lingers. A further design innovation is to supply some iconic representation of the objects which comprise the system's universe of discourse, thus allowing users to manipulate these in a pseudo-direct fashion.

Within the GIS sphere, however, direct manipulation is rare; for example, only experimental systems allow one to perform map overlays by dragging icons into some viewing area [Egenhofer93]. Part of the problem is that it is difficult to represent all of an object's methods (particularly abstract methods) in a gestural or pictorial fashion. More commonly, although there may be some iconic representation of objects, their methods are invoked by selection from some pop-up or pull-down menu. It could thus be argued that the imperative languages in which most systems are programmed eventually permeate through to the user interface, despite the best efforts of designers to construct various facades.

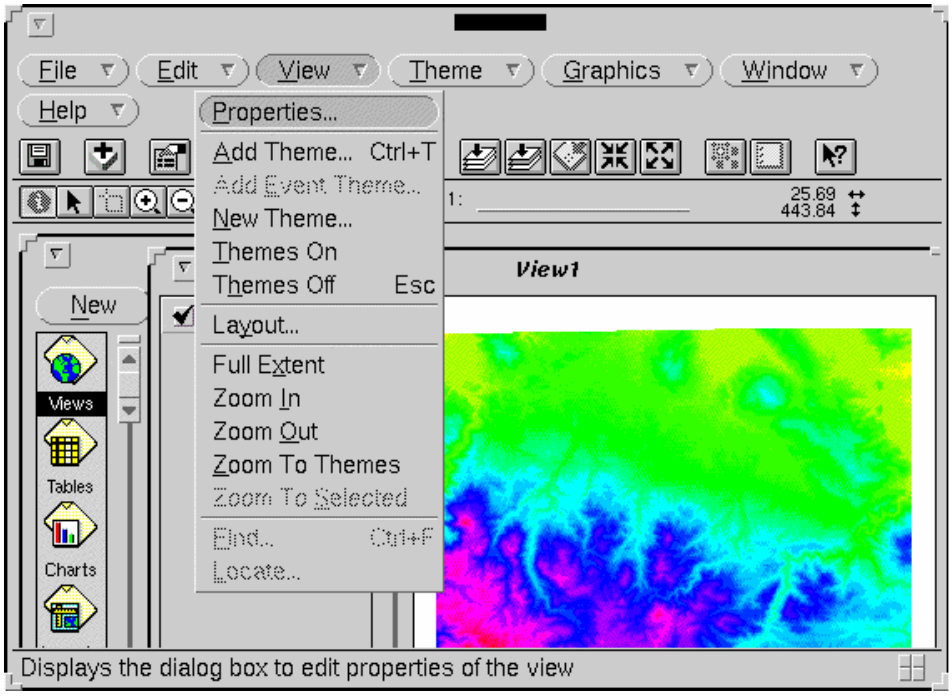

Figure 3. A menu-driven, graphical GIS user interface

It is at this point that planners offer a design alternative. Interactive planners, apart from being 'intelligent', are distinctive because the user inputs goals rather than procedures. That is, the interaction is declarative; a feature of planning's basis in logic and nondeterministic search.

Thus, a prospect which has been tantalising for some time is closer to realisation: users, instead of issuing numerous instructions in order to achieve their goals, may instead interact with machines in the converse fashion, by describing their goals and relying on the machine to infer the necessary instructions. It is slightly ironic that, if planning technology becomes sufficiently well-understood to be appropriated by the mainstream (in the manner of the relational calculus, for example), then these systems are less likely to be deemed intelligent and may come to be regarded as routine constraint satisfiers!

This concept of the utility of declarative interaction rests upon the assumption that it is easier or at least preferable for users to describe goals rather than generate sets of instructions. It is recognised that planners could be said to foster an interaction style of **indirect** manipulation, because such support systems intervene between the user and the (representation of the) domain objects.

One may anticipate that planners may be perceived as introducing superfluous overheads when supporting the kind of simple and self-evident tasks which currently admit well to direct manipulation or, for that matter, to imperative interaction in general. More specifically, it may be hypothesised that the acceptability of interactive planners may be expected to increase as the unit tasks in any domain involve longer sequences of instructions for their completion. The most practical scenario is one in which a variety of forms of interaction are available to the user.

2 An Interactive Planner for GIS

The work reported here employs the public-domain planner Ucpop4.0 [Weld94], written in Common Lisp. Planners may be distinguished by various features, which merit description at this point. The essential features of Ucpop are that it:

1. Is regressive, i.e. search proceeds by selecting operators which can achieve the goal state, then placing the preconditions of these operators onto an agenda of revised goals, until the current state is reached. This strategy is more focused than progressive search methods, and thus has performance advantages in domains where there are a large number of operators compared to the average number of goals involved in any plan.
2. Builds plans from first-principles, as opposed to the strategy of composing a larger plan from some pre-existing library of plan fragments. This latter approach effectively enables learning or experience to enhance performance, but is often described as hierarchical or abstract planning instead. Planners which cannot work from first principles may suffer from inflexibility due to the as-

sumption that one may anticipate users' goals and store a compiled response [Tenenberg91].

