A Proposed Perspective Shift: Viewing Specification Design as a Planning Problem

Stephen Fickas
John Anderson

CIS-TR-88-15
November 17, 1988

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE
UNIVERSITY OF OREGON

A Proposed Perspective Shift: Viewing Specification Design as a Planning Problem

Stephen Fickas

John Anderson

Computer Science Department
University of Oregon
Eugene, OR. 97403

Abstract

We argue that in certain problem domains, AI planning can be viewed as a foundation for generation, critiquing, and elaboration of a specification. Two specification design projects in our group are used as a focus of discussion.

1. Introduction

Our interest is in formal models of the specification process. In this paper we argue that with the right perspective and set of assumptions, work in AI planning may be brought to bear on three aspects of designing a software specification: proposing a specification (generation); analyzing a proposed specification for problems (critiquing); modifying a specification to mitigate problems (elaboration).

One of our primary assumptions is that a designed artifact is intended to achieve certain goal(s) of the potential user(s) of the artifact. In particular, tools are designed to allow the user to carry out certain plans which will achieve his/her goals.

There are three ways for the designed artifact to enable the plans of the user. First, the artifact could be an instrument which the user needs to complete an operation. That is, the designer is providing an operand which fills a role in an operator. Second, the design could specify the relationships between two or more objects. These relationships could satisfy the preconditions of an operator. For example, putting a light switch near a door makes it possible to turn on the light upon entering a room. Finally, the artifact could itself be an operator. This would be the case in specifying a software system to manage some process, e.g., an automated library system.

For an automated planner to be of use in specification design, we believe that it is important for the system to be able to reason about user goals and plans. In particular, it is not only important for a design to enable certain goals of potential users; it is also important to disable the goals of potential mis-users, i.e., the system must have knowledge of potentially harmful plans that should be prevented or at least made difficult to achieve.

Finally, it is important for a specification system to have knowledge of likely plan failures. Even if a plan is enabled, it is not always carried out correctly. The system should be able to identify potentially harmful outcomes of incorrectly applied plans.

1.1 The Place of Specification Design in Planning Research

The two primary differences between specification design and the more traditional planning paradigm are:

- There is no way for the designer to know exactly what conditions will exist when a plan involving the designed artifact is executed.
- There is no chance for patching a plan at execution time.

These two facts bring certain aspects of the planning process to the foreground, while eliminating other issues from consideration. We are forced to consider issues concerning potential plan failure, but do not need to consider interleaving planning and execution.

Because of the lack of opportunity to patch plans at execution time, it is important that the designer consider every possible set of circumstances in advance. Since most designs have at least a few novel features, it is impossible to have prior experience with the exact version of the artifact to be produced. Therefore, it is important to be able to bring experience from closely related designs to bear in predicting potential problems and failures. One of the primary issues that we are exploring is the reuse of previous experience in planning and problem solving.

In this paper we will use two example systems as a basis of our discussion. The first is an existing computer-based critic system used to critique a specification. We will discuss our attempts to recast it in a planning light, and then to extend the system to deal with specification elaboration as well. The second system we discuss is one designed to study questions in abstract, analogical planning. Here we will discuss our attempt to apply it to the problem of generating a specification through reuse of previous analysis.

Before either system can be discussed, we must lay out the general set of assumptions we make in coming to our planning view.

2. Basic Assumptions

To use a planning approach in the design of a specification for a system S, we make the following assumptions:

1. A functional specification of S can be represented in terms of a set of objects Obj, a set of relations Rel, and a set of operators Op that add and delete members of Obj and Rel.
2. The "requirements" of S can be stated as of two types, achievement goals Gach and avoidance goals Gav, both of which represent a state or set of states described in

terms of elements in Obj and Rel.

3. An achievement goal Gach is met when some sequence of operator applications from Op (i.e., a plan) can be found that will transform an acceptable initial state into a state in which Gach is satisfied.

4. An avoidance goal Gav is met when no sequence of operators applications from Op (i.e., no plan) can be found that will transform any acceptable initial state into a state in which Gav is satisfied.

