Integrating General Purpose Planners and Specialized Reasoners: Case Study of a Hybrid Planning Architecture

Subbarao Kambhampati, Mark R. Cutkosky, Jay M. Tenenbaum, and Soo Hong Lee

Abstract—**Many real-world planning problems involve substantial amounts of domain-specific reasoning that is either awkward or inefficient to encode in a general purpose planner. Previous approaches for planning in such domains have either been largely domain specific or have compromised with shallow models of the domain-specific considerations. In this paper, we propose a hybrid planning architecture for such domains, which utilizes a set of specialists to complement both the overall expressiveness and the efficiency of a traditional hierarchical planner. Such an architecture promises to retain the flexibility and generality of classical planning framework while allowing deeper and more efficient domain-specific reasoning through specialists. The architecture, however, has several ramifications on the internal operations of the planner as well as its interactions with the specialists. First, continual interactions between the planner and the specialists necessitate an incremental, interactive, and least-commitment oriented approach to planning. Second, as the planner and the specialists in such a model may employ heterogeneous reasoning mechanisms and representations, a complete understanding of the operations of one by the other is not possible. This necessitates designing interfaces at the right level of abstraction, to efficiently mediate the interactions between them. In this paper, we investigate these issues with the help of our implementation of a hybrid planning architecture for a manufacturing planning domain.**

I. Introduction

MANY complex real-world planning problems require significant amounts of deep domain-specific reasoning that is awkward or inefficient to encode into a traditional planner. An example of such problems is process planning for machining, which involves extensive reasoning about geometry, kinematics, and cutting and clamping forces. The classical planning framework, in which the planner is modeled as an isolated module with all knowledge relevant to plan generation at its disposal, is inadequate for addressing such problems because it is impractical to encode deep models of specialized considerations in the constrained-action representations used by classical planners. While extending the action representation sufficiently to encode these considerations is a possibility, it suffers from the drawback that the cost of planning becomes prohibitive as the expressiveness of the domain models increases [23]. Most previous approaches for planning in such situations have dealt with these issues either through very domain specific planning algorithms (e.g. [8]), or by restricting themselves to shallow models of the specialized considerations (e.g. [5], [26]).

In this paper, we investigate an alternative approach: a hybrid planning model that utilizes a set of specialists to complement the expressiveness and reasoning power of a traditional hierarchical planner. Such a model allows us to retain the flexibility and generality of the classical planning framework, while allowing deeper and more efficient domain-specific reasoning through the specialists. It can provide better computational efficiency since the specialists can employ methods that are best suited for particular kinds of analyses. It also facilitates better modularity by avoiding duplication of capabilities between the planner and the specialists.[1]

Planning in such a hybrid model does, however, place several constraints on the operation of the planner and the specialists, and raises many important issues regarding the exact role of the specialists, and the interfaces between them and the planner. To begin with, the specialists may be used to detect interactions that the planner itself cannot detect, or to extend the plan to make it satisfy additional constraints not modeled in the planner's own domain model. Further, some of these specialists may be involved in their own specialized planning (synthesis) activities. The analyses of the specialists may be dependent on the state of the plan, and the commitments made by the specialists may, in turn, have a direct bearing on the plan. Consequently, to avoid inconsistent commitments that could lead to costly intermodule backtracking, the planner and the specialists must each keep track of the constraints

Manuscript received March 22, 1992; revised February 17, 1993. This research was supported in part by the Office of Naval Research under contract N00014-88-K-0620 by the National Science Foundation under grant IRI-9210997, and in part by ARPA/ROME Laboratory Planning Initiative under grant F30602-93-C-0039. A preliminary version of this paper has been presented at the 9th National Conference on Artificial Intelligence, Anaheim, CA, July, 1991 [15].

S. Kambhampati was with the Center for Design Research and Department of Computer Science, Stanford University, Stanford, CA 94305. He is now with the Department of Computer Science and Engineering, Arizona State University, Tempe, AZ 85287.

M. S. Cutkosky and S. H. Lee are with the Center for Design Research, Stanford University, Stanford, CA 94305.

J. M. Tenenbaum is with Enterprise Integration Technology, Palo Alto, CA 94301. He is also affiliated with the Department of Computer Science, Stanford University, Stanford, CA 94305.

IEEE Log Number 9212933.

[1]For example, in process planning, similar geometry considerations are used by both assembly planners and fixture planners. Thus, keeping those considerations as a separate specialist avoids duplicating them.

imposed on their decisions by the commitments made by the others.

As the planner and the specialists may employ heterogeneous reasoning mechanisms and representations, a complete understanding of the operations of one by the other is not possible. To facilitate fruitful interactions between them, we need to design interfaces that are at the right level of abstraction to enable each to recognize the constraints placed on their results because of commitments made by the other.

Planning in such architectures is a continual, rather than a one-shot process. The constraints imposed by the specialists on the plan force the planner (and the specialists) to contend with a continually evolving problem specification. The evolutionary nature of planning has implications for the internal operation of the planner and the specialists. For example, hierarchical abstraction, and the ability to represent plans with partial commitment (partial ordering, etc.) are important for allowing the specialists maximum latitude in specializing the plan according to their considerations. More importantly, since inconsistent commitments between the planner and the specialists cannot be completely avoided, incremental operation, in terms of the ability to reuse previous results while accommodating new constraints [10], [12], [13], is essential for efficiency. (Note that in contrast to the classical planning model, where such replanning ability is justified purely in terms of the internal efficiency of the planner, here it is also motivated by the desire to promote efficient interaction between the planner and the specialists.)

In this paper, we investigate these issues with the help of our implementation of a hybrid planning architecture for a manufacturing planning domain. Given the description of parts in terms of their component features, and the corresponding geometric models, the objective of planning in this domain is to provide a plan for machining that part from raw stock. Generating such plans involves a significant amount of reasoning about geometry, kinematics and cutting, and clamping forces. Our implementation handles these specialized considerations by combining a domain-independent hierarchical planner with two domain specialists. In the rest of the paper, we describe our implementation of this hybrid planning architecture, and discuss the details of interfaces and interaction management between the planner and the specialists.

A. Organization

The rest of this paper is organized as follows: In Section II, we focus on the particular characteristics of our manufacturing planning domain that necessitate hybrid planning approach. In Section III, we present the hybrid architecture that we have implemented for planning in this domain, and discuss the operation of the planner and the specialists, as well as the interfaces between them. Section IV presents details of the nominal planning cycle in this architecture. Sections V and VI discuss the issues of intermodule backtracking, and incremental plan revision in this architecture. Section VII describes some limitations of our current implementation, and discusses the directions being explored to overcome them. Section VIII discusses related work, and Section IX concludes the paper.

II. Process Planning for Machining in Next-Cut: The Domain Characteristics

The domain that we address in our work is process planning for machined parts. The planner is part of a prototype concurrent design system called Next-Cut [3], in which planning and analysis are performed step-by-step as a designer constructs or modifies a design. In such a system, the planner serves two purposes: it generates plans for machining parts, and it provides the designer feedback about the manufacturing implications of design decisions.

The input to the planner consists of the description of a part in terms of features, dimensions, tolerances, and corresponding geometric models. Window I in Fig. 1 shows the geometric description of a simple component—which we shall refer to as `cross-product` (a cross-shaped component used for supporting a shaft). Window II in the same figure shows the description of one of its features, `hole-4`, in terms of its attributes, such as diameter, depth, and position tolerance. Window III shows part of a rough sketch of the process plan for machining `cross-product`. The process plan includes a sequence of "setups" (particular orientations in which the workpiece should be restrained using fixturing devices, such as a `vise` or `strap-clamps`), the set of machining operations (`drilling, milling, boring`) that should be carried out during each setup, and the tools (`0.25in-dia-twist-drill`) to be used during each machining operation.

There are several complexities involved in planning in this domain. First, there are typically interactions between different features such that machining one feature first may make it difficult or impossible to machine subsequent ones. What makes these interactions difficult from a classical planning point of view is that most are geometric in nature, and detecting them requires geometric reasoning. Similarly, determining the necessary setups involves considerable reasoning about the intermediate geometry of the part, as well as kinematic and force equilibrium analyses.