3. Is partially ordered or nonlinear, i.e. if alternate action sequences can achieve the same goal state, then the algorithm avoids committing to any one sequence unnecessarily, with consequent gains in performance, end-user support, and flexibility of execution (i.e., it is possible to infer opportunities for parallel execution).

4. Is domain-independent, i.e., the various choices which arise during planning are made without recourse to any domain-specific heuristics, such as "always draw maps before displaying legends". The employment of this general search control strategy preserves completeness, at the cost of some performance. A programmer's interface allows the incorporation of more specific heuristics, which effectively imparts some of the character of an expert system to the planner.

5. Assumes that the planner has access to all necessary information about the state of the world, and that action effects are both instant and deterministic. These restrictions may be regarded as unreasonable within certain real-world domains (which has led to a concern for planners based upon fuzzy or modal logics), but are more reasonable in the case of some artificial software worlds.

The visualisation goal described in Section 1 is represented using existentially-quantified, first-order predicates and Ucpop4.0 syntax in figure 4 (universally-quantified goals and negation are also supported).

This example was chosen partly because of the comparative length of the plan which is required to satisfy the goal. In a previous imperative interface, this goal was identified as a unit task requiring the most involved macro. An example of a relatively complex operator representation is shown in figure 5.

The entities in this domain are both persistent, e.g. data files, and more ephemeral, e.g. the contents of graphics windows. The main features of this example are, first, conditional effects (e.g., the effects of the command are different depending whether the window contains any frames) and, secondly, universal quantification over a dynamic object universe (e.g., the above command has the effect of destroying all existing contents of the window, without having to nominate those contents explicitly).

Assuming that it is desired for the planner to mediate the user-application interaction in a UIMS fashion, two interfaces require attention. The first is between the planner and the application. It is routine to transform the output of the planner into a series of application callbacks, but deeper discussion is deferred until Section 2.1.2. The main interface concern at this point is with the user.

Clearly, after criticising contemporary GIS user interfaces, it would be inconsistent to claim that the predicate logic interface of figure 4 represents an advance in usability! In its raw form, this interface poses a number of hurdles for casual users:

1. mastery of Lisp/Ucpop syntax;
2. mastery of the semantics of predicate calculus, including conjunction, negation, and existential/universal quantification;
3. lack of guidance about the types of goal statements which are possible.

```
:goal   (exists (window ?window)
          (exists (frame ?frame)
          (exists (scale-bar ?scale-bar)
          (exists (map ?map)
          (exists (legend ?legend)
          (and
                  (background-colour window ?window white)
                  (displayed-in window ?window map ?map)
                  (kind map ?map two-d)
                  (refers-to map ?map data roads)
                  (contains window ?window frame ?frame)
                  (position frame ?frame "0 40 75 100")
                  (displayed-in frame ?frame legend ?legend)
                  (refers-to legend ?legend data roads)
                  (displayed-in window ?window scale-bar ?scale-bar)
                  (position scale-bar ?scale-bar "0 0") ))))))
```

"I would like to see the roads map in plan view, superimposed upon a white background, containing a legend in the bottom right corner and a scale-bar in the top centre"

Figure 4: A GIS goal, expressed in terms of both first-order predicate logic and natural language (variables are prefixed with '?')

It may be recognised that these types of problems are also familiar from the database world, which has the advantage of providing conceptual leverage.

For example, it allows one to compare and contrast goal description languages (and techniques) with more familiar database query strategies, despite the fact that plan synthesis is not generally regarded as an information retrieval task.

The predicate logic interface of figure 4 may be seen as an analogue of SQL: declarative (in comparison to its predecessors), demanding (for inexperienced users), and also limited by its first-order formalism (i.e., it is not possible to pose a meta-query about which predicates are available).

```
(:operator d-rast
    :parameters (?container ?name ?data ?map)
    :precondition (and (selected ?container ?name) (data ?data) )
    :effect (and
            (displayed-in ?container ?name map ?map)
            (kind map ?map two-d)
            (refers-to map ?map data ?data) )
            (forall (?A ?B)
                    (when (displayed-in ?container ?name ?A ?B)
                    (not (displayed-in ?container ?name ?A ?B))))
            (forall (?frame ?id ?X ?Y)
               (when (and (contains ?container ?name ?frame ?id)
                          (displayed-in ?frame ?id ?X ?Y))
                          (not (displayed-in ?frame ?id ?X ?Y)) ))
            (forall (?colour)
             (when (background-colour ?container ?name ?colour)
             (not (background-colour ?container ?name ?colour))))
            (forall (?frame1 ?id1 ?colour1)
            (when (and (contains ?container ?name ?frame1 ?id1)
                       (background-colour ?frame1 ?id1 ?colour1))
              (not (background-colour ?frame1 ?id1 ?colour1)))) ))
```

"The effect of displaying some raster data is that the currently selected window now has that map present in it. Whenever the window already has contents, this map overwrites both the background colour and the previous contents of the window, including that of any frames contained within the window".

Figure 5: A planning representation of a GIS command, also expressed in terms of natural language (variables are prefixed with '?')