5. The set of acceptable initial states of S, used in the planning process discussed in steps 3 and 4 above, can be represented as partial state descriptions.

6. A functional specification is said to be *non-satisfying* when a requirement is not met, i.e., given a particular operator set, there exists a goal Gach that cannot be satisfied or a goal Gav that can.

7. *Incompleteness* is a type of non-satisfaction that occurs when an achievement goal cannot be met.

8. *Unsafeness* is a type of non-satisfaction that occurs when an avoidance goal cannot be met.

9. *Conflict* is a type of non-satisfaction that occurs when an operator allows a specification to be at the same time complete but unsafe (or vice versa).

Given that one is willing to live under these assumptions and definitions, we argue that construction of a functional specification can be viewed at least partly as a planning process: we must be able to find a sequence of operators (i.e., a plan) that satisfies a goal Gach (for all Gach) while at the same time prohibiting any plan that satisfies a goal Gav (for all Gav). This translates into a problem of constructing the sets Obj, Rel and Op, given the sets Gach and Gav.

For the purposes of this paper, we will assume that Obj and Rel are predefined and given to us. We thus limit the specification problem to one of constructing an operator set Op that satisfies the requirements. One can view this as a three step process: 1) generate a candidate operator set X, 2) analyze whether X is satisfying, i.e., meets achievement and avoidance goals, and 3) if X is non-satisfying, modify X to overcome specification problems (e.g., unsafeness, incompleteness).

The first system we present assumes that step 1 has been carried out prior to its invocation; its focus is on step 2, critiquing. Our discussion of the system will be of its use as a basis for a new system that acts as an operator set critic, using planning to form a critique. Given this shift in perspective, we will also discuss extending the new system to modify an operator set that has come under criticism, i.e., step 3.

The second system we present addresses step 1: with modifications we discuss, it can be used to generate a candidate operator set Op given the sets Obj, Rel, Gach, Gav and domain knowledge from the user. We will briefly describe its operation in generating Op and further extensions we see possible for addressing critiquing and modification as well.

The example domain we will use in this paper is that of physical resource management systems. Objects in this domain are things like books, video tapes, patrons, customers, staff. Relations are things like item-borrowed, password-of. Typical operators are check-out, check-in, query-for-who-has-what-item.

3. Critiquing an operator set

We have built a specification critic system that acts as the basis for our discussion of operator set analysis in a functional specification [Fickas and Nagarajan, 1988]. We will first discuss our existing critic, and then our proposed changes to view it as a planning system.

The critic takes as input a formal specification S and a ''context'' in which to analyze S. The critic outputs, through both text and simulation, a critique of S in the particular context. The specification language the critic accepts is an extended form of Petri Net.

The critic itself has three major components: 1) a model of the domain, 2) a matcher for connecting portions of the model to the specification under analysis, and 3) a critiquer for supplying the analysis. The critic's model is used to represent 1) a set of policy issues for building systems in a particular domain D, and 2) a set of relevant specification cases to consider for systems in D. The domain D we have chosen initially is that of automated library systems. The policy issues and cases we use are taken from 1) written texts and articles on analyzing problems in the this domain, and 2) protocols of human analysts, familiar with the domain, constructing and critiquing specifications.

3.1 Policies and Cases

In the critic's model, policy issues represent *potential* achievement and avoidance goals of the system. Based on discussions with domain experts and a study of the domain literature, seven broad policy issues were defined for resource borrowing systems[1]:

1. Allow users to have a large selection to choose from.
2. Allow users to gain access to a useful working set and keep it as long as necessary.
3. Maintain the privacy of users.
4. Recognize the human dynamics of group (patron, staff, administration) membership.
5. Account for human foibles, e.g., forgetting, losing items, stealing.
6. Account for development resource limitations, e.g., money, staff, and time available to develop the system.
7. Account for production environment limitations, e.g., money, staff, and time available to run and maintain the delivered system.

[1]We make no claim that this is either a necessary or sufficient list of policies, but simply one that has allowed us to handle the set of resource management problems that we have studied to date. Also, it is clear that certain policies in this list extend beyond this domain.

