Object-centred planning

D.E.Kitchin
scomdk@zeus.hud.ac.uk

T. L. McCluskey
lee@zeus.hud.ac.uk

The School of Computing and Mathematics,
The University of Huddersfield,
Huddersfield, UK

October 1, 1996

Abstract

In this paper we argue generally for a move to an object-centred formulation of precondition planning. Our previous work has demonstrated some of the advantages of systematically creating and compiling an object-centred domain model. Here we describe a complementary object-centred planning engine which exploits such a domain model, and review some of its benefits. We also survey related work that has attempted to exploit the object-oriented approach in planning.

1 Introduction

Much AI planning research has concentrated on the design, analysis and efficiency of planning algorithms, with relatively little attention being paid to the creation and capture of planning domain models. Representational issues can have a big impact on planner performance. For example, it has been shown that different encodings of the same problem domain used with the same planner can give different results [9]. The construction of planning domain models is an important issue in AI planning, especially if we wish to promote the use of planning technologies in more realistic applications. Chien [5] cites "the amount of effort required to construct, debug, verify and maintain the planning knowledge base" as a major factor in deciding whether AI planning techniques are applicable to real-world problems.

Many successful practical planning systems have relied partially for their success on the fact that much of the planner's work is actually done by the domain writer, as Drummond points out in [8]: "the body of an operator contained an entire plan, complete with an existing causal structure defined by the person writing the operators ..... establishment of the preconditions had been dealt with by the person who wrote the operators." OPlan [6] also uses "pre-packaged plan fragments". While these systems are successful in that they have been usefully applied to practical problems, it is difficult to draw useful generalisations from them. Allen et al also make this point in [2] where they state that while planning systems such as SIPE and O-Plan have been successfully used for some practical problems they "become ineffectual for large problems and are logically incomplete and theoretically rather opaque." As Chien points out in [5] we need to enable domain experts to create and validate planning domain models without having to rely on AI planning experts.

Domain modelling is complex and debugging and validation can be difficult. To facilitate the 'shift to industry' we need to address issues of knowledge representation and domain capture and modelling - even in toy worlds a good representation is important, in complex, real-world domains it is even more so. The object-centred method we have developed [17, 16] provides a tool-supported, systematic method for the creation and validation of precise planning models. One of the benefits of a systematic method of domain encoding is that one can have confidence in the model created, and the modeller can use various tools to aid the process and check the encoding. Tool support also assists in model maintenance.

As well as helping in the validation of planning domain models, we believe this method can help bridge the gap between real and theoretical planning, and help in various ways with regard to planner efficiency. In [18] we show how a tool-supported method can be used to create such a domain model, and in [16] we describe the whole method is depth, showing how huge planning speed-up can result from such an approach.

We have already had promising results with object centred encodings and compilation tools using a conventional planner: here we describe the design of a planning algorithm which is consistent with the domain model in that it functions at the object level. In the final section we survey related work that has attempted to exploit the object-oriented approach in planning.

2 An Object-centred Planner

2.1 State Space Definition

First we specify the planning ontology, using a multi-robot extension of the familiar STRIPS world as an example. Let there be[1]:

– a set of sorts *Sort* denoting object classes, e.g. $\{box, door, room, robot, key, colour, arm\}$,

– a set of object identifiers e.g. $\{box1, box2, .., door12, door23, .., robot1, robot2, ..\}$

each of an (obvious) sort, and a set of predicates *Predicates* each with strongly-sorted arguments. Instances of members of *Predicates* that are always true are stated in a set called the *atomic_invariants*. There are 45 members in this set in our current model of the multi-robot world, examples are

$colour\_door(door12, red), part\_of(tom, toms\_arm), connect(room2, room3, door23)$

For all $x \in Predicates$, if x' is an instantiation of x, and $x' \in atomic\_invariants$ then all instances of x not appearing in *atomic_invariants* are assumed necessarily false. Those pred-

[1] a full object-centred specification of this world is available via anonymous ftp at prometheus@hud.ac.uk in file pub/julie/r3_world

icates appearing in *atomic_invariants* are called static, and all other predicates are called dynamic.