On the other hand, planners and conventional databases do differ quite markedly in that their underlying formalisms emphasise either the dynamic or structural aspects of some domain, respectively. As a result, whilst the behaviour (i.e., the state transitions) of the GIS domain is explicated by the planning model, the universe of discourse is only implicit. This may, however, be explicated using an ERA diagram, as shown in figure 6.

One advantage of the data model of figure 6 is naturally that the ontological structure of the domain is revealed, e.g., it is apparent that some predicates function as **attributes** of entities (position, background-colour) whereas others serve to **relate** two entities (contains, displayed-in, refers-to).

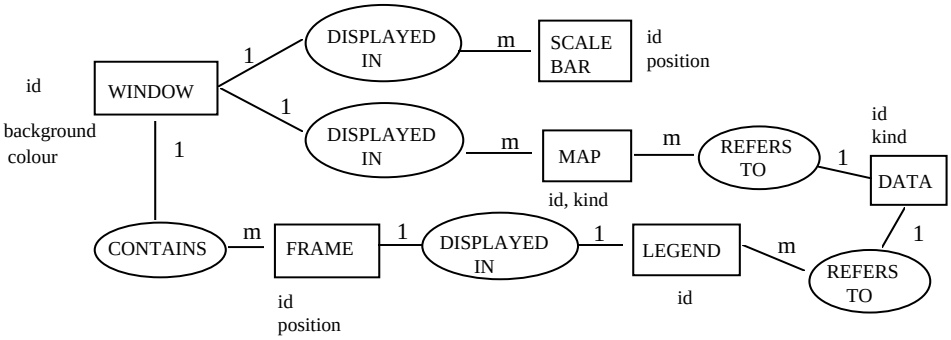

Figure 6. An entity-relationship representation of the universe of discourse underlying a GIS domain

It is also notable that one entity (window) is not present in the natural language goal specification of figure 4, i.e., this entity is consequential upon the goal of displaying maps. Similarly, one relation (refers-to) is effectively implicit in the natural language specification. It would seem important to impress these distinctions upon end-users.

As a preliminary measure, the logic-based interface may usefully be augmented with some standard, higher-order predicates, such as 'entity', 'attribute' and 'relation' (neglecting for the moment esoteric modelling issues such as whether attributes may be considered to be a special entity). This initiative provides the basis for a certain amount of guidance if one then postulates a meta-query facility; however, the problems of mastery of logic remain, and a new problem of meta-query construction arises. Graphical interfaces, alternatively, provide the general features of revealing domain ontologies and reducing problems of syntax in the interaction. An example of this approach for the GIS domain is shown in figure 7.

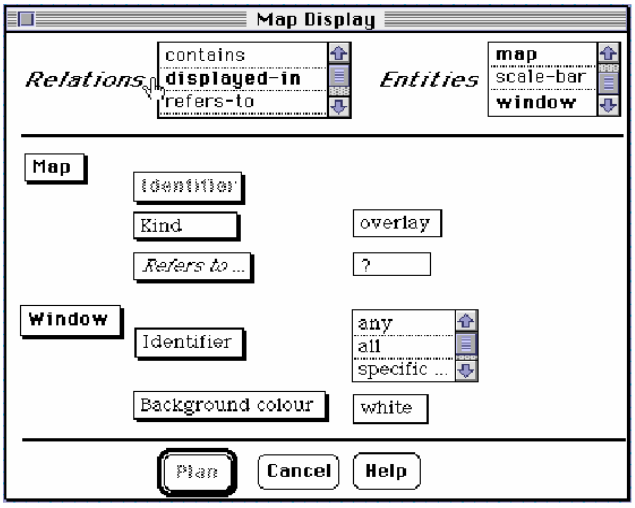

Figure 7. A form-filling interface for specifying goals to an interactive planner

Not unexpectedly, this style of interface resembles a form-filling interface to a relational database. In the spirit of deductive databases, details of whether the plan is being 'retrieved' or 'derived' are suppressed. One design issue which is not immediately apparent from such a static example is that of dialogue control; for example, it is ambiguous whether the dialogue is driven by selection of relations or by selection of entities. A relation-driven dialogue requires that the user selects one or more relations of interest, in order that further entity/attribute fields are subsequently presented for selection or input. Insofar as relations may be interpreted as functions or procedures, this approach violates somewhat the ideals of declarative interaction (see [Etzioni94a] for an example). An entity-driven dialogue possesses the virtue of presenting a more object-oriented view to the user. Currently, both approaches are accommodated.

The user interface is context-sensitive, in more than one respect. First, the data model specifies certain constraints, e.g., that maps but not data can be displayed. This knowledge is used to cause appropriate forms to be displayed, based upon prior selections. Secondly, it is occasionally desirable to impose an order of field filling upon the user, which is achieved using field disabling techniques. For example, it is deemed inappropriate at the interaction point in figure 7 for the user to nominate a map identifier. These dynamics have at present been achieved simply by writing procedural graphics code, without any prior specification. It is recognised that standard UIMS practice is to construct an executable specification of interaction-object behaviour, and it is intended to investigate planning formalisms for this purpose.