Each of these seven can be further refined, e.g., maintain privacy of users' borrowing record, maintain privacy of users' queries, etc.

To become a goal, a policy issue must take on a high utility value. These utility values must be supplied by the user to set the context for analysis. Currently we use a qualitative approach to utility, giving each policy issue a value of *important* or *unimportant* (a value of *unknown* is also possible). A value given to policy issue P is inherited by all ancestors of P. Thus, marking the policy issue of accounting for human foibles as important will in turn mark all refinements of that policy -- forgetting books, stealing video tapes -- as important. Conversely, we can mark policies in a finer grain if necessary: prevention of stolen items is unimportant, but reminding forgetful users of borrowed items is important.

A policy issue that is given a value of important is viewed as a goal of the system to be specified. For instance, if the user gives the policy issue of maintaining user privacy a value of important, then privacy becomes a goal to achieve, and is part of the context in which the critic will analyze the corresponding specification. For our purposes, this will become an avoidance goal: avoid a state where one user can gain access to information about another.

Note that by choosing various policy values, we can ''take the view'' of various system users. For instance, if we take the (selfish) view of a user of a library, we might mark a large selection set and unlimited borrowing as important, and all other policies as unimportant. We can later change policy values to reflect the library administration's view by marking privacy and minimization of cost as important, and rerun the critic. In summary, by changing the context (i.e., the achievement and avoidance goals), we can get radically different analysis of the same specification (i.e., operator set).

The other important component of the critic's model is a catalog of cases. Each case can be viewed as representing a scenario, possibly abstract, that can lead to the satisfaction of one or more achievement and/or avoidance goals. In essence, the catalog of cases attempts to capture the typical ways a system may be used and misused by outside agents.

3.2 Type of analysis

Given a specification and context (in the form of achievement and avoidance goals), the critic produces three classes of criticism:

Non-support: An achievement goal marked as *important* is satisfied by a case C. A match for C is not found in the specification.

Obstruction: An avoidance goal marked as *important* is satisfied by a case C. A match for C is found in the specification.

Superfluousness: An achievement goal marked as *unimportant* is satisfied by a case C. A match for C is found in the specification.

The third critique is based on the notion that components added to a specification in support of an unimportant goal will tend to add unnecessarily both to the complexity of the system, and to the cost of its operation and maintenance.

Note that it is possible, and in fact typical, for the same case plan to satisfy both achievement and avoidance goals simultaneously. For example, suppose both the achievement goal of keeping a good stock on hand, and the avoidance goal of removing necessary resources from a user's working set are marked as important. The case *force turnaround* allows both goals to be achieved, leading to a contradiction. While we would like to think that goals such as this are never both simultaneously marked as important, it is not untypical for a client to describe a conflicting set of goals. In fact, most borrowing systems can be viewed almost solely as compromises between conflicting concerns. One key process in specification design is then coming to grips with conflicting policies through various forms of trade-off and compromise.

3.3 A new plan-based critic

There are several representational changes to be made to define our new plan-based critiquing system. First, we must move from Petri Nets to the sets Obj, Rel, and Op as our specification representation. For example, the Petri Net in figure 1 would be translated into a check out operator with Obj consisting of books and patrons, Rel consisting of available-books, potential-borrowers, and checked-out, and Op consisting of the single operator check-out that has preconditions of a potential borrower and an available book, removing the available book and marking it as checked out..