Dynamic predicates are split into a set of disjoint subsets, such that each subset is *owned* by a sort. A sort is said to be dynamic if it owns one of the subsets, as these predicates are used for describing the state of objects contained in that sort.

A state of a dynamic object is called a *substate* (or 'sort-abstracted state'). Valid substates are defined in our framework, for each sort, as an instantiation of one of a set of clauses. Each one of these clauses represent a class of substate that the object might be in. For example sort *Box* owns the four dynamic predicates:

$$\{box\_in(Box, Room), box\_next\_box(Box, Box),\\ \quad box\_next\_key(Box, Key), box\_near\_door(Box, Door, Room)\}$$

and the substates of an object of sort *Box* are defined by the set of clauses:

$$\begin{array}{l}\{\{box\_in(Box, Room), box\_near\_door(Box, Door, Room), connect(Room, Room1, Door)\},\\ \{box\_in(Box, Room), box\_next\_box(Box, Box1), Box \neq Box1)\},\\ \{box\_in(Box, Room), box\_next\_key(Box, Key)\},\\ \{box\_in(Box, Room)\}\}\end{array}$$

A clause is written using a subset of the predicates owned by the sort, together with any necessary static predicates (in the example above these include *connect* and $\neq$). We have used a kind of "closed world" shorthand for defining clauses with the following semantics: those predicates that appear are assumed true in the substate class, and all other predicates owned by the sort are assumed false. All variables (shown in capitals) range over their sorts. In our multi-robot world the following substate satisfies the first substate class above, and so is a valid substate for object $box1$:

$$\{box\_in(box1, room2), box\_near\_door(box1, door23, room2)\}$$

Let *Substates* be the set of all substates of any sort. The state space of a planner is defined in two stages. First, we specify the universal state space as the set of all mappings between dynamic object identifiers and substates[2]:

$USS = Dynamic\_Ob\_ids \rightarrow Substates$

Part of a state might be:

$$\begin{array}{l}\{box1 \mapsto \{box\_in(box1, room2), box\_near\_door(box1, door23, room2)\},\\ door23 \mapsto \{closed(door23), locked(door23)\},\\ robot2 \mapsto \{robot\_in(robot2, room3), robot\_next\_key(robot2, key3)\}\\ \ldots\}\end{array}$$

[2]in deterministic planning we assume that in any world state the substate of each object is unique

Note that USS is total and well defined, and for any $u \in Dynamic\_Ob\_ids$, $USS(u)$ must be an instantiation of one of the clauses belonging to u's sort, such that any static predicates in the substate appear in the $atomic\_invariants$.

Next we restrict the USS by specifying relational invariants between substates within a full state. An example of a state invariant is that two robot arms (A_1 and A_2) cannot be holding the same key (K). In our formulation, this is expressed as an inconsistency constraint:

$inconsistent\_constraint(arm\_used(A_1, K)\&arm\_used(A_2, K)\&A_1 \neq A_2)$

The definition of the *valid* state space is then the space of mappings VSS, where

$VSS = Dynamic\_Ob\_ids \rightarrow Substates$

and VSS is a subspace of USS, and the range of any map in VSS does not satisfy any of the inconsistency constraints.

To summarise, in the classical formulation, a 'state' is usually defined as a conjunction of predicates under the closed world assumption. In our object-centred formulation, a state is a conjunction of object substates satisfying state invariants.

2.2 Goal Conditions

In the classical formulation, goal conjunctions are often expressed as sets of literals. Here we interpret a goal expression as a map between objects and **sets of** substates, explicitly showing the conjunctive and disjunctive nature of goals.