Somewhat curiously, one other example of a form-filling interface to a planner [Etzioni94a] appears to be based upon neither an explicit data model nor typed predicates. It is claimed that the form-filling approach overcomes users' discomfort with logic. More precisely, such an approach may be expected to reduce problems of syntax, but the ability of graphics to facilitate a grasp of the semantics of logic is considered in this paper to remain an empirical question. One potentially troublesome feature, and one which distinguishes the above interface from that for a relational database, is the requirement for the user specifically to employ quantified identifiers.

The form-filling interface may be criticised for its linguistic nature, which contrasts with the graphical nature of the ERA diagram on which it is based. One progression is to propose that entities in the goal have an iconic representation. For example, if the user wishes to delete data file "F" or close window "W", then conventional graphical interface techniques allow one to establish a relationship between icons and their referents. The planning situation, however, is complicated by the requirement to accommodate quantifiers (e.g., "I would like to see a map of **some**/**every** data file"). This requires some graphical representation of both anonymous entities and sets; an example of the latter being the palettes employed within interactive drawing packages. A further design issue is the graphical specification of relations and attributes. Conveniently, some of the predicates in the ex-

ample GIS domain (position, contains, displayed-in), by virtue of their spatial connotations, may be readily defined by drawing. For example, a map icon may be dragged inside a window icon in order to convey that the former is 'displayed-in' the latter. Negated predicates, alternatively, are challenging to represent graphically.

It is ironic that, if this notion of graphical goal specification could be carried to its extreme, then the user interface would resemble an advanced direct manipulation interface to a GIS, albeit augmented with quantifiers and negation. Such an interface must depart further from conventional direct manipulation, however, by being insensitive to the sequence of operations. For example, it must be legitimate to drag a scale-bar followed by a map into some viewing area representation (in order that the planner can infer how to display both these entities), whereas in the actual application the scale-bar would become occluded by this sequence. Iconic planner interfaces therefore may gain some design inspiration from direct manipulation, but also must support a form of visual, automatic programming.

2.1 Implementation issues

The work reported in section 2 was intended to demonstrate two concepts:

1. That contemporary planners possess sufficient expressiveness to support significant tasks within a GIS domain;
2. That enhancements may be made to the programmer's interface such that at least satisfactory user interaction becomes feasible.

In itself, this demonstration distinguishes this work from most previous reports of interactive planners, such as [Senay89]. However, a variety of further practical considerations must be addressed before contemplating putting this system into production. Performance is one major concern; the less the system responds in real time, the less its suitability as a UIMS component, and the more its potential status becomes relegated to that of on-line help. Other considerations include the feasibility of interfacing the planner to an application, and the software development effort required.

2.1.1 Performance

The work reported in this paper employs a restricted, although intentionally challenging, sub-set of GIS operators. A complete GIS might involve 300 operators, and so scalability is obviously an issue. Regressive planners scale-up well provided that the application commands tend to have unique effects, suggesting that performance degradation may be as much a function of the compiler as it is of the planning algorithm.

Theoretically, a major influence on planner performance is the average branching factor in a domain [Weld94], which broadly corresponds to the number of alternative actions which must be considered at any choice point. Less formally, an 'ideal' domain is one in which all actions have unique effects, and no action negates any

of the preconditions of other actions. One distinctive feature of this GIS domain appears to be operator complexity, with a rule-of-thumb being that increases in the average number of effects per operator increase the probability of operator interdependencies. Apart from the domain itself, a second influence on performance is the type of queries which are posed of that domain.

For example, quantifiers in the goal statement tend to increase solution times. As a crude generalisation, our experience is that plans of three steps are synthesised in subjective real time on a Unix workstation. The seven step plan of figure 2 is returned in 2-3 s, as something of an extreme example (although some planning failures may take as long to report). In an image processing domain, it has been indicated that plan lengths of 10 steps may be typical, and that reliance on both domain-dependent search heuristics and pre-existing plan libraries is required [Chien-94].

Without resorting to these measures, other options are available for improving performance:

1. The employment of search heuristics which supplement those of Ucpop, but yet which need not be considered domain-specific, e.g., work on the hardest/easiest goals first, avoid considering action sequences which 'undo' each other, use fewest operators, distinguish between 'primary' and 'incidental' effects. These may be regarded as metaplanning heuristics. Provided they weight choices rather than prohibit avenues of search, completeness is retained.

2. It should also be noted that latitude exists for improved planning algorithms; in particular, the possibility of extending the least commitment approach to incorporate typed operators. By reasoning with classes rather than instances of operators, a planner ought to be able to gain performance in the same way that Ucpop does by reasoning with classes rather than instances of the arguments of those operators. Existing work into typed operators has not had direct performance concerns [Anderson88, Kramer94]. It may be shown that typed operators depend upon an object taxonomy [Tenenberg91], also a research frontier for planners, which incidentally reinforces the comments about the desirability of data models which were made in the context of user interface construction.

The usual assumption made in planning is that the shortest plan (found by breadth or best first search) is of most interest. However, if one postulates that the user may wish to inspect a range of alternative plans, possibly with some associated explanation, then both performance and completeness considerations become even more crucial. Considerations of interactivity result in the further design stimulus that there could be advantages in analysing the goal before it is submitted to the planner; in particular, with a view towards estimating solution time. This requires that some comparatively naive heuristics are employed; otherwise, the actual planning process provides the definitive estimate! For example, an analyser may readily ascertain how many different actions will be required for plan solution, and may

then investigate in a preliminary fashion the degree of independence of those actions. It is also possible to envisage a dialogue in which an interactive user is invited to assent to modification of the goal statement, e.g., by the binding of existential quantifiers, or by the deletion of certain predicates.