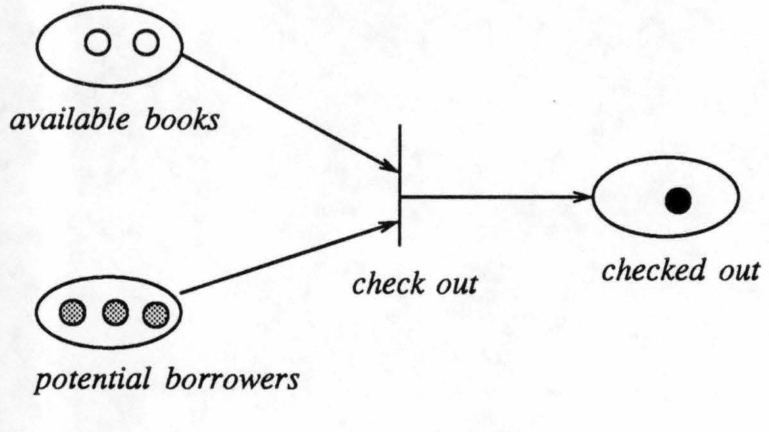

Figure 1. The check out transition

Second, we must restate our policy issues in terms of the goal sets Gach and Gav. For example, the policy issue of giving users a useful working set is translated into the achievement goal of need(Patron,Book) *and* checked-out(Patron,Book).

Third, the cases must be viewed as storing plans as opposed to scenarios, i.e., as a sequence of operator applications that lead to a goal or goals being satisfied, given an initial state. This falls out naturally from the two translations discussed above.

Finally, the critic's matcher is responsible for binding components in a case to components in the specification under analysis. In the old critic, this involved matching case scenarios to components in the input Petri Net representing the specification. Now we must map a case's Obj, Rel and Op sets to the equivalent sets in the specification. We argue that the same interactive style of matching will apply.

3.4 Mitigating criticism

To summarize, we propose a critic that takes as input an operator set Op and a set of achievement and avoidance goals Gach and Gav. The critic has a catalog of cases or plans indexed to Gach and Gav. Each case represents a typical plan for satisfying members of either or both goal sets. The critic analyzes the input set Op by attempting to match operators in Op to operators in its case plans.

Stated in another way, the critic attempts to find paths that users might take in a system to achieve both good and bad ends. With paths viewed as plans, this is a type of case-based planning process. We are not trying to plan the construction of the operator set itself (that is taken up in section 4), but instead to use one that is given to us to construct (actually match on) plans that will satisfy requirements.

While we believe this is an important use of planning in critiquing a functional specification, its use as we have described it is mainly one of explanation -- the critic shows how an operator set can be put to good and bad use relative to a set of requirements. Our more long term interest is in the next step in the analysis process, what to do with a non-satisfying specification, one in which achievement goals cannot be satisfied and/or avoidance goals can. In particular, how can we modify the specification's operator set to mitigate such problems?

An example may be useful here. The critic contains a case that achieves an avoidance goal of one user finding out what another user has borrowed. Given an initial state of two borrowers A and B, and a resource R to be borrowed, the plan for this case is as follows: borrower B checks out resource R; borrower C gains access to B's identity; C queries the system, as B, to find what resources B has checked out; C learns that B has checked out R.

Suppose the user has marked the relevant goal as important to avoid, and has supplied an operator set that includes a check out action and a query action. Suppose further that the query action has a precondition that users can only query on their own borrowing record.

When we run the critic, the case described above matches with the operator set supplied by the user. The plan step not accounted for by the user, gain access to another's ID, is assumed to be available to plans in this domain unless specifically prohibited by the user. Thus, the critic's domain model includes components in the environment as well as components in the system to be built.

We have an undesirable situation: a goal the user marked as important to avoid is shown to be achievable with a particular plan. Can we modify the plan to avoid the bad outcome from being reached? One library analyst we interviewed suggested we focus on disabling the C-queries-as-B action by monitoring the query action with a staff person [Fickas *et al*, 1988]. Most computer-based systems today take the other route, and attempt to make it difficult to carry out the gain-access-of-other's-ID action by including various password schemes.

Of course, one might question all of this intricate modification to the plan when we can simply remove the query operator from the specification, thus squashing the plan once and for all. This in fact would be feasible if the query operator was not playing a key role in any plans satisfying an important achievement goal. In other words, does removing the query operator from the user's operator set cause an achievement goal to become non-satisfiable?