For example, consider the goal literal set:

$$\begin{aligned}&\{robot\_next\_door(robot3, door45), box\_next\_box(box1, box2),\\ &\quad \neg\ open(door12), in\_room(robot3, room3)\}\end{aligned}$$

This involves 3 key dynamic objects $robot3, box1, door12$ ($box2$ is not a key object as in our domain formulation box_next_box describes the substate of the object in its first slot) and hence is translated into a goal consisting of a conjunction of three objects' substate expressions:

$$\begin{aligned}&\{robot3 \mapsto \{\{in\_room(robot3, room3), robot\_next\_door(robot3, door45)\}\}\\ &door12 \mapsto \{\{closed(door12), unlocked(door12)\}, \{closed(door12), locked(door12)\}\}\\ &box1 \mapsto \{\{box\_in(box1, room1), box\_next\_box(box1, box2)\},\\ &\qquad\qquad \{box\_in(box1, room2), box\_next\_box(box1, box2)\} \ldots\}\}\end{aligned}$$

Hence we define the space of acceptable goals as GC:

$GC = Dynamic\_Ob\_ids \rightarrow Substates\text{-}set$

where each member of $Substates\text{-}set$ is a set of substates of one sort. Note that for any goal G, $dom(G) \subset dom(I)$, as goals can only be posed for known objects. The negated goal above evaluates to a set of goal substates that do not satisfy the expression after the negation. This formulation makes implicit disjunction explicit, as a single literal goal may correspond to a set of object states (as in $box1$'s goal state). Finally, both existential and universal goals can be given a well-founded semantics this way, using this explicit representation of dynamic objects' substates.

2.3 Operators

Operators change the state of dynamic objects. We define an operator as a structure:

$op(I, P, N, C)$

where:

- I is the operator's identifier
- P is a set of *prevail conditions*, such that each $P_v \in P$ contains dynamic predicates owned by one sort s, and, optionally, static predicate(s). When instantiated P_v will implicitly define one or more object substates of sort s.

  Example: in a *pushthrudoor* operator, the two prevail conditions would be

  $\{arm\_empty(Robot, Arm), part\_of(Robot, Arm)\}$
  $\{open(Door), unlocked(Door)\}$

  The first condition is included as the robot cannot hold anything while pushing.
- N is the set of necessary changes, with members of the form P_n *CHANGES TO* E_n, where P_n is a substate condition of sort s as defined for P above, and E_n represents the effect of the change in terms of the new object state. E_n is thus one of the substate classes of sort s. For the operator to execute, all the conditions P_n must be met in the current state. An operator is thus potentially applicable to objects in a variety of substates, but when applied (i.e. the P_n slots are matched with an element of *VSS*) it must result in an instantiation of each E_n giving a unique output substate for each object affected.

  The necessary effects for the pushthrudoor operator are:

  $$\begin{array}{ll} \{\{box\_near\_door(Box_1, Door_1, Room_1), box\_in(Box_1, Room_1), & \\ connect(Room_1, Room_2, Door_2)\}, & \\ \quad CHANGES\ TO & \{box\_in(Box_1, Room_2)\}, \\ \{robot\_in(Robot_1, Room_1), robot\_next\_box(Robot_1, Box_1)\} & \\ \quad CHANGES\ TO & \{robot\_in(Robot_1, Room_2), robot\_next\_box(Robot_1, Box_1)\}\} \end{array}$$

- C is the set of conditional changes, with members of the form P_c *CHANGES TO* E_c, where P_c is a substate condition of sort s as defined above, and E_c is one of the substate classes of sort s. Any of the conditions that are satisfied result in an object changing state as specified by the second component of the pair. In this case, if a P_c has an uninstantiated variable v after the necessary effects have been carried out (v is a so-called free variable), then the conditional effect must be carried out for each object of v's sort.