2.1.2 Application Interface

One standard assumption of many research planners is that exogenous events do not cause the state of the environment to change, i.e. the world is assumed to be closed. In the case of a single-user application, it is also reasonable to assume that the operating system prevents exogenous users from changing the state of the file system, the graphics display, etc. On the other hand, the execution of any plan needs to be followed by a process in which the planner updates its notion of the current application state, as it is unsafe simply to rely on inference for this information.

Therefore, the application must provide commands which return state information, in addition to commands which effect state changes. The planner may then reason about how it can obtain state information, alongside reasoning about how it can achieve target goal states. This requires that planning and execution are interleaved.

In the GIS domain, the implementation of these principles has proved to be problematic, as the application supplies more facilities for altering its state than it supplies for verifying its state. Regarding the planner as a software robot, it could be said that any artificial entity which interacts with the application is hampered by an imbalance between effectors and sensors, reflecting once again a legacy of imperative applications. One solution is to supplement the application with more state-interrogation routines, but at the cost of some extensive low-level programming. It would be ideal if the application could be reprogrammed to signal the planning system after every state change, and this strategy has in fact been adopted for a Unix domain [Etzioni94b].

Without indulging in such modifications, one less than satisfactory approach is to restrict user goals to those which may subsequently be verified by the planner. The discussion so far has assumed that the planner constitutes an intelligent front-end to an imperative application, and thus the perennial UIMS issue arises of how aware the application should be of its user interface. Alternatively, if developing an object-oriented application, then the planner might function as an executable schema.

A further refinement is to address the problems which may occur if the application changes state between the time of planning and the time of execution. In that case, error recovery and replanning are required, generating advanced robotics issues such as how the planning system might become aware of execution errors, and whether it should replan partially or totally.

2.1.3 Software Development Effort

Planners are knowledge-based systems, and so knowledge acquisition is a practical issue which should not be neglected. In contrast to rule-based expert systems construction, however, there are a number of advantages. The latter generally require the encoding of personal experience, which is elusive almost by definition, whereas planners involve the more rationalist enterprise of constructing accurate models of the 'physics' of some domain. The level of abstraction of those models is driven by an analysis of prospective goals. Some application knowledge may be expected to be found in user manuals and documentation, and thus planner development may involve explicating the implicit.

Specialised planner development environments are rare. In the case of Ucpop, code may be written using a Lisp-aware text editor and checked for syntactic and basic semantic conformity. A graphical debugger allows the developer to trace reasons for anomalous or failed plans at run-time. Greater scope certainly exists in the area of static analysis of the knowledge-base, for example, by inferring action categories [Anderson88], or by depicting networks of action dependencies [Murata91].

3 Planning and HCI specification

Planning essentially requires that a knowledge-base containing descriptions of operator (or action) semantics is wedded to a search engine in order to produce problem-solving behaviour. In HCI, action descriptions or representations are also of interest, given the general concern with specifying the dynamics within domains such as UIMS, CSCW and TA. Discussions of HCI specification are typically not wide-ranging, and it is occasionally possible to detect the slightly myopic view that each of these domains has unique representation problems, and thus requires a unique formalism. This is not to deny that research has discovered some useful, specific abstractions (one example being the notion of roles within CSCW), but that the differences between these fields may not be as deep as is sometimes implied.

A second observation which needs to be made at this point is that there is not unqualified enthusiasm for dynamics specification. One long-standing controversy within the UIMS field has been whether the employment of explicit dialogue control models leads to a rigid form of interaction, e.g., [Took90]. Frustration about the lack of user acceptance for systems based upon group work-flow models has existed within the CSCW field almost from its inception, e.g., [Fitzpatrick94]. TA has been suggested to be something of a HCI panacea but, more recently, reservations have arisen about the sophistication of systems derived from temporally-ordered task networks, e.g., [Copas94]. Whilst these problems have their individual features, a common theme also emerges: that specification tends to lead to inflexible systems.

Responses to this problem range from the irrational (that specification should be abandoned in the hope that the implementors of the system will make satisfactory design decisions), to the naive (that problems of inflexibility will be solved by more rigorous analysis), to capitulation (that systems should simply possess modeless dynamics, even if analysis does suggest dependencies between actions). A more satisfactory response is that specifications should express constraints rather than hard-coded action sequences, although it could not be said that there is general appreciation of the implications of this view within the HCI field.

One implication is that specifications are required to be more declarative, i.e. these should state relations which must be preserved.

A second implication is that some constraint solver should be available for generating the dynamics at run-time, as opposed to the strategy of enumerating most of the dynamics at compile-time. The representation employed is obviously a large factor in the success of any constraint solver, and so it is preferable not to consider specification in isolation.