Encoding the geometric and force knowledge required for these analyses in a general purpose planner is impractical, both because of the awkwardness of translating the analytical procedures underlying such analyses into the planner's representation, and because of the subsequent inefficiency of planning with such detailed models. As a consequence, incorporating geometric and fixture related considerations into the planning has traditionally been a weak point of previous work in automated process planning. In some cases it is assumed that interactions have been enumerated before planning, perhaps by a human user (e.g., GARI [5] and PROPEL [26]). In other cases a

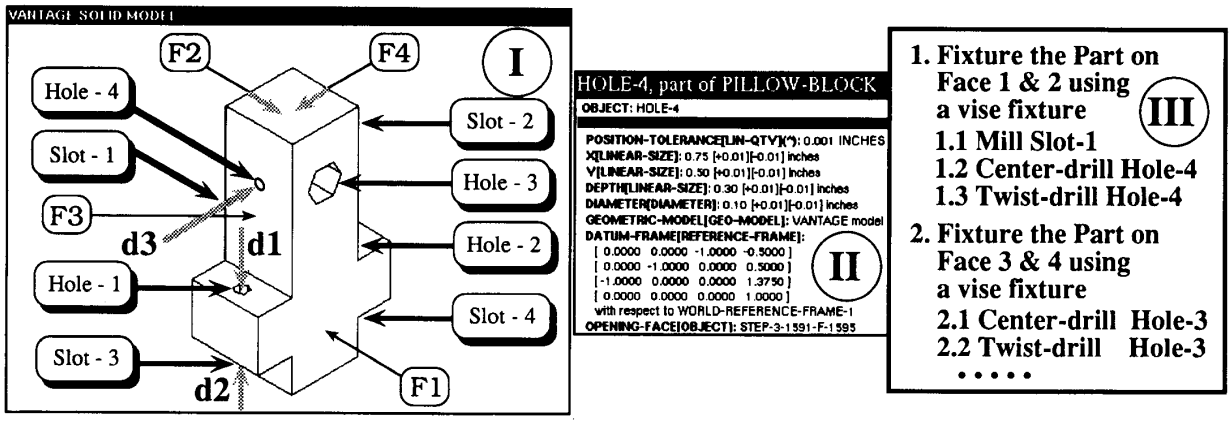


Fig. 1. Geometric and feature-based specification of a part and a fragment of the plan for machining it.

number of rules for detecting such interactions are incorporated as preprocessing steps to the planner itself (e.g., MACHINIST [8]). However, the rules are typically at a shallow level. Moreover, the level of abstraction is not made explicit and it is, therefore, hard to tell when the rules are sufficiently accurate. Obviously, a more systematic approach is to delegate the geometric and force related concerns to specialists best suited to handle them, and view process planning as a team activity involving interactions between the planner and these heterogeneous specialists.

III. PLANNING ARCHITECTURE IN NEXT-CUT

Fig. 2 shows the schematic of the planning architecture in the NEXT-CUT environment. A general purpose planner is used for selecting appropriate machining processes and tools and composing them into a machining plan. A geometry specialist is used to detect and resolve geometric interactions that arise during machining, and a fixturing specialist is used to decide the orientations and clamping forces for holding the part during machining.

There are two forms of communication between the planner and the specialists in the NEXT-CUT environment. The first, and more straightforward, is through the shared central model. The central model contains a description of the part in terms of its component features, the attributes of the features and their geometry, which all modules can access and modify. The planner and specialists can also communicate directly through specialized interfaces (e.g., interaction graph, and setup graph). The rationale here is that to facilitate a deeper cooperation between the planner and the specialists, they also need to have customized interfaces that reflect the shared representations. We will see that these interfaces facilitate efficient reasoning about interactions between the modules. In this paper, we will be concentrating on the interfaces between the planner and the two specialists. It should, however, be noted that customized interfaces can also exist between the specialists.[2] As can be seen from Fig. 2, in our implementation, the feature bounding volumes and tool paths serve as a direct interface between the geometry specialist and the fixturing specialist.

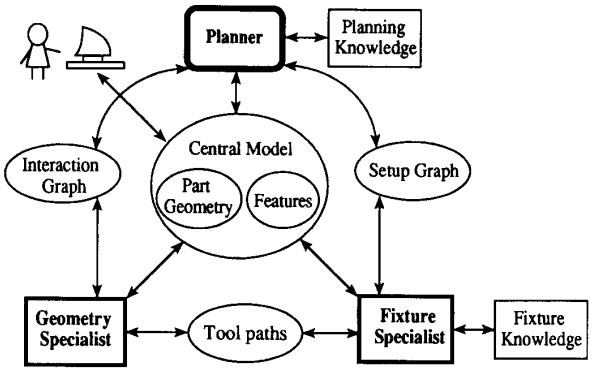
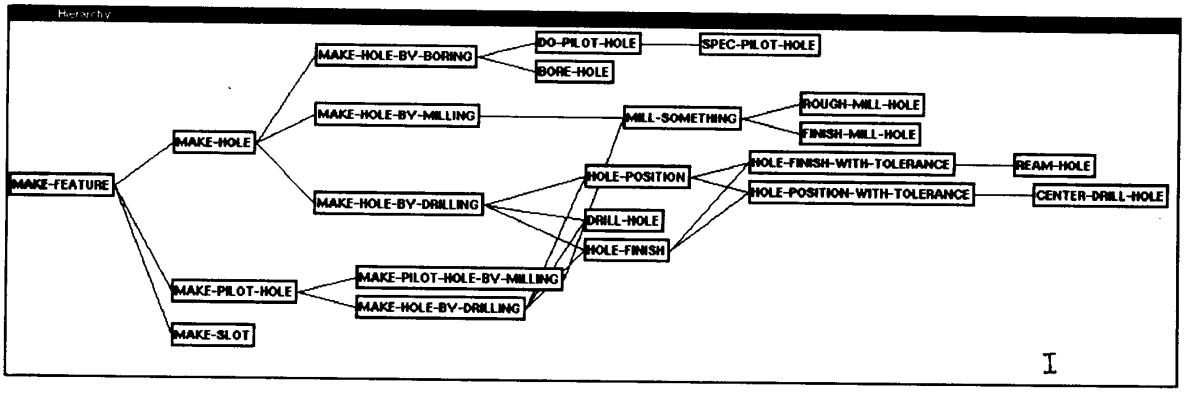


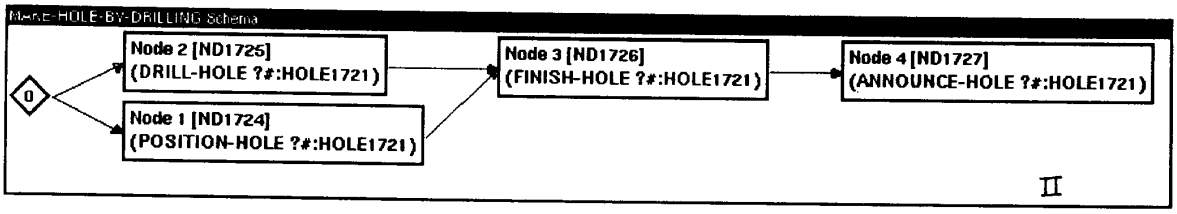
Fig. 2. Schematic diagram of the planning architecture in NEXT-CUT.

A. Planner

The basic planning paradigm that we use is that of nonlinear hierarchical planning [25], [27]. In this paradigm, the planning tasks involve the satisfaction of a conjunction of goals and the planning process consists of successively refining the planning tasks with the help of a set of a prespecified task-reduction schemata. The reduction schemata consist of plan fragments for achieving various goals. As an example, Window I in Fig. 3 shows the various schemata for machining holes, while Window II shows an individual task-reduction schema, MAKE-HOLE-BY-DRILLING in detail. As can be seen from Window II, the MAKE-HOLE-BY-DRILLING schema is a collection of

[2]Although we take a planner-centered view of the world in this paper, and consider fixture planning and geometric reasoning as specialists, it should be emphasized that within the NEXT-CUT framework, the machining planner is considered as another specialist at the same level as fixture planner and assembly planner.






```
{SCH1731}
   MAKE-HOLE-BY-DRILLING::(MAKE-HOLE ?HOLE1713)
      Expansion:
           0 {<0::ND1715>[:DUMMY]}
           1 {<1::ND1716>[:ACTION(POSITION-HOLE ?HOLE1713)]}
           2 {<2::ND1717>[:ACTION(DRILL-HOLE ?HOLE1713)]}
           3 {<3::ND1718>[:ACTION(FINISH-HOLE ?HOLE1713)]}
           4 {<4::ND1719>[:PRIMITIVE(:ANNOUNCE-HOLE :NAME ?HOLE1713)]}
      Conditions:
           <<SC1720>> :USE-WHEN (HOLE-SPEC ?HOLE1713) :at 1
           <<SC1721>> :USE-WHEN (SPEC (DIAMETER ?HOLE1713) ?DIAMETER) :at 1
           <<SC1722>> :USE-WHEN (TOOL ?TOOL) :at 1
           <<SC1723>> :USE-WHEN (EQUAL (TOOL-TYPE ?TOOL) :TWIST-DRILL) :at 1
           <<SC1724>> :USE-WHEN (EQUAL (DIAMETER ?TOOL) ?DIAMETER) :at 1
           <<SC1725>> :USE-WHEN (SPEC (BOTTOM-CONDITION ?HOLE1713) ?BOTTOM-CONDITION1714) :at 1
           <<SC1726>> :USE-WHEN (BOTTOM-DRILLABLE ?BOTTOM-CONDITION1714) :at 1
           <<SC1727>> :COMPUTE (SLB-CL::FIND-TOOL-HOLDER-INTERFERENCES (QUOTE ?HOLE1713) (QUOTE ?TOOL))
      Effects:
           <<SE1728>> :ASSERT (<= (POSITION-TOLERANCE ?HOLE1713) 0.004) :at 2
           <<SE1729>> :ASSERT (<= (DIAMETRAL-TOLERANCE ?HOLE1713) 0.005) :at 2
           <<SE1730>> :ASSERT (MAKE-HOLE ?HOLE1713) :at 4
      Vars: (?HOLE1713 ?BOTTOM-CONDITION1714)
                                                                    III
```

Fig. 3. Specification of machining operations as task reduction schemata.

partially ordered planning steps for machining a hole by drilling it. In particular, it specifies that the hole has to be positioned, a drilling operation has to be carried out, and finally the drilled hole should be improved as necessary (e.g., improving diametral tolerance, surface finish, etc.). Notice that this fragment does not provide details about how the position and diametral tolerances should be achieved; those details are left for subsequent reductions. It is in this sense that the schema specifies an abstract plan fragment. As shown in Window III, the internal representation of the schema also includes information about which conditions have to be satisfied, and which effects are asserted at each step. The conditions dictate to a large extent whether a particular schema is suitable for accomplishing a particular task. For example, if either the tolerance requirements of a hole are very high, or the hole happens to be of a nonstandard size, MAKE-HOLE-BY-DRILLING will not be a candidate for machining that hole.