The conditional effects for the pushthrudoor operator are:

$$
\begin{aligned}
&\{\{key\_next\_box(Key_1, Box_1), key\_in(Key_1, Room_1)\} \\
&\qquad CHANGES\ TO \qquad \{key\_in(Key_1, Room_1)\}, \\
&\{box\_next\_box(Box_2, Box_1), box\_in(Box_2, Room_1)\} \\
&\qquad CHANGES\ TO \qquad \{box\_in(Box_2, Room_1)\}, \\
&\{robot\_in(Robot_2, Room_1), robot\_next\_box(Robot_2, Box_1)\} \\
&\qquad CHANGES\ TO \qquad \{robot\_in(Robot_2, Room_1)\}\}
\end{aligned}
$$

The first conditional effect can thus be paraphrased "if any keys were next to the *Box* in the room vacated then in the changed state this no longer is true"

As a result of this formulation, we can define operator applicability as follows:

$$
\begin{aligned}
&applicable(op(I, P, N, C), S) \Leftrightarrow \\
&\qquad (\forall\, P_v \in P, \exists\, c \in dom(S) : S(c)\ satisfies\ P_v) \wedge \\
&\qquad (\forall\, x \in N, \exists\, c \in dom(S) : S(c)\ satisfies\ LHS(x))
\end{aligned}
$$

for any $S \in VSS$ (*LHS* returns the left hand side of the condition - effect pair).

2.4 An object-centred planning algorithm

An object-centred, non-deterministic, total order planning algorithm is shown in figure 1. It searches through a space of partial plans of the form:

$$node(N, A, I, G, S, P)$$

where N is the node's identifier, A is the parent's identifier, $I \in VSS, G \in GS$. S is the partial solution containing the operator sequence that already has been applied to the problem's initial state to produce the node's current state I. P is the purpose of the node's existence. The *method* whose preconditions the node was established to solve is the purpose of the node.

In line 3 we introduce a termination condition defined as follows, for any $I \in VSS, G \in GC$:

$$achieve(I, G) \Leftrightarrow \forall\, c \in dom(G) : I(c) \in G(c)$$

In this case we say that goal condition G is met in state I.

In line 5. we use the function:

$difference(I, G, D)$

to do the goal-directed job of finding the difference between the objects' current states and their desired states. D is thus a set of pairs of the form (I(c),G(c)), where

$c \in dom(G) \wedge \neg\, (I(c) \in G(c))$

Input: $Init \in VSS$, $Goal \in GC$
Global: The Planning Domain Model
Output: Soln: Operator sequence

plan($Init$**,**$Goal$**,** $Soln$**)**
1. Store $node(root, null, Init, Goal, [\ ], null)$ as an open node;
2. Retract an open node $node(N, A, I, G, S, P)$ from the Store;
3. IF N = root and achieve(I, G) THEN exit with Soln:= S;
4. IF not achieve(I,G) THEN
5.   difference(I,G,D);
6.   Pick a pair $(I(c), g)$, where $(I(c), G(c)) \in D$ for some object c, and $g \in g(c)$;
7.   Pick a method m that reduces the difference $(I(c), g)$;
8.   IF applicable(m,I) THEN
9. $I\_advanced$:= Apply m to I;
10.   Store $node(N, A, I\_advanced, G, S + +m, P)$ as an open node
11.   ELSE
12.   Pick an instantiation of m, call it m';
13.   Generate $Wp \in GC$, the weakest precondition of m' with respect to g;
14.   Store $node(New, N, I, Wp, [], m')$
15. ELSE
16.   Retract $node(A, Ap, Ip, Gp, Sp, Pp)$ from Store;
17.   Let $Ip\_Advanced$:= Apply P to (Apply S to Ip);
18.   Store $node(A, Ip, Ip\_Advanced, Gp, Sp + +S + +P, Pp)$ as an open node
19. GOTO step 2

Figure 1: An object-centred planning algorithm

In line 7. a *method* is in the most simple terms a domain operator, but it can easily be generalised to some pre-compiled operator sequence as discussed in the next section.

If the method is applicable, then it is applied to the existing node, solving one of the goals for an object (lines 9-10). If not then the weakest precondition of an instantiation of the method is generated in the form of a goal condition Wp, and this is used to create a new open node (lines 12-14).

The steps following deal with the situation where a node's goal condition is achieved in its current state. The solution sequence is applied to the parent's current state, followed by the purpose (method) of the successful node. The parent is then re-asserted with this advanced state.