In the UIMS dialogue modelling field, it is commonly accepted that event models are more powerful than context-free grammars and state transition networks [Green86], and this is reflected in the widespread adoption of specifications based upon process algebras. These support run-time constraint satisfaction in the minimal sense that, if one or more actions are specified as alternatives within some sequence, then any dialogue generator would be required to make a choice on some basis.

More sophisticated reasoning, however, would seem to require domain axioms referring to system state. Intuitively, the concept of constraint satisfaction may be seen to be related to the concept of context-sensitive dialogues, a feature potentially supported by rule-based models. It has been shown that a simple rule-based formalism employing propositions (rather than predicates) subsumes the expressiveness of event models [Olsen90]. Rule-based systems, however, have been criticised within AI for various reasons, including their lack of structure, and also because they encourage the encoding of a comparatively shallow association between situations and conclusions. Model-based reasoning is seen as a progression, in which deeper, physical knowledge is employed.

Planners epitomise the model-based reasoning approach because of the causal relationship which is captured between preconditions and effects. This paper also demonstrates that planners epitomise the constraint satisfaction approach to generating system dynamics, as plans are constructed at run-time as a result of symbolic problem-solving. The distinction between planners and some forms of rule-based systems, however, is not as clear as these observations might imply. The operator descriptions contained within planner knowledge bases may be reinterpreted as rules of the general form "if preconditions and action is chosen, then effects". Model-based knowledge may therefore be regarded as a representation discipline which is imposed upon the rule-based tradition. Similarly, model-based reasoning

may be regarded as a development of the reasoning supplied by production system interpreters. In other words, planners may be regarded as specialised inference engines, which accounts for the occasional attribution that regressive planners, for example, employ backward chaining.

Causal action knowledge is also a feature of one influential UIMS, namely UIDE [Sukaviriya93]; however, this employs a different form of inferencing than planners. Such model-based UIMS reason in a projective fashion, i.e. given a sequence of one or more actions, the system computes the next state of the application (in contrast to planners, which find partially-ordered paths between states). Projection algorithms are computationally unremarkable in comparison to planning, as these appear to be deterministic and do not involve backtracking. (It is unclear whether parallel actions are supported, which potentially might require the system to resolve conflicts).

UIDE has been promoted as an automatic dialogue generator, but also subscribes broadly to constraint satisfaction principles; one qualification being that the simulation performed by the constraint solver is probably too routine to be deemed intelligent. Projection does have the advantage of supporting the provision of advice about the consequences of executing nominated command sequences, and it may be anticipated that projection and planning tend to be reciprocal cognitive activities of the user (as illustrated respectively by two prototypical questions: "what if... ?" and "how can I... ?"). Thus, an ideal UIMS would accommodate both forms of reasoning.

Contemporary planners may be further distinguished from UIMS by their expressiveness, with the incorporation of negation, existential and universal quantification, and conditional effects frequently being considered necessary for modelling anything other than toy domains. As indicated previously, one major deficiency of planners is their general disregard of data models, although this paper demonstrates that a hybrid technique is straightforward. Contemporary UIMS take the additional step of employing object-oriented data models, with inheritance naturally increasing the expressiveness of structural aspects of the domain.

Causal knowledge is also an implicit feature of some formalisms which claim no direct heritage in knowledge-based systems. In general, techniques which model the semantics of state transitions, such as high-level PNs, fit into this category. An early comparative review of UIMS formalisms which includes PNs is provided by [Cockton87]. Discussion about PNs is complicated by the facts that, firstly, the technique is highly fluid and thus provides great opportunity for individualistic extensions and, secondly, extant applications of PNs within HCI have tended not to exploit their full power. Because of PN diversity, it may not be particularly meaningful to regard these as a formalism in their own right, but instead as a transition network which is augmented with both input and output information for each transition. (An example of a particular form of PN is shown in figure 8, with more discussion to follow shortly). Some HCI examples of PNs employ deterministic nets (i.e., nets containing no choice points), in which case the expressiveness de-

generates to something approaching a finite state machine. It is also customary for authors to emphasise that PNs explicate parallelism, which provokes the issue of whether the nets are intended to represent transition **possibilities** or, instead, actual sequences of transitions. In the latter case, the net effectively degenerates to a graphical process description, although examples of nets containing explicit parallelism directives are in fact quite rare.

In order to position PNs within the context of this paper, it may be observed that most HCI examples to date, e.g., [Palanque95], employ a form in which actions are effectively associated with both preconditions and effects, expressed as single states. If these states are subjected to a finer grain of analysis and represented as a conjunction of predicates, then a predicate/transition net is obtained, as depicted schematically in figure 8.