A hierarchical plan can be formally characterized as a 3-tuple, $\mathcal{P}$: $\langle\langle T, O, \mathcal{V}\rangle, T^*, D\rangle$, where T is a set of plan

steps (tasks) with O defining a partial ordering over them; and T^* is the union of tasks in T and their ancestors with D defining a set of parent, child relations among the tasks of T^*. Planning consists of refining abstract planning tasks [such as (`Make-hole Hole-2`)] into concrete subtasks with the help of these task reduction schemata, until every task in the plan is "primitive" (i.e., the planner knows how to perform that task).[3] Fig. 4 shows how the (`Make-hole Hole-2`) task is refined, with the help of MAKE-HOLE-BY-DRILLING schema (in Fig. 3), into three subtasks (`Position-hole Hole-2`), (`Drill-Hole Hole-2`), and (`Finish-Hole Hole-2`), which, in turn, are reduced to more concrete subtasks. During this refinement process, any interactions between the newly introduced steps and the existing steps are resolved by posting ordering and binding constraints on the plan. As a classical hierarchical planner, the planner only detects the interactions that become evident in terms of clobbered preconditions. The partially ordered plan for machining the `cross-product` is shown in Fig. 6 (see Section IV for further discussion). The planning strategy is "least-commitment" in that the planner starts with the assumption that the various design goals can be achieved in *any* order and imposes ordering relations only to remove interactions or to satisfy constraints. Avoiding overcommitment in this way facilitates subsequent processing of the generated plan for satisfying optimality criteria (e.g., merging machining steps to reduce setups and tool changes) [11].

As pointed out in Section I, the planner needs the ability to modify its plans incrementally, both to promote efficient interactions with the specialists and to deal with user-imposed changes in the design of the part. Our planner supports incremental plan modification by maintaining the causal dependencies among the individual steps of a plan, and the decisions underlying the development of that plan, in a representation called a "validation structure." It utilizes the PRIAR modification framework [10]–[12] for carrying out the modification (see Section VI for details).

B. Specialists

The specialists in our framework either augment the specification of the problem as seen by the planner and detect interactions that the planner itself cannot detect, or utilize the generated plan to make their own further commitments. In our system, the geometry specialist (see below) is of the former type, while the fixturing specialist is of the latter. The analyses by the specialists impose implicit constraints on the plan developed by the planner (and vice versa). The interfaces—the interaction graph, and the setup graph—help the modules to keep track of these constraints.

1) Geometry Specialist: The geometry specialist in the NEXT-CUT environment uses solid models of the part and features to detect a variety of geometric interactions that may affect the machining or fixturing of parts. Examples of such interactions include interferences between the tool paths for machining a feature, and the volumes of other features (or the part itself). In the case of the `cross-product` shown in Fig. 1, the tool access path for machining `hole-4` (shown by the shaded arrow `d3` in the figure) interferes with the feature volume of `slot-1`. Window I in Fig. 5 shows the geometry specialist's description of the interference detected in this case. Such interactions are ubiquitous in machining and are, therefore, computed with every design or plan change. Since the exact details of the tool paths are not yet known,[4] and also since exact volume intersections can be time consuming, our geometry specialist uses conservative rectangular bounding box approximations of the material that a tool could remove from the part, and of the total volume swept out by a tool [19].

Once such interferences are detected, appropriate actions must be taken to resolve them (if possible). The geometry specialist does this by analyzing the interferences. In particular, suppose an interference $\mathcal{I}_{fd}$ is detected for the tool approach direction d of feature f. The geometry specialist checks to see if the volume of the detected interference $\mathcal{I}_{fd}$ is wholly subsumed by the volumes of some subset $\mathcal{F} = \{f_i\}$ of other features of the part. If this is the case, then the interference $\mathcal{I}_{fd}$ can be avoided by machining the features in $\mathcal{F}$ first (if no such set $\mathcal{F}$ is found, then the feature f cannot be made in tool approach direction d). This essentially imposes a set of constraints O_{fd} on the machining order of the individual features: $O_{fd} = \{(f_i < f) | f_i \in \mathcal{F}\}$[5].

In the case of interference between `hole-4` and `slot-1`, the analysis by the geometry specialist shows that the interference between the part, and the tool path for making `hole-4` in the direction `d3` is completely subsumed by the feature volume of `slot-1`. Thus, this interaction can be avoided by machining `slot-1` *before* machining `hole-2` if `hole-2` is to be made in the direction `d1`.

In this fashion, the geometry specialist detects the interactions for each feature and each possible tool approach direction for making that feature, and computes the appropriate ordering relations for avoiding those interactions. Once this is done, the geometry specialist heuristically selects a single tool approach direction for each feature (based on such criteria as the number of geometric interactions to be resolved in that direction) and

[3]Sometimes a task does not require further reduction because all of its effects already hold in the current situation (in planning terminology [22], such tasks are called *phantom goals*). For example, in the plan for machining `cross-product`, shown in Fig. 6, the finishing step was not required for HOLE-2, since the specified diametral tolerance for HOLE-2 is guaranteed by the drilling step itself. Thus the (DIAMETRAL-TOLERANCE HOLE-2) step is *phantomized* (shown with dashed lines in the figure) and does not constitute a step to be executed in the final plan.

[4]The detailed geometry of tool path depends on the exact tool that is selected for machining the feature, which will only be known after the machining planning is over.

[5]The symbol "$<$" is used to denote precedence relation between two entities. Thus the expression $a < b$ means a should precede b.

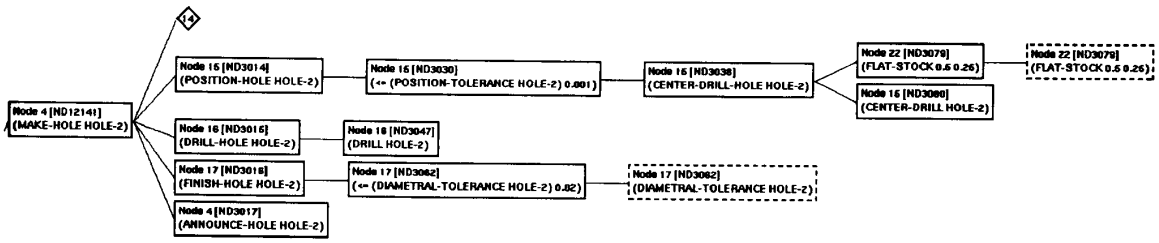


Fig. 4. Reducing an abstract task into concrete subtasks.

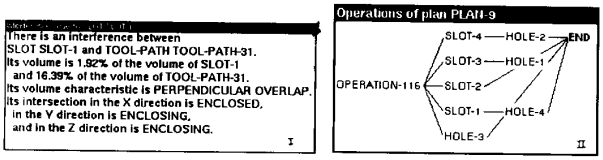


Fig. 5. Detecting and resolving geometric interactions for the `cross-product`.

conveys the corresponding feature orderings to the planner by constructing (or updating) the interaction graph (see Section IV).[6] Window II in Fig. 5 shows the interaction graph corresponding to the `cross-product` (note the ordering relation between `slot-1` and `hole-2`).

2) Fixturing Specialist: The objective of the fixturing specialist is to decide which operations of the plan will be done in which setup, and to arrive at fixture arrangements for locating and restraining the part as it is machined. The windows F1–F4 in Fig. 7, show a fixturing plan for manufacturing `cross-product`. An important consideration here is to reduce the number of setups. The operation of the fixturing specialist can be seen as having two phases, with the first phase consisting of proposing setups and the second phase consisting of testing them, employing geometric, kinematic, and force calculations. To reduce the number of setups, the fixturing specialist merges the steps of the machining plan based on the expected orientation of the part (given by the tool approach direction selected for that feature by the geometry specialist; see above) during those steps. In the second phase, it checks if the part can actually be fixtured in the proposed setups, and selects fixture elements for restraining the part during machining. This involves selecting a particular sequence (total ordering[7]) of the proposed setups (consistent with the ordering constraints among plan steps that comprise the setup groups), and ensuring that the geometry of the workpiece at the start of each setup allows it to be fixtured satisfactorily. The specific sequence of fixturing groups that are tested by the fixturing specialist then constitutes the fixturing plan. A constraint graph called the "setup graph," which contains information about the chosen setup groupings, and the ordering relations among them, acts as the interface between the fixturing specialist and the planner (see Section IV).

[6]Thus, the orderings imposed by the geometry specialist are conditional on the tool approach directions chosen, in the sense that if at a later point, the fixturing specialist decides to make a feature in a different orientation, then the ordering in the interaction graph would change. For a more detailed description, see [19].

[7]The need to ground the fixturing checks relative to the particular (intermediate) geometry of the part, and the difficulty of generating and maintaining partial geometries, are the main reasons why the fixturing specialist is forced to select a specific total ordering.

IV. The Planning Cycle

In this section, we discuss how the planner and the specialists interact through the interfaces to produce and revise plans. Table I shows a high level description of the planning cycle.