2.5 The benefits of plan generation at the object level

The clean theoretical aspects of classical planning can be kept at the object level, and even using our basic object representation (that is without object hierarchies, aggregation, sub-classing, and so forth) it can be seen as a step forward towards managing the complexity of larger domain models for pre-condition planners. Although this formulation does not solve any combinatorial problems inherent in planning, it does bring the following benefits:

- It clarifies the meaning of planning states, operators, operators' conditional effects, and tightens which predicates can be used in operator components.
- It leads naturally to integrity rules and proof obligations for making sure that a domain model is self-consistent (in [16] we give a proof that *pushthrudoor* is consistent). Up to now domain modelling in classical planning seems to have ignored the issue of what is a valid state in an application, and what is a valid operator.
- The differencing that results in the pairs (I(c),G(c)) focusses attention on those objects which are not in the required substate. This allows *compilation* techniques to produce generalised plan fragments spanning (I(c),G(c)) for each sort, which, because they are indexed by an initial substate as well as a goal, can be highly effective. We detail the kind of speed-up this can bring to a literal-based planner in [16].
- Emphasis on valid object substates, and state invariants, allows compilation to produce strong heuristics for goal ordering, along the lines of [15]. The choice of which object to work on during plan generation is helped by such pre-processing tools.
- The plan generation function can effectively be embedded in a tools environment as detailed in [18]. This provides support for verification and validation of the domain model which supports the planning function.

We are also investigating extensions to the framework along the lines of adding sort hierarchies and object aggregation, which should further simplify domain models. Another promising area for exploration is the use of sorts and substates to cleanly capture non-deterministic operators, and domains in which the exact substate of an object is unknown. We can remove the constraint that a valid state is a kind of vector of substates; instead a valid state is generalised to be a vector of **sets of substates**. Incomplete information about an object means that it is represented as a set of substates - i.e. the range of possible situations in the domain that it could be in. Likewise, the constraints on operators can be loosened so that their effect causes an object to be in a set of possible states. This can be done, for example, by allowing the right hand sides of the substate change rules in operators to match on more than one substate class. When an operator is executed it will therefore leave the object in a set of substates. The attraction in using object-centred representation for this extension would be that the range of states (and hence uncertainty) is bounded by the range of possible substates specified for each object.

3 Related Work

"A representational scheme in which the facts are indexed by terms denoting entities or objects of the domain is called an object-centred representation" [21]. An object-centred planning representation can be seen as the identification of important classes of objects in the problem domain, and associating with them related information.

This object-centred approach enables us to exploit the structure of the problem domain, something advocated by Jonsson and Backstrom, [12], who point out that many types of planning problem are highly structured and planners can improve their efficiency by making use of this. They exploit this structure by looking at structural restrictions in a state-transition graph induced by the operators, in contrast to our approach which is motivated mainly by domain engineering issues. Aitken and Shadbolt [1] favour the use of a structured approach to knowledge based system development (KADS), and believe there are benefits to be gained by having a clear separation of domain and control knowledge. Barros, Valente and Benjamins [3] also advocate knowledge-level analysis, that is, of the way knowledge is used and represented in planning systems and believe there are considerable benefits to be gained from this in terms of planning application success.

3.1 Object-orientation in planning in the past and now

In this section, we will look at other work that has taken an object based approach. Some early work was carried out by Nilsson in [21]. Here he discussed representations which combined a number of related predicate calculus expressions into units that were identified with important objects in the problem domain. All information about a particular object was contained in one of these units, so that all relevant facts could be retrieved together. Events and binary relations were used for expressing facts about objects, which gave the advantage of modularity. Some AI systems have been developed which employed versions of unit language [4, 24, 10]. This type of representation is related to Minsky's frames [19], Moore and Newell's beta-structures [20] and Schank and Abelson's scripts [22].