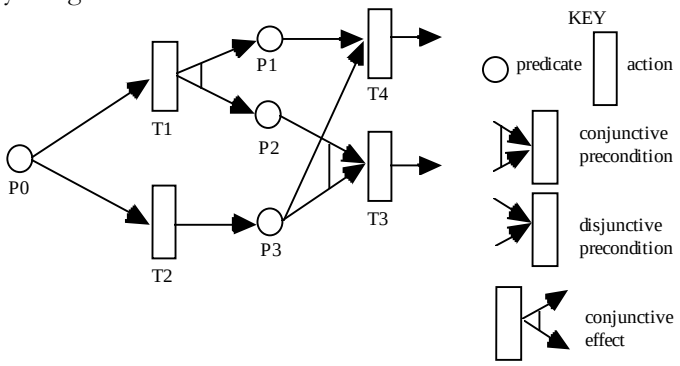

Figure 8. Schematic diagram of a predicate/transition net

The representation of figure 8 is akin to that employed by either planners or existing model-based UIMS, with the main difference being that the unit of representation is not the individual action but instead a network of actions related by their inter-dependencies. PNs may thus be regarded as the visible output of a dependency analysis of some action knowledge-base. This observation provokes the issue of why modellers should be burdened with performing the analysis manually, as is current practice.

Regressive planners, for example, continually search for actions whose effects will satisfy the preconditions of other actions. [Murata91] presents an algorithm for a basic form of PN generation, which effectively involves joining the 'nets' representing individual actions on the basis of common places.

It may be speculated that an ideal system would provide the modeller with graphical editing facilities for the knowledge-base, suggesting that PNs could also mediate user input. Figure 9 summarises this discussion regarding the inter-relationship between existing model-based UIMS, planners, and PNs.

In order to illustrate the commonalities between planners and PNs, it was originally intended to represent the GIS domain of this paper in PN form. However, expres-

siveness problems instantly arose when attempting to represent the semantics of the commands of figure 2.

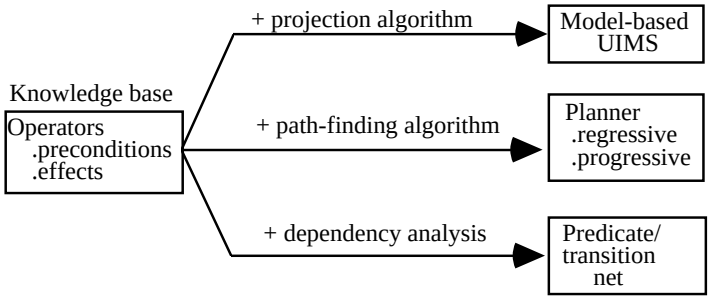

Figure 9. The inter-relationship between existing model-based UIMS, planners, and Petri nets

First, there is the problem of conditional effects (for example, the effects of the 'd.rast' command of figure 5 are different depending upon whether any other maps are already on display). One way of proceeding is to model each condition as a pre-condition of a set of related commands. The Ucpop planning algorithm effectively performs such a command cloning, but that is no justification for engaging in such inelegance at the representation level. In addition, there are problems of both negation and universal quantifiers (for example, one of the effects of the 'd.rast' command of figure 5 is that **all** previous contents of the window are now **not** displayed). It has been proposed that negation might be accommodated within PNs by the use of so-called inhibitor arcs [Anglano94], but we are unaware of techniques for representing universal quantification.

These expressiveness problems may readily be solved by a small number of notational extensions. It would be preferable if these extensions could be introduced in an ontologically unambiguous fashion, which is arguably not currently the case with the language of 'places', 'tokens', etc.. It may also be preferable if any extensions that were introduced were tempered by considerations of executability, as is customary with planning. The commonly-held advantages of specifying independently of implementation are recognised; however, a more integrated approach can have the advantage of providing guidance for a specification process which is under refinement.

For example, PN modellers are at liberty to graft procedural programming constructs and other extra-logical features onto their nets. These extensions increase the versatility of the technique, but potentially at a cost of reduced conceptual coherence; an issue which has received no attention in the HCI literature to date. As a second example, PNs theoretically permit disjunctive (i.e., nondeterministic) effects; a controversial issue within planning. There is general consensus that real-world planners ought to be able to function with incomplete information about the environment; however, there is less consensus about the utility of functioning with an incomplete model of one's capabilities.

Regarding the executability of PNs, 'reachability analysis' is recognised as important, and broadly corresponds to the planning task of finding a sequence of operators which will transform the current state to some target state. At its most simplistic, reachability involves using existing planning techniques, e.g., [Zhang90]. Progressive planning has typically been employed and, as indicated previously, this approach is generally not considered to scale-up well. Isolated instances of regressive planning (known as 'backward reachability' in PN parlance) have been reported [Murata91, Anglano94]. One unique contribution of PN research is the use of matrix equations to generate reachability solutions; a potentially exciting feature given the performance problems which plague heuristic search. Unfortunately, the former technique has a narrow range of application [Murata89], apparently being restricted to deterministic nets.

The discussion so far has had a UIMS dialogue flavour although, as indicated previously, principles of dynamics modelling are of more general relevance. The TA field exhibits less formal diversity, partly because of an entrenched view that TA should involve task decomposition and sequence description, e.g., [Hartson90]. This approach has the unfortunate effect of resulting in a comparatively static task network, which has implications for the sophistication of any user-computer dialogues, advice-giving systems, etc., which might be derived from that network.

This restricted view of what constitutes 'task analysis' also tends to neglect that, firstly, TA could involve knowledge acquisition and, secondly, that high-level cognitive simulations (i.e., those unconcerned with the micro-architecture of cognition) typically involve some task representation which is necessarily executable. If a broader focus is adopted, then many expert systems may also justifiably be regarded as executable TA, typically employing a rule-based model.