When the specification of a part, such as that of `cross-product` as shown in Fig. 1, is entered for the first time, the geometry specialist computes the possible geometric interactions between its features (as shown by the example in Window I of Fig. 5). Specific ordering constraints to avoid these interactions are then conveyed to the planner via the interaction graph (Window II).

Given the plan representation discussed in Section III-A, the *interaction graph* can be seen as an augmentation to the top-level specification of the problem. In particular, the interaction graph can be represented by a directed acyclic graph (DAG) $\mathcal{G}$: $\langle F, O_g \rangle$ whose nodes are the individual features of the part and whose edges define a partial ordering on the machining of different features. From the discussion in Section III-B, we can see that $O_g = \cup_f O_{fd}$, where d is the chosen tool approach direction for feature f, and O_{fd} is the set of precedence constraints imposed by the geometry specialist to resolve any tool path interferences in machining f in direction d.

The effect of the analysis by the geometry specialist is that instead of starting with unordered goals, the planner orders them according to the restrictions imposed by the interaction graph. In particular, the planner starts with an initial task network $\langle T', O' \rangle$, with T' containing the set of tasks of the form t_i: *Achieve*(*feature*$_i$), and orderings of type $[t_i$: *Achieve*(*feature*$_i$)$] \prec_{O'} [t_j$: *Achieve*(*feature*$_j$)$]$, if and only if *feature*$_i \prec_{O_g}$ *feature*$_j$. The final plan thus incorporates the orderings imposed by the planner, as well as those inherited from the interaction graph.

The machining plan for `cross-product` is shown in Fig. 6. (The diamond-shaped steps are dummy steps, and steps with dashed boundaries correspond to "phantom" steps, i.e., steps whose intended effects are made true by other steps). Note, in particular, that the

TABLE I
HIGH LEVEL DESCRIPTION OF THE PLANNING CYCLE

Given a new or changed specification:
1) **Geometry Specialist:** (*Input:* The solid model of the part and the features)
   Compute geometric interferences and update interaction graph
2) **Planner:** (*Input:* Feature specification, interaction graph, setup graph)
   (a) If no machining plan exists, generate one using the feature specification and the interaction graph. If there are any tool-holder collisions, backtrack to the geometry specialist (see Section V).
   (b) If a machining plan exists, modify it to accommodate the new specifications (changes in feature attributes, interaction graph or setup graph), while respecting any implicit constraints imposed by the setup graph and the interaction graph (see Section VI).
3) **Fixturing Specialist:** (*Input:* Machining plan, feature geometry, and setup graph)
   (a) If a fixturing plan does not exist, construct the setup graph by merging steps of the machining plan. Select a setup sequence and compute the fixturing details for it. If no total ordering is found, backtrack to the planner (see Section V).
   (b) If a fixturing plan does exist, update the setup graph to reflect changes (if any) in the machining plan. Use it to incrementally revise the existing fixturing plan. Update the setup graph.

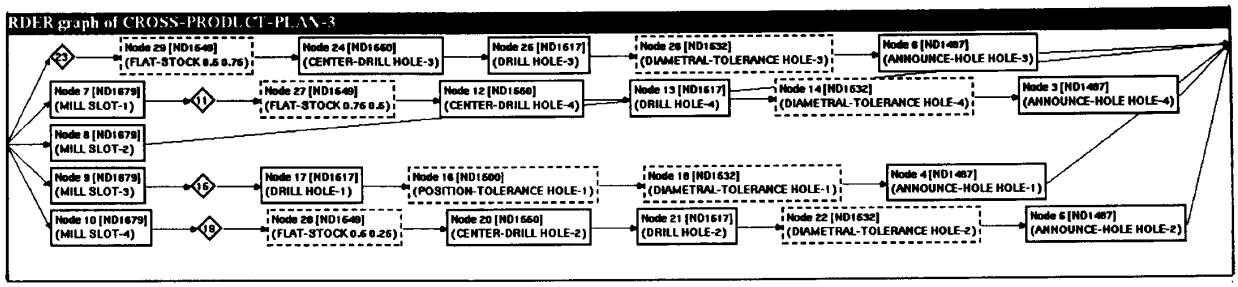


Fig. 6. Machining plan for the `cross-product`.

machining steps for `slot-1` and `hole-4` (in the lowest branch of the plan in Fig. 6) are ordered according to the constraints specified by the interaction graph (Window II in Fig. 5).

Next, based on this plan, the fixturing specialist chooses setups for fixturing. From the planner's viewpoint, the fixturing specialist is partitioning the plan steps into groups, based on a set of equivalence classes defined in terms of the expected orientations of the part during plan execution. Such a partitioning induces an implicit partial ordering among the setups. As discussed in Section III-B, this partitioning is followed by checks to ensure that some total order of setups consistent with this partial ordering can actually be fixtured.

The *setup graph* can thus be seen as a DAG S: $\langle \Omega, O_s \rangle$, where each member $\omega \in \Omega$ is a set of plan steps that can be machined in a particular setup, and O_s is a partial ordering on the setups, induced by the corresponding partial ordering on the plan steps.

The constraints on the setup graph from the planner's viewpoint are that Ω be a set of mutually exclusive and exhaustive subsets of tasks in T, so that the partitioning is consistent with the partial ordering among the tasks. To ensure the latter, the following two constraints must be satisfied:

1) $\forall \omega \in \Omega$, $\forall t_1, t_2 \in \omega$ $\nexists t \in T$ s.t. $t \notin \omega \wedge (t_1 \prec t \prec t_2)$.
2) $\forall \omega_1, \omega_2 \in \Omega$, if there exists a task $t_1 \in \omega_1$, and $t_2 \in \omega_2$, such that $t_1 \prec t_2$ in the plan, then it should necessarily be the case that $\omega_1 \prec_{O_s} \omega_2$.

O_s thus defines the partial ordering induced among the setups as a result of merging the steps of the plan.

For the `cross-product` example, Window II-A in Fig. 7 shows the setup group mergings computed, and Window II-B shows the description of the individual plan steps merged under each setup group. Notice that the graph is partially ordered at this point.

From the point of view of the fixturing specialist, each $\omega \in \Omega$ is a fixturing group. In general, once the fixturing specialist makes a merging of the plan steps according to the above constraints, there is an implicit partial ordering among the fixturing groups [as stated in condition 2) above]. From the standpoint of fixturing, this merging is consistent as long as the fixturing specialist can find a sequence of the setup groups consistent with this partial ordering, which satisfies the fixturing constraints (see Section III-B). To this end, the fixturing specialist first selects

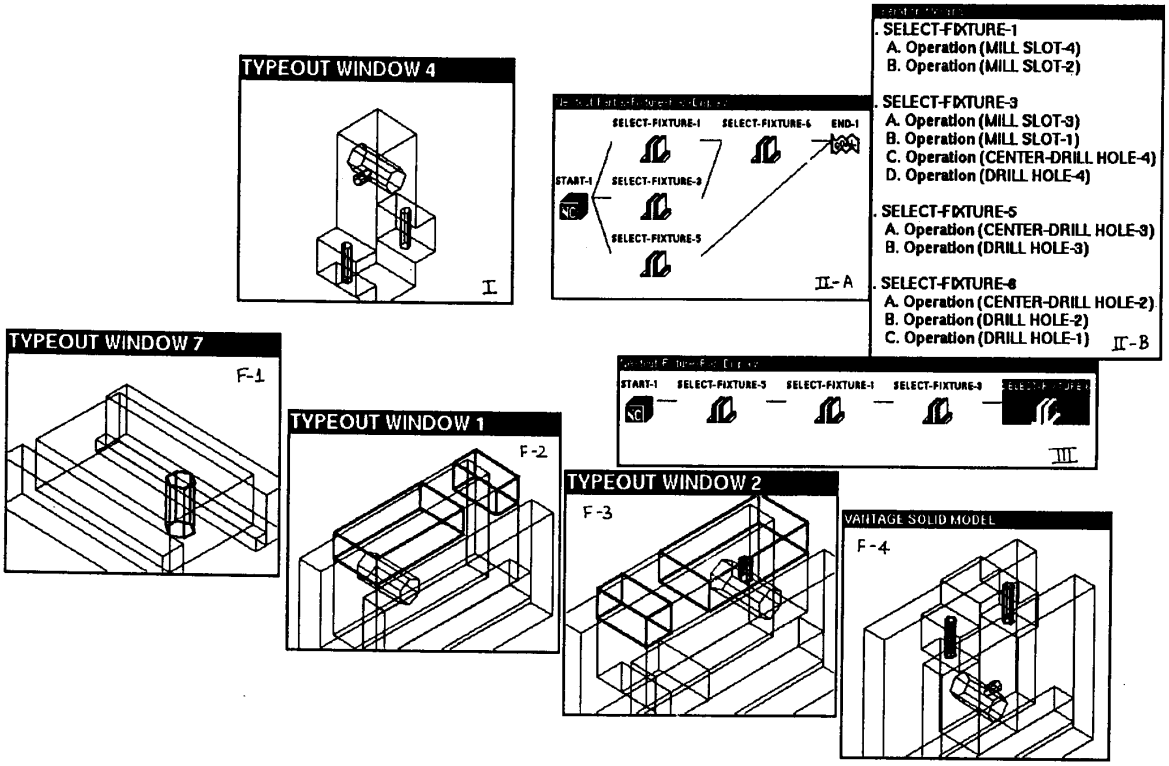


Fig. 7. Fixturing plan for the `cross-product`.

a total order on $\mathcal{S}$ based on some heuristic considerations [19], and then carries out fixturing analysis in accordance with that sequence.[8] Once a totally ordered sequence of setups is selected, that further constrains the orderings among the steps of the machining plan implicitly. In particular, selecting a total order on a setup graph $\mathcal{S}$: $\langle \Omega, O_s \rangle$ is equivalent to adding a set of additional ordering relations O_F among the setups in Ω such that $O_s \cup O_F$ induces a total order on $\mathcal{S}$. Every new ordering, $\omega_i \prec_{O_F} \omega_j$, among setups, translates to additional orderings among plan steps such that $\forall t_i \in \omega_i$ and $\forall t_j \in \omega_j$, $t_i \prec t_j$ (even if t_i and t_j do not have any ordering relations imposed among them by the geometry specialist or the planner). Such implicit constraints have to be respected to ensure conservatism of any future plan revision (see Section VI).