We can see how this might happen by looking at another case, one that satisfies the achievement goal of accounting for human forgetfulness. The case plan is as follows: borrower B checks out resource R; B moves to a state in which he or she has forgotten the identity of R; B queries the system to find what resources B has checked out; B learns that B has checked out R; B moves to a state in which he or she is aware of the identity of R; B checks the resource R back in (eventually).

If the goal of accounting for human forgetfulness is marked as important, then the query operator will match this plan; the forgetting action is again part of the domain model of the critic, and is assumed to be available for any case that needs it in matching its plan. If we remove the query operator, then one of three outcomes is possible: 1) another operator from the specification's operator set can be found to match this plan, 2) no other operator can be found to match this plan, but yet another plan is found that satisfies the goal (i.e., this plan is redundant), 3) no plan can be found to satisfy the goal (i.e., we have a conflict with regards to the query operator). It is the third case that will lead us to consider the more subtle types of plan modifications discussed.

Finally, we note that these two examples can be reversed. We may start out with no query operator, note that an achievement goal is not being satisfied, add the query operator to remedy the situation, and then note that an avoidance goal has now become satisfied.

Given all this, we can make a first cut at a general statement of the problem for the new critic[2]:

> **Incompleteness**: An achievement goal marked as important is satisfied by a set of cases C. A match for an element of C is not found in the specification. *Modify the operator set so that at least one case in C matches.*
>
> **Nonsafeness**: An avoidance goal marked as important is satisfied by a set of cases C. A match for at least one element of C is found in the specification. *Modify the operator set so that no case in C matches.*
>
> **Superfluousness**: An operator O is not used in any plan that satisfies an achievement goal marked as important. *Remove O from the operator set.*

[2]Note that the following assumes that we have captured all possible plans for achieving domain goals. More realistically, we will need an auxiliary planner that is able to build plans on the fly.

While this is the problem statement, it is clearly not the problem solution. In particular, which cases should we focus on when more than one is a candidate? And given a focus, which modifications should be made to that case's plan to least affect the achievement of the requirements as a whole?

We have begun to look for answers to these questions form several sources. First, it seems clear we must bring some discipline to modifications made to the operator set: a potentially infinite number are possible. We believe that this discipline can at least partially be achieved by representing the resource constraints of the system, and the cost a modification has in terms of resources used. As an example, what would be the cost of having a staff person oversee (or actually carry out) all queries to the borrowing record, a modification suggested by one analyst to avoid user privacy being violated?

Of course, there are other resources than the standard money and time. For instance, in an earlier specification we were working on in the domain of sporting events, the question came up of how to modify the specification to avoid tie games. Here the resource is human endurance. In some games, extra innings are allowed up until some maximum. In boxing, this does not work. Instead, judges are not allowed to produce ties, i.e., we do not allow a tie score after regulation time. In any case, paying attention to overall resources available and representing operator costs provides some restriction on the set of possible modifications (see [Kant, 1983] and [Fickas, 1985] for related work on problem solving costs).

Second, we have found that conflicts arising in a specification of a well charted domain are typically not novel. Further, solutions to well known conflicts are often themselves well known to experts in the domain. Why shouldn't these conflict resolutions be stored in a catalog as well? For instance, there are various well known compromises between the conflicting goals of supply versus demand, and the goals of privacy versus enforcement. Why not tie conflicts arising in our critic's case plans to these compromises?

Indeed, Sycara has explored this approach of cataloging compromise solutions in the domain of labor-management bargaining [Sycara, 1987]. A member of our group, Bill Robinson, is attempting to incorporate a similar mechanism into a specification designer [Robinson, 1988], a tool that we believe may play an important role in the operator modification problem.

Third, expert analysts appear to have a set of domain heuristics for making modifications to a specification. For instance, an analyst was asked to comment on a functional specification that included an operator that kept a history, for each book, of who had checked the book out in the past. She first commented on the negative aspects of such a query, i.e., that it may allow user confidential information to be given out illegally. This is a case that is also caught by our critic using a case similar to the one discussed previously of one user gaining access to another's ID. However, she went further by suggesting a modification:

> ''Library statistics are interested in what category of borrower checks out what books. So they can look up and see that faculty checked it out or graduate students, or public borrowers, or even by department. I mean, it can be used for management information. However, you don't really need to know the individual who had it before.''