PLANRIK [7] is a domain independent hierarchical nonlinear planner which uses object states. The system uses information stored in a knowledge base about the order in which object plans have to be achieved. For each object chosen during planning a network of required object states (nros) is created. Precedences are obtained among these object states by doing a topological sort of the nodes of the nros to get a linear sequence of object states. A partial linear plan is then generated for the selected object that satisfies this sequence, and the object plan can then be merged with the global plan. It appears that a precedence interpretation table is created for each object, during plan generation, and that the strategy or ordering layer is recreated for each new planning problem. The authors do not discuss the efficiency implications of this, but we note that the work differs here from our approach which is to make use of a stronger domain model, allowing goal orders to be precompiled and applied to any planning problem posed for the problem domain.

Sheu and Kashyap's [23] work on object-based process planning in automatic manufacturing environments uses an object-oriented knowledge base. Parts are described in a knowledge base, and similar parts abstracted into families. Parts can then be matched against these families, and if identified, the associated manufacturing knowledge can be used. If the part can't be identified the knowledge base provides a set of high level operators. Apparently they do not incorporate the object based approach into the actual generation of plans.

In Kazi's [13] proposed object-oriented reactive planning system objects are to be represented in an increasingly specialised sequence of objects in an inheritance hierarchy and plan fragments are associated with objects. This object-oriented knowledge base of objects and actions and the plan fragments are used in conjunction with the reactive planner. Again, this work does not seem to incorporate the object centred approach into plan generation, but to restrict its use to the creation of abstractions and hierarchies in the knowledge base.

3.2 Pre-processing and static analysis

The use of an object-centred approach to domain encoding and planning makes the exploitation of the structure of the domain through the use of pre-processing techniques particularly appropriate. Here we look at some recent work on pre-processing and static analysis, which includes that by Gerevini and Schubert [11] on the precomputation of parameter domains and state constraints, Kelleher's work on the automatic generation of domain constraints from operators [14] and Chien's static and completion analysis techniques to detect syntactic errors and debug the planning knowledge base [5].

Kelleher [14] presents the conditions for operators which allow the derivation of domain constraints, specifying impossible or invalid states or combinations of states. The work was motivated by the fact that writing operators and constraints can be a difficult task, especially as constraints often represent implicit and obvious information. His method is suitable for operators with or without variables, and although it is necessary to have some restrictions on the expressiveness of the operators these are no worse than in many other "practical" planners. The method makes use of necessarily co-occurring states to derive domain constraints.

Chien [5] uses static analysis to detect syntactic errors and completion analysis to debug the knowledge base. Static analysis, which can be used both on- and off-line, checks if pre-specified problem-classes are solvable - that is, it can help in the detection of unachievable goals. Completion analysis tools partially automate the process of constructing subproblems to detect bugs. These tools have been used in conjunction with both a task reduction planner and an operator-based planner, and Chien presents empirical evidence of their usefulness.

Gerevini and Schubert [11] obtain planning speed-up by inferring parameter domains of operators. These are pre-processed. Their method uses forward propagation of sets of constants occurring in the initial conditions and in operator postconditions - that is, they are precomputing sets of possible values of preconditions. They note that pre-processing is also useful for debugging domain models and point out that pre-processing costs are generally small in comparison with planning times, and that their utility is well worth the cost.

4 Conclusions

In conclusion, there is evidence of an increasing interest in the applications of object-oriented techniques to the planning process, but this, with the exception of the PLANRIK system, seems to consist mainly of the use of object-oriented knowledge bases, rather than planning at the object level. Our previous work has shown the benefits that can be obtained by using a systematic, object-centred approach to the creation of domain models, and further, that compilation techniques applied to this model can greatly increase plan generation efficiency. Here we have presented an extension to that work in the form of a design for an object-centred planner. We believe that our approach will begin to help in the management of the complexity of larger domain models, focussing attention on issues such as what constitutes a valid state or a valid operator. Our future work will concentrate on integrating the object-centred planner with a supporting tools environment, and in empirically evaluating its effectiveness in plan generation.

References

[1] J. Aitken and N. Shadbolt. Knowledge level planning. In C. Bäckström and E. Sandewall, editor, *Current Trends in AI Planning.* IOS Press, 1994.