For the `cross-product` example, the fixturing specialist selects one total ordering (shown in Window III in Fig. 7) consistent with this graph that is satisfactory from the fixturing viewpoint, and computes a fixturing plan (in each fixture setup, the features to be manufactured in that setup are shown highlighted). It then updates the setup graph with additional orderings corresponding to the selected sequence. The Windows F-1–F-4 in Fig. 7 show the details of the fixturing plan. At this point, we have a complete process plan for machining `cross-product` (see Section III).

[8]Note that different setup sequences have differing fixturing properties as they correspond to different intermediate geometries of the part during machining.

V. Intermodule Backtracking

When inconsistencies arise between the commitments made by the planner and the specialists, the linear control flow discussed in Section IV is disrupted, and backtracking is necessitated. When this happens, there are, in general, a variety of backtracking alternatives, some intramodule, and some intermodule, each presenting a different set of tradeoffs. In this section, we will concentrate on the issues involved in intermodule backtracking, which is guided by the interfaces between the planner and the specialists.

Consider the interaction between the geometry specialist and the planner (as described in the previous section). Sometimes, the tool approach directions chosen by the geometry specialist for individual features (see Section III-B) may not be acceptable to the process planner. This could cause the planner to backtrack to the geometry specialist. In the `cross-product` example, suppose the geometry specialist selected a different set of tool directions that would involve making `hole-1` and `hole-2` from above (i.e., choosing `d1` as the tool ap-

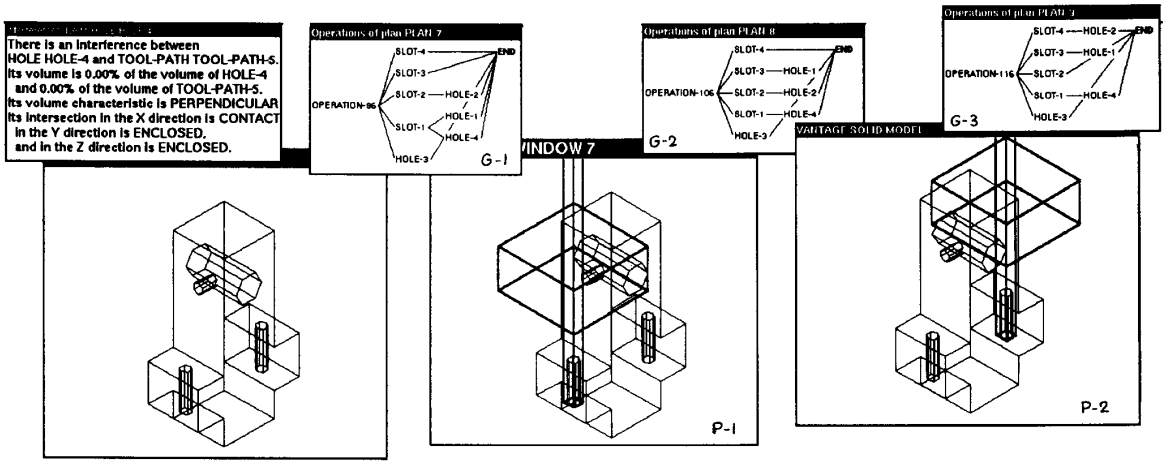


Fig. 8. Backtracking between the planner and the geometry specialist.

proach direction for `hole-1`; see Window I in Fig. 1). Window G-1 in Fig. 8 shows the corresponding interaction graph. When the planner starts with this interaction graph, it finds that `hole-1` cannot be made in the direction `d1` because of the possibility of tool holder colliding with the part[9] (see Window P-1 in Fig. 8). At this point, the planner backtracks to the geometry specialist. The geometry specialist typically has two alternatives. If the tool holder interference can be avoided by imposing additional orderings among the features (see Section III-B), then it updates the interaction graph with those additional orderings and restarts planning. If this is not possible, then it prunes the problematic tool approach direction for the feature in question, and selects a new tool approach direction for the feature. Once this is done, the interaction graph is updated with the orderings corresponding to the new tool approach direction and the planner is restarted.

In our example, the geometry specialist finds that there is no way of avoiding the tool holder collision if `hole-1` is made in direction `d1`. Thus, it prunes `d1` from consideration, and selects `d2` as the new approach direction for `hole-1`. Window G-2 in Fig. 8 shows the new interaction graph. Note that in this graph, the ordering between `slot-1` and `hole-1` is replaced by the one between `slot-3` and `hole-1`. The planner continues, and finds that there will be a similar tool holder collision if `hole-2` is made from above (Window P-2 in Fig. 8). Another backtracking takes place and the geometry specialist suggests making `hole-2` from below. The updated interaction graph is shown in Window G-3 of Fig. 8. This is the same as the interaction graph shown in Fig. 5, and consequently, the planner will be able to succeed and produce the machining plan shown in Fig. 6.

Another interesting case of intermodule backtracking arises when the fixturing specialist fails to find a fixturing arrangement to accommodate all the machining steps in a particular merged group ω in the setup graph. In such a situation, it will first try splitting ω into two or more setups $\{\omega_1, \cdots, \omega_m\}$ so that these groups can be fixtured individually. Note that splitting an existing setup group this way will not necessitate any revision in the machining plan (in particular the constraints 1) and 2) on setup graph, discussed in Section IV, are not violated).

Sometimes, however, there may be a particular machining step which cannot be made in the chosen orientation without running into fixturing difficulties. In such cases, there are two options: The first is to try an alternative tool approach direction for the feature associated with that machining step, and merge the operation for that feature with some other steps in the plan. As discussed before, changing the orientation this way may cause new geometric interactions and may indirectly impose new ordering relations on the machining plan by changing the interaction graph. (For example, if the fixturing specialist decides to make `hole-1` in Fig. 1 in direction `d1` instead of `d2`, then the geometry specialist will detect a new interaction between `hole-1` and `slot-1`.) The second option is for the fixturing specialist to test a different total order on the setup graph (see Section IV), so that the machining steps corresponding to the problematic feature appear earlier or later in the sequence (the part geometry will be different when the feature is made).

In the first option, since there may be new interactions between the features, the machining plan would need to be revised, taking into account any updates in the interaction graph. In comparison, the second option involves

[9]Tool holder collisions cannot be detected by the geometry specialist before the machining plan is generated, since the information about the particular tool that is selected to make the feature is not available at that time. During planning, once a tool is chosen, the planner tests for tool holder collisions by sending messages to the geometry specialist to check for the possibility of interference between the chosen tool and the part. The test itself is encoded as a "compute condition" (c.f. NONLIN [25]) on the appropriate task reduction schema (see Window III of Fig. 3).

only additional fixturing analyses. The tradeoff is not however as straightforward as this—analyses by the fixturing specialist are typically more time consuming than any incremental analysis by the planner. So, currently our system prefers the first option (even though it causes intermodule backtracking). To deal with such tradeoffs in a more domain-independent fashion, the modules need to have some idea about the cost of violating individual constraints. In Section VII, we propose a possible solution for this.

VI. Intramodule Backtracking and Incremental Plan Revision

We have seen that inconsistent commitments by specialists may necessitate intermodule backtracking, which is often costly. To contain this, and to improve efficiency of the overall planning, it is important for individual modules to have the ability to accommodate changes in their specifications by incrementally modifying their plans. Similar revision is also necessitated in response to designer initiated specification changes. In both cases, the revision needs to be conservative both to ensure internal efficiency of planning, as well as to contain the ripple effects of changes in the plan on the analyses of other modules. Furthermore, to improve the overall efficiency, the planner's ability to reuse its plans will have to be supplemented by the specialists' ability to reuse their previous analyses. In our implementation, both the planner and the fixturing specialist have the ability to reuse previous results. While each module maintains the internal dependencies on its plans, the external (intermodule) dependencies are maintained through the interfaces.

The planner uses the PRIAR modification framework [10], [12] to carry out plan revision. The modification is guided by a representation of internal dependency structure of the plan called *validation structure*. Validations can be seen as 4-tuples v: $\langle E, t_s, C, t_d \rangle$, with the semantics that the condition E, which is an effect of t_s, should be held true form task t_s to t_d, to support the applicability condition C of task t_d. The validation structure is used as a basis to justify individual planning decisions (steps, ordering constraints, and task reductions) of the plan. In particular, we justify validations in terms of the overall goals of the plan, and then justify the other planning decisions in terms of the validations they support [12], [14]. Such a justification structure allows the planner to locate parts of the plan that become superfluous whenever a particular retraction occurs.