The analyst's proposed modification is from recording the individual borrower to one of recording his or her classification within the university. From this, we might define a general heuristic as follows:

if information N is being legally kept about user U1,
and it is possible for N to be gained illegally by user U2
then change N to a form N' that makes its possession by U2 legal

Clearly the form of N' will rely on domain knowledge.

Note that the approach suggested above is a modification of a specification's operator set *and* a modification of the Rel and Obj sets as well. Along these same lines, we might consider modifying the initial state of a plan that we are trying to prevent from occurring. For instance, one library analyst suggested we could, in essence, simply remove the borrowing record from a specification to prevent the case of one user gaining access to another user's borrowing record. As even more extreme measures, we could consider removing the resource R or allow no more than one borrower B. Any or all of these modifications to the plan's initial state would keep the plan from being enabled.

Carbonell suggests a similar disable-precondition heuristic in his POLITICS system to model planning and counterplanning among human agents [Carbonell, 1981]. Along the same lines, Wilensky defines a broad model of goal conflict and planning [Wilensky, 1983], again in the realm of interacting human agents. Taking the view that the system being specified is one active agent and the user the other, it appears that at least some of the components of Carbonell's and Wilensky's planning models can be incorporated here.

Finally, we can study specification modification from a process view. For instance, Feather has begun to classify domain-independent specification changes in terms of the ramifications such changes will have on the existing structure [Feather, 1989]. The goal of this work is to 1) identify what type of specification modification steps lead to conflict, and 2) catalog strategies for dealing with it. In essence, Feather is analyzing the *process* involved in designing a specification as opposed to attempting to analyze the *product* that ensues. This leads us into the second system we wish to discuss, one that is involved in the design process itself.

4. Generating the operator set

The last section discussed a specification analysis system, one that accepted a specification (a set of objects, relations, and operators) and requirements (a set of achievement and avoidance goals) as input, and produced a critique. In this section we consider a generation system, one that accepts as input just the objects, relations, and requirements, and produces a suggested operator set.

The generation system is built around a hierarchical planner that generates and uses abstract plans [Anderson and Farley, 88]. The planner is being used to explore issues in planning by analogy, planning given incomplete knowledge, and dealing with potential plan failures. We are using the design of formal specifications as one application of our planner. Before discussing this application, we will list the key components of our system:

- A mechanism that builds up an abstraction hierarchy of operators, given a set of primitive (executable) operators as input. The abstract operators in the hierarchy are generated automatically by extracting the shared preconditions, adds and deletes of two or more primitive operators. For example, checking out a library book shares certain aspects with checking out a basketball from the PE department. An abstract "check-out" operator is created and made the parent of the two primitive operators.
- A mechanism that builds up an object hierarchy, based on the roles of objects in similar operators. For example, "PE-dept-basketball" and "library-book" play corresponding roles in the two primitive "check-out" operators above. Thus, they both belong to the class of "check-out-able" objects. An abstract object is created to represent this class. The abstract object fills the appropriate role in the abstract operator.
- A mechanism that generates abstract plans from executable plans by replacing the primitive operators with abstract operators. The abstract plans are stored in the operator hierarchy as macro-operators. This process has several advantages over storing primitive plans: reduction in storage space, since one abstract plan covers several primitive plans; reduction in search time, since fewer plans are stored; and the potential for analogical planning, since an abstract plan formed in one situation can be specialized in different ways for different situations.
- A general-purpose, hierarchical planner. Our planner inherits many features from NOAH [Sacerdoti 77] and Friedland's MOLGEN planner [Friedland and Iwasaki 85].
- A mechanism for retrieving potential failures while generating a plan. Plans are often attempted even when some of their preconditions are not satisfied. For example, one may go to the video store to check out a particular movie, only to find that the video has been checked out by someone else. By storing outcomes of failed plans, the planner can predict likely failures at the same time it is retrieving a plan. (This mechanism has not been implemented.)