Isolated examples of more constraint-oriented approaches to TA exist. One of the original examples of a cognitive simulator, GPS [Newell72], also happens to be one of the original examples of a planner, with a more contemporary incarnation in [Blandford93]. ETKS [Borkoles92] employs a formalism based upon actions, preconditions and effects, but neglects task-plan generation in favour of compile-time specification. [Palanque95] employs what is effectively a predicate/transition net towards TA (although it is unclear whether tasks or devices are actually being modelled). In the last two examples, an object-oriented data model is also employed in order to represent structural aspects of the user's conceptual world.

One research issue associated with using formalisms based upon action semantics within TA is the readiness with which higher-level, conceptual actions may be identified. As possible evidence of difficulty, some models which are said to derive from either a cognitive simulation or task analytic perspective in practice are barely distinguishable from lower-level application models, e.g., [Blandford93, Palanque-95]. On the occasions when this anomaly is acknowledged, the usual justification is that experienced users are expected to possess faithful mental models of cause-and-effect within the application or device with which they are interacting. This lack of discrimination between user and application models is undesirable in those

cases where TA is being used to enhance some application. Referring back to the example goal which has been used throughout this paper, GIS users typically do not wish to display maps, etc., for idle reasons. Instead, they may have higher-level goals, such as planning routes, or deciding upon regional zoning policies. The existing planner cannot support those goals directly because the 'awareness' of the application is limited to files, maps, legends, etc. If it is wished to provide support for higher-level goals like route planning, then the application needs to be augmented so that it, firstly, contains higher-level data types such as routes and, secondly, provides higher-level commands (or methods, in an object-oriented application) such as 'compare routes' which operate on those data types. This approach requires that the user's conceptual world may be modelled independently of the application's world.

Conclusion

This paper has demonstrated that contemporary planners are sufficiently expressive that it is feasible to build intelligent interfaces which support some significant user tasks within a GIS domain. A broad view of these developments suggests that more is involved than just the provision of intelligence: paradigms of user interaction may be enabled to evolve from an imperative towards a more declarative style.

The advent of interactive planners raises design issues of goal description techniques, and some alternatives have been analysed. It was shown that the user interface to planners cannot be constructed in a methodical fashion without access to an explicit data model of the domain; something lacking in existing planners. The performance of contemporary planners has been found to be encouraging for these to mediate the user-application interaction in a UIMS fashion, although further research is required into both performance enhancement and interactive facilities.

The advent of interactive planners raises concerns about an imbalance in conventional application command sets; between commands for effecting state changes, and those for verifying current state. Constraint satisfaction techniques have been proposed as a general approach for solving the problem of inflexible system dynamics, and planners have been shown to support that approach. Planning representations have been analysed in relation to HCI specification practices, with the conclusion that many model-based formalisms could usefully exploit either the expressiveness of planners, or the dynamic run-time control which planning algorithms provide.

Acknowledgements

The authors wish to thank the anonymous reviewers of this paper for their constructive comments.

References

[CERL93] *CERL Grass 4.1*, Reference Manual, US Army Construction Engineering Research Laboratory, 1993.

[Chien94] Chien, S., *Using AI Planning Techniques to Automatically Generate Image Processing Procedures*, in Proceedings of Second International Conference on Artificial Intelligence Planning Systems (Chicago, June 1994), K. Hammond (Ed.), AAAI Press, Menlo Park, 1994, pp. 219-224.

[Egenhofer93] Egenhofer, M.J., Richards, J.R., *Exploratory Access to Geographic Data Based on the Map-Overlay Metaphor*, Journal of Visual Languages and Computing, Vol. 4, 1993, pp. 105-125.

[Etzioni94a] Etzioni, O., Weld, D., *A Softbot-Based Interface to the Internet*, Communications of the ACM, Vol. 37, No. 7, July 1994, pp. 72-76.

[Fischer91] Fischer, G., *The Importance of Models in Making Complex Systems Comprehensible*, in « Mental Models and Human-Computer Interaction 2 », M.J. Tauber, D. Ackermann (Ed.), North-Holland, Oxford, 1991.

[Pednault88] Pednault, E., *Synthesizing Plans that Contain Actions with Context-Dependent Effects*, Computational Intelligence, Vol. 4, No. 4, 1988, pp. 356-372.

[Senay89] Senay, H. et al., *Planning for Automatic Help Generation*, in Proceedings of the 1st IFIP TC 2/WG 2.7 Working Conference on Engineering for Human-Computer Interaction EHCI'89 (Napa Valley, 21-25 August 1989), G. Cockton (Ed.), North-Holland, Amsterdam, 1990, pp. 293-311.

[Tenenberg91] Tenenberg, J.D., *Abstraction in Planning*, in « Reasoning About Plans », J.F. Allen, Kautz, H.A., Pelavin, R.N., Tenenberg, J.D. (Eds.), Morgan Kaufmann, San Mateo, 1991, pp. 213-283.

[Weld94] Weld, D.S., *An Introduction to Least Commitment Planning*, AI Magazine, Vol. 15, No. 4, 1994, pp. 27-61.