Given this justification framework, incremental revision is seen as the process of repairing the validations that are found to fail due to specification changes or externally imposed constraints (e.g., increasing the tolerance specification or the diameter of a hole). The repair actions depend upon the nature of the failing validations. If the failing validation can be achieved by a new task, then the new task is added to the plan. On the other hand, if the validation is not achievable, then a dependency-directed backtracking mechanism is invoked to undo the corresponding planning decisions which caused it to fail. By the end of this repair process, the inapplicable parts of the plan will have been pruned away and some new, nonprimitive tasks will have been specified to the planner. These tasks are then reduced to primitive tasks (by the methods described in Section III-A) to give rise to a complete plan [see [10], [12] for details].

The fact that the planner is part of a hybrid architecture leads to some additional complexities in plan revision, which we will discuss presently. To begin with, in our hybrid planning architecture, the plan contains both the constraints which were imposed by the internal interaction-resolution routines, and those imposed by the specialists (through interfaces). At the end of a normal planning cycle (discussed above), there are three types of ordering constraints among the steps of the plan:

1) Orderings inherited from constraints imposed by the geometry specialist. In the example that we are following (see Fig. 6), the ordering `(mill slot-1)` $\prec$ `(drill hole-4)` is of this type (as it is inherited from the machining ordering constraint `slot-1` $\prec_{O_g}$ `hole-4` imposed by the geometry specialist).

2) Orderings imposed by the planner during the planning (i.e., $t_i \prec_O t_j$) [e.g. `(center-drill hole-2)` $\prec$ `(drill hole-2)` in the example].

3) Orderings imposed by the fixture specialist when it chooses a total order over the setup graph [i.e., t_i belongs to the setup ω_i and t_j belongs to the setup ω_j, such that $\omega_i \prec_{O_F} \omega_j$]. For example, `(drill-hole1)` $\prec$ `(mill slot-1)` in the example [since the step `(drill hole-1)` is included in setup `select-fixture-4` in the fixture plan which precedes the setup `select-fixture-2` that includes `(mill slot-1)`].

Since we are no longer concerned solely with the internal consistency of the revised plan (as in [10], [12]), but with the global consistency, both the planner and the specialists must be satisfied with the revised plan. In particular, to avoid costly ripple effects, the planner must keep track of any implicit constraints imposed by the specialists, through the interfaces, and respect them during any plan revision.

Unlike internal constraints, external constraints cannot be justified by the plan validation structure. For example, validation structure based justifications can only be provided for the ordering constraints of type 1), and not for those of types 2) and 3). Thus, during plan revision, the planner is only capable of reasoning about the ramifications of violating the orderings of type 2). Violating the other two types would lead to intermodule backtracking. In particular, violating orderings of type 1) may lead to geometric interactions, while violating orderings of type

2) may lead to costly refixturing analyses. To avoid these difficulties, the planner currently considers the externally imposed orderings to be nonnegotiable, and attempts to accommodate them by changing the plan first. (This may, however, sometimes adversely affect the flexibility of modification; see Section VII.)

Revising plans to accommodate externally imposed constraints in this way may in turn affect the analyses of the specialists. For example, whenever the machining plan is revised, the fixturing specialist, which makes its commitments based on the machining plan, needs to recheck the fixturing decisions. As in the case of the planner, this replanning is facilitated by keeping track of the dependencies between the fixture plan, the process plan, and the part geometry.[10] The dependencies between the fixture plan and the machining plan are maintained through the setup graph. Using this information, the fixture agent can figure out which setups are potentially affected by the changes in the machining plan, and thus need to be reconsidered. Fig. 9 shows the type of dependency information that the fixturing specialist can construct, based on the setup graph and the part geometry. In this example, features A, B, and C are composed of primitive geometric elements (surfaces and edges) FA-1, FB-3, etc., and are created through machining operations OA1, etc. The fixture agent has merged operations OA1, OA2, and OA3 and created a fixture arrangement, SETUP-1 for them. Similarly, OB1, OC1, and OC2 have an associated fixture arrangement, SETUP-2. Let us suppose that feature A is modified. The process planner reevaluates operations OA1–OA3 and modifies them as necessary and the fixture agent reevaluates SETUP-1. SETUP-2 might seem to be unaffected, but it must also be reevaluated if there is an "indexing dependency" between it and one of the elements of feature A (e.g., a face constituting feature A was used to fixture the part in the vise in SETUP-1). Once it has been determined which setups must be reevaluated, the fixture agent checks the clampable regions of each and modifies the fixture arrangements as necessary. Further details about fixture replanning can be found in [19].

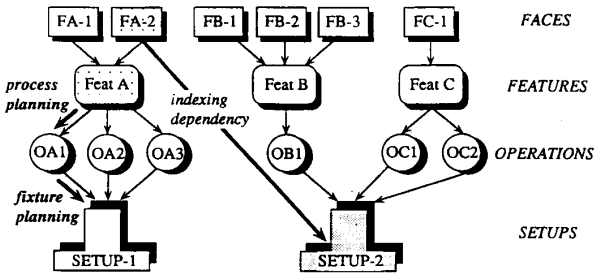


Fig. 9. Dependencies among features, operations, and fixture setups and primitive geometric elements of features.

A. Example

In the `cross-product` example, suppose the designer changes the specification of the part, moving the set-screw hole from the side to the top of the part (i.e., removing `hole-4` and adding `hole-5`), increasing the diameter of the middle hole, `hole-3`, and tightening the position tolerance of `hole-1` (as shown in Window I in Fig. 10). This change has an effect on the interactions detected by the geometry specialist. The updated interaction graph is shown in Window II of Fig. 10. The updated interaction graph, and the new specification of the part features become inputs to the planner. Since a machining plan already exists, the planner uses its incremental modification capability (see above) to accommodate these new specifications into the existing plan. Window III in Fig. 10 shows the revised plan that the planner produces by accommodating this change. The black nodes in the figure represent the parts of the original plan (shown in Fig. 6), while the white ones correspond to the newly added parts. Notice, in particular, that the increased diameter of `hole-3` makes it unsuitable for drilling, and it is machined by milling instead. Similarly, the tightened position tolerance of `hole-1` necessitates the addition of a center drilling step to its machining operations. Fig. 11 shows how the hierarchical structure of the plan is utilized in localizing modification to just those parts of the subplan that are affected by a changed specification. In this case, the change in position tolerance requirements affects only the `hole-position` operation of `hole-1`. The rest of the hierarchical plan for making `hole-1` remains unchanged. Windows IV-A and IV-B show the new setup graphs corresponding to the changed plan. In this case, by comparing the previous fixture plan (shown in Fig. 7) with the new setup graph, the fixturing specialist realizes that the changes imposed by the revised machining plan necessitate a new setup for the machining of `hole-5`. In addition, the previous setups that dealt with the machining steps of `hole-5` (which is now deleted) as well as those of `hole-3` and `hole-1` whose specifications have been changed, are potentially suspect. Window V shows the fixture plan which corresponds to a particular total ordering of the partial ordering shown in Window IV-A. Once again, the black nodes represent the parts of the fixture plan that are salvaged from the original plan, while the white ones represent the results of new analysis. Note that the only fixturing setup that is completely new is the one corresponding to `hole-5` shown in Window VI. The incremental operation of the planner and the fixturing specialist contains the ripple effects of the specification change, and makes the effect of the specification changes directly manifest in the process plan.

[10]Note that the specification for the fixturing specialist consists of the (changed) description of the features (which is available in the central knowledge base) as well as the (changed) machining plan.

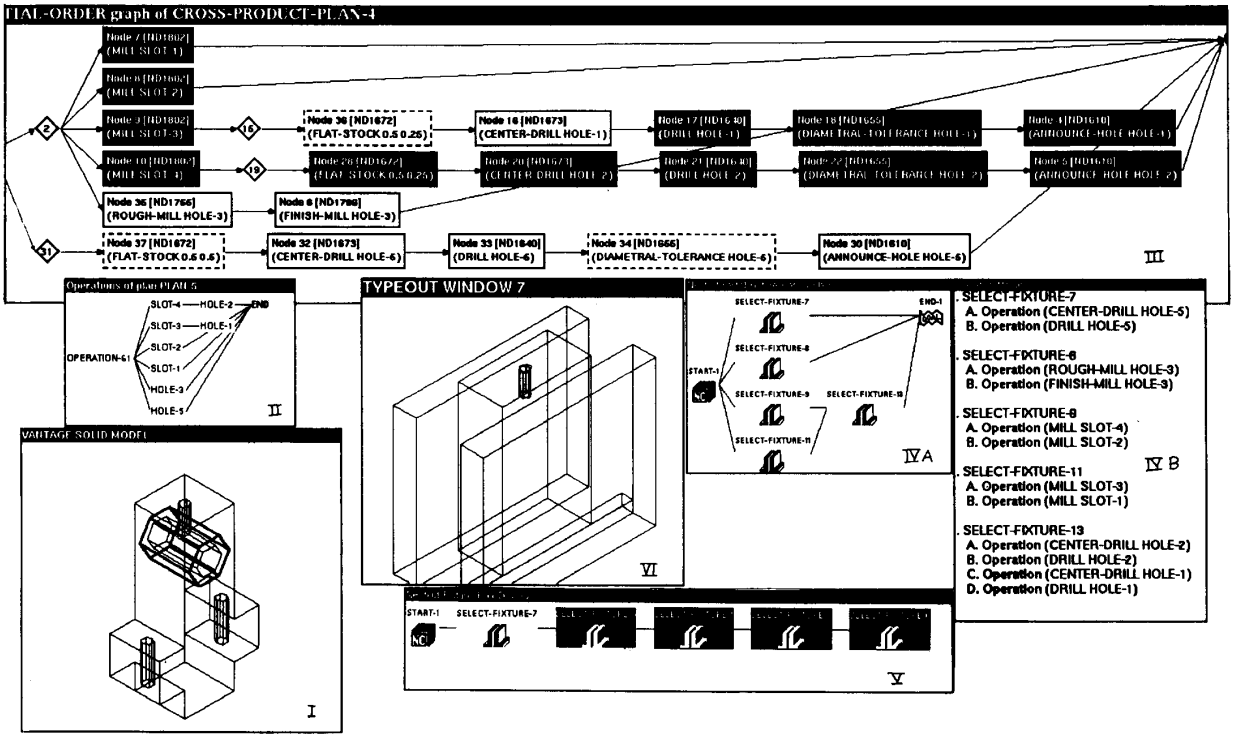


Fig. 10. Revising the plan in response to external constraints.