[2] J.F. Allen, G. M Ferguson, and L.K. Schubert. Planning in Complex Worlds via Mixed-Iniative Interaction. In Austin Tate, editor, *Advanced Planning Technology*, pages 53–60 . IOS Press, 1996.

[3] L. Barros, A. Valente, and R. Benjamin. Modeling Planning Tasks. In *Third International Conference on Artificial Intelligence Planning Systems*, 1996.

[4] D.G. Bobrow and T. Winograd. An Overview of KRL A Knowledge Representation Language. *Cognitive Science*, 1:3 – 46, 1977.

[5] S. Chien. Static and Completion Analysis for Planning Knowledge Base Development and Verification. In *Third International Conference on Artificial Intelligence Planning Systems*, 1996.

[6] K. Currie and A. Tate. O-Plan: the open planning architecture. *Artificial Intelligence*, 52:49 – 86, 1991.

[7] E. Diaz-Infante and C. Zozoya-Gorostiza. PLANRIK: A Hierarchical Nonlinear Planner Based on Object States. In *Proceedings of the Florida AI Research Symposium*, 1996. To be published.

[8] M. Drummond. On precondition achievement and the computational economics of automatic planning. In C. Bäckström and E. Sandewall, editor, *Current Trends in AI Planning*. IOS Press, 1994.

[9] O. Etzioni. Why PRODIGY/EBL Works. In *Proc. AAAI*, 1990.

[10] P. Friedland. *Knowledge Based Experiment Design in Molecular Genetics*. PhD thesis, Computer Science Department, Stanford University, 1979.

[11] A. Gerevini and L. Schubert. Computing Parameter Domains as an Aid to Planning. In *Third International Conference on Artificial Intelligence Planning Systems*, 1996.

[12] P. Jonsson and C. Bäckström. Tractable Planning with State Variables by Exploiting Structural Restrictions. In *Twelfth National Conference on Artificial Intelligence*, 1994.

[13] Zunaid Kazi. Integrating Human-Computer Interaction with Reactive Planning for a Telerobotic System. Ph.D. Proposal,Department of Computer and Information Sciences, University of Delaware, DE, USA, 1994.

[14] G. Kelleher. Determining General Consequences of Sets of Actions. Technical Report CMS.14.96, School of Computing and Mathematical Sciences, Liverpool John Moores University, 1996.

[15] T. L. McCluskey and J. M. Porteous. On Extracting Goal Structure from Planning Domain Specifications. In *Proceedings of the 14th Workshop of the UK Planning SIG*, 1995.

[16] T. L. McCluskey and J. M. Porteous. Engineering and Compiling Planning Domain Models to Promote Validity and Efficiency. Technical Report RR9606, School of Computing and Maths, University of Huddersfield, 1996.

[17] T. L. McCluskey and J. M. Porteous. Planning Speed-up via Domain Model Compilation. In M. Ghallab and A. Milani, editors, *New Directions in AI Planning*, pages 233–244 . IOS Press, 1996.

[18] T.L. McCluskey, D.E. Kitchin, and J.M. Porteous. Object-Centred Planning: Lifting Classical Planning from the Literal Level to the Object Level. In *Proceedings of 8th IEEE International Conference on Tools with Artificial Intelligence*, 1996. To be published.

[19] M. Minsky. A framework for representing knowledge. In P. H. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, 1975.

[20] J. Moore and A. Newell. How can MERLIN understand? In L. Gregg, editor, *Knowledge and Cognition*. Lawrence Erlbaum Assoc, 1973.

[21] N.J. Nilsson. *Principles of Artificial Intelligence*. Springer-Verlag, 1982.

[22] R.C. Schank and R.P. Abelson. *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum Assoc, 1977.

[23] P. Sheu and R.L. Kashyap. Object-based Process Planning in Automatic Manufacturing Environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1987.

[24] M. Stefik. An examination of a frame-structured representation system. In *Sixth International Joint Conference on Artificial Intelligence*, 1979.