We propose using our system in the following way to help in constructing a functional specification for a system S:

1. To construct a specification for S, we will start with the objects, relations, and goals of S. Our approach will be to generate the functions (i.e., the operator set) of S using the given information as indices into the various hierarchies kept by the planning system.
2. In the simplest case, the operators to appear in the specification are a subset of the operators in the planner's database. The planner retrieves and/or generates a set of plans, call it Pset, for achieving the goals of S. All of the primitive operators that appear in any plan in Pset go into the initial specification set, call it Xset. This set is then criticized as described in the previous section, to detect any avoidance goals that can be achieved using the operators in Xset.
3. Xset is gradually refined during an iterative process in which the planner suggests a set of operators for Xset, the critic (planner) finds problems with the operator set, and the user makes suggestions for resolving the problems. Resolution may involve adding or removing operators, adjusting the preconditions on certain operators, or relax-

ing restrictions and allowing avoidance goals to be achieved under certain circumstances. This process is repeated until the user is satisfied with Xset and the goals it achieves.

The planning system can also be used to generate an operator set for a domain the planner knows nothing about. A planner "knows about" a domain if it has operators and plans for achieving goals in that domain. The system can be used to generate an operator set for S if it knows about one or more domains which are analogous to S. For example, if the planner knows a lot about libraries but nothing about video rental stores, this approach can be used if the proper correspondences are established between the two domains. The key to this method is the connection between the object and operator hierarchies.

The links between the object and operator hierarchies allow the formation of hypothetical operators: if the planner is told that a video-tape is a "check-out-able" object, it will assume that there is a "check-out-video-tape" operator that is a child of the general "check-out" operator. Thus, by giving a functional description of an object, the user can give the planner hints about new operators involving the object.

The user can save some effort and simply tell the system that video-tapes are like library-books. The planner will assume that all operators that apply to library-books (e.g., browse, check-out, return, copy) have counterparts that apply to video-tapes. However, this method is more likely to generate spurious operators (e.g., rip out a page of a video).

The ability of the planner to hypothesize the existence of operators based on functional properties of objects is used in the following way.

1. We place the objects in S into the planner's object hierarchy manually. Each object is placed into all of the relevant categories (e.g., liftable, check-out-able). Once we have placed the objects into the proper classes, the planner retrieves the abstract operators that are linked to these classes.

2. For our first cut at an operator set (Xset) for the system S, we make a copy of each abstract operator, replacing the abstract objects with objects from S.

3. We now map goals from S onto goals in the planner (again manually). This is accomplished by replacing predicates in the goals in S with predicates that the planner knows about. That is, we define the new goals in terms that are in the planner's vocabulary. Call the resultant set of planner goals Gset.

4. Next, the planner generates plans that achieve the goals in Gset. The set of plans for achieving them are returned; this is Pset. We filter Xset by removing any operators that are not part of Pset. If an operator is not part of a plan that is needed in the new domain, then it is superfluous. We now have a more refined version of our operator set for S.

5. We now attempt to massage the operators of Xset into a concrete form for use in S. This is also a manual process, in which the user in effect teaches the system about the new domain. Some operators may need to be altered to reflect aspects unique to the new domain. For example, checking out a video movie requires paying a fee, unlike checking out a library book. Other operators may have to be removed. Generally,

these operators serve some purpose that may need to be filled by an alternative operator. For example, viewing a video is analogous to reading a book, so the user may have to replace "read-video" with "watch-video".

As before, refining Xset is an iterative process. The planner proposes an operator set, the user ensures that the operators are legitimate operators in S, the planner tries to find problems with the operator set, and the user tries to resolve those problems.

The result of this process is a set of new operators that are analogous to operators that have been found useful in other domains. Stated briefly, we are using mappings provided by the user that tie the new domain into others that we have previously analyzed. Our planner stores its information in such a way that these mappings can be used to generate a candidate operator set for the new domain.

The advantage of using an abstract planning mechanism is that we can find commonalities between the new domain and domains we know more about. These commonalities provide hints about the types of operators that will be needed to achieve our goals in the new domain.