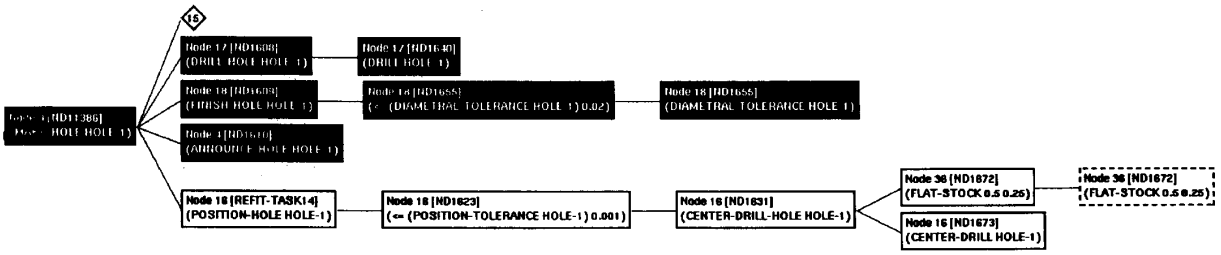


Fig. 11. A fragment of the hierarchical structure of the revised machining plan, showing how hierarchy localizes revision.

VII. Discussion

In the previous sections, we described our implementation of a hybrid planning architecture for process planning in Next-Cut. The architecture avoids duplicating capabilities of the specialists in the planner, thereby eliminating redundancy, and improving efficiency and modularity. Our implementation was automatically able to generate process plans that satisfy the constraints of geometry, machining, and fixturing specialists cooperating in an integrated framework. We have discussed the issues involved in integrating the general purpose planner with a set of specialists, and proposed ways to handle each of them. In particular, we focused on the interactions between the planner and two types of specialists, one which can be seen as augmenting the specification of the problem as seen by the planner, and another which utilizes the generated plan to make its own further commitments. We developed interfaces between the planner and the specialists that allow both to explicitly keep track of externally imposed constraints. In the case of the planner, all the external constraints have been modeled as additional orderings and mergings among machining steps. We have demonstrated that these interfaces allow the planner to function with a minimal understanding of the internal operations of the specialists, or the domain specific knowledge they employ. We have also discussed additional demands placed on the internal operation of the planner in such a hybrid planning environment. In particular, we described how the ability to incrementally modify existing plans to accommodate external constraints effectively controls the proliferation of secondary interactions, in the event inconsistent commitments between the planner and the specialists are detected.

In this section, we look at some of the limitations of our current model, and discuss the directions that we are exploring to overcome them. The discussion is divided into two parts, with the first part concentrating on the issues of planning in our model, and the second part concentrating on the architectural issues regarding interfaces between heterogeneous modules.

A. The Planning Methodology

Although the interfaces described in the previous sections allow the planner to keep track of the externally imposed constraints on the plan, they do not provide any indication of the reasons for constraints, or the cost (e.g., in terms of additional processing by the specialists) that would be incurred if those constraints are violated. At present, we get around this problem by assuming that the external constraints are *nonnegotiable* (see Section VI). However, such an assumption is too inflexible in that it may lead to excessive intermodule backtracking.

Consequently, we are exploring a framework in which the external constraints are accompanied with an explanation structure—a "window of applicability"—that provides a rationale for the constraint and the circumstances under which the constraint[11] remains valid (justified) [11], [13]. Such a framework will allow the planner to make educated decisions as to which constraints can be relaxed during the plan revision process. At the simplest (and least powerful) level, the explanations could include information such as whether the constraint is a hard one or a soft preference, provide a cost measure associated with violating the constraint, and/or attach a list of quantities upon which the justification for the particular constraint depends. Within the process planning domain, some of the fixture specialist's decisions are in response to "hard" fixturing constraints, while others are in response to "soft" preferences. The former affect fixture feasibility, while the latter affect optimality. Clearly, it would be useful to document this to help the planner in making decisions that might affect the fixture specialist.

A more powerful (and more difficult to construct) explanation may provide some sufficient and necessary conditions under which the constraint is justified. When justifications are attached to the externally imposed constraints, they need to be at a level of detail that is commensurate with the planner's model of the domain. There may be a spectrum of such justifications for any constraint, with tradeoffs between the detail of the justification and the ability of other modules to reason with it. The correct level of detail depends ultimately on the degree of similarity between the domain models and the inference strategies of the planner and the specialists. At one extreme, justifications might consist of precise descriptions of geometric interferences that caused some undesirable interactions. While such detailed justifications have been used to advantage in interfacing modules, such as geometry specialists and assembly planners that share similar domain models [28], they may not be suitable in interfacing a geometry specialist and a machining planner, which use significantly different domain models. At the other extreme, in complex environments with heterogeneous modules, even a justification that merely specifies the variables that the external constraint depends on, could be useful.

A related issue is the level of interfaces: In the current implementation, we modeled the specialist interfaces solely as imposing external ordering relations and merging constraints on the plan. Within the classical planning framework, we could also accommodate interfaces that augment the specification of the planning problem. For example, instead of doing the interaction resolution, the geometry specialist could provide a high-level description of the interference to the planner, and allow it to resolve the interactions itself. The task reduction schemas of the planner could then be written in such a way as to allow the planner to use the description of interferences to resolve interactions. This may sometimes provide a finer grained interaction between the specialist and the planner.

B. Architectural Issues

As described earlier, our hybrid planning architecture uses customized interfaces between the planner and the specialists. In light of the task-dependent nature of interfaces, a critical concern in generalizing this architecture is the effort involved in adding a new specialist to the architecture. Within NEXT-CUT's planning architecture, adding a new specialist is a two-stage process. First, each full-fledged specialist must at least be able to access the central model in the NEXT-CUT architecture (see Section III), and respond to any notification[12] messages sent to it [4]. Next, depending on the expected interactions between the planner and the specialist, a custom interface may have to be designed. The rationale here is that although modules need to exchange generic messages, to facilitate a deeper cooperation between the planner and the specialists, they also need to have customized interfaces that reflect the shared representations. The latter allows for efficient reasoning about interactions between the modules.

[11]The term "constraint" is used here in a general sense to include the ramifications of any type of commitment made by individual modules. In particular, window of applicability explanations could apply to *any* computed results which are being used by other modules (including the module which computed it).

[12]One characteristic of our architecture is that the overall model (state) of the problem solving remains distributed across specialists—the machining plans remain with the planner, the fixturing details remain with the fixturing specialist, and so on. Although this has several advantages (as we have discussed earlier), it also necessitates a more distributed approach to communications in the architecture. In our more recent work, we have started exploring ways of streamlining the communication aspects of the architecture substantially, using the notion of "notifiable agents" (c.f. [4], [21]). In this model, each module registers itself with a central communication module, giving information about the types of services it is capable of providing, and the set of external variables which it is dependent on. The former allows other modules to access information from this module, while the latter allows this module to be notified when variables that are of interest to it change.

Such interfaces need to be designed based on the overall characteristics of the task domain, and the types of constraints imposed by two modules on each other. Within the feature-based manufacturing domain, the interfaces will typically need to deal with geometric and precedence-based constraints, since geometry and precedence are typically the two main items of discourse in this domain. Suppose a tolerance reasoner is to be added to the architecture to help the planner take tolerance constraints into account. In this case, the tolerance reasoner requires the precedence information in the plan to propagate tolerances. The planner, on the other hand, needs the propagated tolerance information to prune infeasible plans.

Although the individual interfaces themselves are task dependent, some general principles can be gleaned from our experience with interfacing heterogeneous specialists to a planner. In designing interfaces between two specialists, we are interested in keeping track of the constraints that each module's actions impose on the other. The interfaces are kept at the highest level of granularity that can still facilitate this interaction. Consider for example, the interface between the planner and the geometry specialist. Based on our experience with successive generations of NEXT-CUT architecture, and related work by colleagues on assembly planning, grasp planning, and tolerance analysis [28], we have found that "bounding volumes" provide a useful abstraction of geometric interferences. For example, they help both the machining planner and the fixturing specialist without tying them down to the details of the geometric modeler.