Along with its benefits, our approach has several shortcomings:

- We only work on achievement goals. Our planning system does not represent avoidance goals. While we are currently working on a means of reasoning about avoidance goals as well (for example, by fooling the planner into thinking that an avoidance goal is an achievement goal), we now must rely on a system similar to the critic of the last section to do this type of analysis.
- There is still much that is manual about our approach. The user must do the mappings of objects and relation/goals from the new domain to what is stored in the planner. Further, the user typically must supply domain knowledge of S in specializing the abstract operators of Xset to concrete ones in S. This is primarily because of the weak type of mapping information we support between objects in S and objects in the planner. For example, we can map a book to a video tape, but we do not capture the subtle and not so subtle differences between the two

In the end, responsibility is divided between system and user according to the abilities of each:

- The system stores and indexes operators.
- The system examines large numbers of operator sequences.
- The user provides domain-specific expert knowledge.

The next section summarizes our work on both the critic and abstract planner as tools for specification design.

5. Summary

We have argued that it may be useful to view certain types of specification problems from a planning perspective. In particular, if a specification can be viewed as a set of objects, relations, and operators, and the requirements as achievement and avoidance goals, then results in AI planning may be brought to bear.

To support our arguments, we have presented two specification design systems[3] that rest on a planning view and attempt to use planning results as a basis of specification generation, critiquing and modification. While neither system is fully implemented, we believe each shows potential for cracking knotty problems in specification design, albeit in restricted domains.

Acknowledgments

We acknowledge the National Science Foundation for support of this work under grant DCR-8603893. We also thank Marek Lees for his initial insight into the planning view.

References

[Anderson and Farley, 1988], Anderson, J., Farley, A., Plan abstraction based on operator generalization, In *Proceedings of AAAI-88*, St. Paul, 1988, pages 100-104.

[Carbonell, 1981] Carbonell, J., Counterplanning: a strategy-based model of adversary planning in real-world situations, *Artificial Intelligence*, V16, 1981

[Feather, 1989] Feather, M., Constructing specifications by combining parallel elaborations, To appear in *IEEE Transactions on Software Engineering*, Available from author at USC/ISI, 4676 Admiralty Way, Marina del Rey, Ca. 90292

[Fickas, 1985] Fickas, S., Automating the transformational development of software, In *IEEE Transactions on Software Engineering*, Vol. 11, No. 11 Nov. 1985

[Fickas *et al*, 1987] Fickas, S., Collins, S., Olivier, S., Problem Acquisition in Software Analysis: A Preliminary Study, Technical Report 87-04, August, 1987, Computer Science Department, University of Oregon, Eugene, OR. 97403

[Fickas and Nagarajan, 1988] Fickas, S., Nagarajan, P., Being suspicious: critiquing problem descriptions, In *Proceedings of AAAI-88*, St. Paul, 1988

[Friedland and Iwasaki, 1985] Friedland, P., Iwasaki, Y., The Concept and implementation of skeletal plans, Technical Report KSL 85-6, Stanford University, 1985.

[Kant, 1983] Kant, E., On the efficient synthesis of efficient programs, *Artificial Intelligence*, V20, 1983

[3] It is not clear to us that we actually need two separate systems. For instance, is the knowledge they represent really different? Or is their separation just an artifact of their roots in different research areas? Currently we believe the latter, and hence, one of our efforts is an attempt to combine the two systems into a single specification design tool.

[Sacerdoti, 1977] Sacerdoti, E., *A Structure for Plans and Behavior*, New York: American Elsevier, 1977.

[Robinson, 1988] Robinson, W., Automating the parallel elaboration of specifications: preliminary findings, Technical Report, Computer Science Dept., University of Oregon, Eugene, OR. 97403, 1988

[Sycara, 1987], Sycara, K., Resolving adversarial conflicts: an approach integrating case-based and analytic methods, Computer Science Dept., Georgia Institute of Technology, 1987

[Wilensky, 1983] Wilensky, R., *Planning and Understanding*, Addison Wesley, 1983