VIII. RELATED WORK

The idea of integrating specialists into a general purpose reasoner to improve the efficiency and/or expressiveness has been studied previously in automated reasoning [7]. For example, Miller and Schubert [20] describe a reasoning system, that interfaces a general purpose theorem prover with a set of specialists to accelerate it. Here, typically, the general purpose reasoner already has a complete model of the reasoning carried out by the specialists. Brachman *et al.* [1] describe a reasoning system called KRYPTON which integrates a terminological and an assertional reasoning module by modifying the semantics of the unification operation. However, very little work along these lines has been done in planning. One exception is Simmons' GORDIUS system [23], which combines several specialized representations for reasoning about quantities, sets, diagrams, and time into a common framework to accurately and efficiently predict the effects of events. Simmons' work also points out the efficiency considerations involved in separating the plan generation from the deep causal models of the domain.

There is also some overlap between the issues of hybrid planning explored in this paper, and work in multiagent planning (e.g., localized search strategies [18]), distributed planning (e.g., coordinating a set of planners and combining their individual plans into a global solution [6]), and blackboard-based methods for integrating heterogeneous systems (e.g. [9]). The idea of decomposing a planning task into several specialized tasks, each performed by a different agent, has been espoused in Lansky's GEMPLAN [18]. In GEMPLAN, the individual agents all share common representations and inference strategies, which considerably simplifies the interface problem. Nevertheless, the localized search strategies developed in GEMPLAN appear generalizable to architectures involving heterogeneous specialists. Work in distributed planning such as [6] has typically addressed the issues of coordinating a set of homogeneous planners working on different subgoals of a single problem. It is assumed that the planners all share common representations and vocabulary. Similarly, blackboard-based integration methods typically assume that the individual modules are capable of computing the ramifications of the assertions on the blackboard on their problem-solving cycles. In contrast, we are explicitly concerned with the issues of integration between a general purpose planner and a set of heterogeneous specialists. Constructing interfaces to facilitate effective interaction between the planner and the specialists is, thus, of critical importance in our model.

The issues of combining heterogeneous specialists have received some attention in the distributed artificial intelligence community (e.g., [16], [17], [24]). The emphasis there has generally been on the negotiation and conflict resolution aspects of integration. Our work, in contrast, concentrates on the ramifications of heterogeneous hybrid planning architecture on the operation of the planner and specialists. In our current system, there is no explicit negotiation between the planner and the specialists—there is a strict hierarchy between the planner and specialists and each module treats the externally imposed constraints as hard nonnegotiable ones. Intermodule backtracking occurs only when these constraints cannot be incrementally incorporated by the module without violating some feasibility constraints. Although the lack of explicit negotiation has not been a significant problem in our current planner-centered architecture, we expect it will become more important as we generalize the architecture to allow multiple planners, interacting as peers, as well as with specialist modules.[13] The notion of *windows of applicability*, introduced in Section VII-A, could provide a general basis for negotiation in our architecture.

The importance of incremental revision in supporting distributed cooperative activity, as well as the intricacies of managing externally justified constraints, has also been well recognized in the DAI community. The DTMS work [2], for example, extends the truth maintenance to distributed agents. Unlike DTMS, which merely propagates the ramifications of a change across a distributed multiagent database, the incremental revision strategies discussed in VI explicitly attempt to guide the planner and the specialists in modifying their respective analyses to

[13]Indeed, the specialists themselves can be treated as specialized planners in such an architecture.

accommodate changes incrementally, and minimize the overall perturbation of plans. The technology of DTMS could, however, be gainfully incorporated into our architecture to provide more systematic justification maintenance and dependency propagation.

IX. Conclusions

In this paper, we have explored a hybrid incremental planning architecture which utilizes a set of specialists to complement both the overall expressiveness and reasoning power of a traditional hierarchical planner. We have described our implementation of this model in a manufacturing planning domain, and discussed several issues concerning the interfaces and interaction management between the planner and the specialists. The results of the implementation have been encouraging: Our architecture allowed effective interaction between the planner and the specialists, without binding the planner too tightly to the internal operations or the domain specific knowledge of the specialists. The incremental operation of the planner and the specialists was effective in controlling the proliferation of interactions when inconsistent commitments are detected between the planner and the specialists. The implementation has also taught us several general principles on designing hybrid planning systems; we summarize them briefly:

- Communication between the planner and the specialists takes several forms, including the shared representation of the design and process plan, specialized representations of mutual constraints (e.g., the setup graph) and standardized messages (e.g., the results of intersection tests from the geometry specialist). In all cases, there is tradeoff between expressiveness and abstraction. For example, the geometric intersection results, as in Window I of Fig. 5, were found after some experimentation to be at the right level of detail for making ordering decisions in process planning. More generally, it will be impossible to satisfy a variety of modules with messages and representations at a single level of detail. A solution to this problem may be to exploit hierarchical representations.
- Modules in a hybrid planning environment benefit from hierarchical representations and least commitment approach in problem solving, which keeps options open and reduces the need for backtracking in the face of specification changes and planning conflicts (by allowing maximum latitude to the specialists in refining the plan according to their constraints). In our implementation, for example, we maintain partially ordered machining plans, and setup graphs.[14]
- Each module should reuse previous results whenever practical, both for speed and to make the effects of design changes manifest. Reuse of previous results is particularly useful in managing the interactions between the planners and the specialists. Every time a module computes a new result, it is possible that it may invalidate results previously computed by other modules. However, to the extent that each module reuses previous results, the incidence of new side effects and interactions with other modules is reduced. Thus, if the process planner makes only minor changes to a previous process plan, it is unlikely that major changes will be needed in the corresponding fixture plan.
- The ability to reuse previous plans (and analysis results), as well as to control intermodule backtracking, hinges primarily on keeping track of dependencies within the plans and between the plans, the specifications, and the external constraints imposed by other modules. Interfaces which keep track of externally imposed constraints can thus play an important role in facilitating reuse. More generally, we found that it is important to keep issues of feasibility (constraints) separate from issues of optimality (costs), since the former are far more likely to remain valid from one plan iteration to the next.

Experience with our implementation makes us believe that hybrid architectures such as the one explored here offer a promising avenue of research for dealing with realistic planning domains. Our future work will involve extending the architecture in several directions as discussed in Section VIII.

[14]Of course, the usual tradeoff holds between the delay of commitment and the amount of computation needed to check the consistency of the plan. Thus, in the example of the fixturing specialist, to avoid extensive geometric simulation, a single total ordering of the setups is ultimately chosen for detailed fixture planning.

Acknowledgments

The authors wish to thank Andrew Philpot who helped in implementing the planner and the interfaces, Amy Lansky who provided several valuable criticisms on a previous draft, and the anonymous reviewers of AAAI-91 and IEEE SMC who made several helpful suggestions to improve the clarity of this manuscript.

References

[1] R. Brachman, V. Gilbert, and H. Levesque, "An essential hybrid reasoning system: Knowledge and symbol level accounts," in *Proc. 9th Int. J. Conf. Art. Intell.*, Aug. 1985.

[2] D. Bridgeland and M. Huhns, "Distributed truth maintenance," in *Proc. 8th Nat. Conf. Art. Intell.*, pp. 72–77, 1990.

[3] M. R. Cutkosky and J. M. Tenenbaum, "A methodology and computational framework for concurrent product and process design," *Mechanism and Machine Theory.*, vol. 23, no. 5, 1990.

[4] M. Cutkosky, R. Engelmore, R. Fikes, M. Genesereth, T. Gruber, W. Mark, J. Tenenbaum, and J. Weber, "Pact: An experiment in integrating multiple concurrent engineering frame-works," *IEEE Computer, Special Issue on Concurrent Engineering*, Jan. 1993.

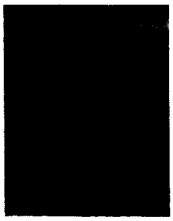
[5] Y. Descotte and J. C. Latombe, "Making compromises among antagonist constraints in a planner," *Art. Intell.*, vol. 27, pp. 183–217, 1985.

[6] E. Durfee and V. Lesser, "Predictability versus responsiveness: Coordinating problem solvers in dynamic domains," in *Proc. 7th Nat. Conf. Art. Intell.*, Aug. 1988.

[7] A. Frisch, "Report on the 1988 workshop on principles of hybrid reasoning," *AI Mag.*, vol. 11, Jan. 1991.
[8] C. Hayes, "Using goal interactions to guide planning," in *Proc. 6th Nat. Conf. Art. Intell.*, pp. 224-228, July 1987.
[9] B. Hayes-Roth, "Dynamic control planning in adaptive intelligent systems" in *Proc. DARPA Knowledge-Based Planning Workshop*, Dec. 1987.
[10] S. Kambhampati, "A theory of plan modification," in *Proc. 8th Nat. Conf. Art. Intell.*, Aug. 1990.
[11] S. Kambhampati and M. R. Cutkosky, "An approach toward incremental and interactive planning for concurrent product and process design," *Proc. ASME Winter Annual Meeting on Computer Based Approaches to Concurrent Engineering*, Nov. 1990.
[12] S. Kambhampati and J. Hendler, "A validation structure based theory of plan modification and reuse," *Art. Intell.*, vol. 55(2-3), pp. 193-258, June 1992.
[13] S. Kambhampati and J. Tenenbaum, "Planning in concurrent domains," in *DARPA 1990 Workshop on Innovative Approaches to Planning, Scheduling, and Control*, 1990.
[14] S. Kambhampati, "Characterizing multi-contributor causal structures for planning," in *Proc. 1st Int. Conf. AI Planning Systems*, 1992 (extended version to appear in *Artificial Intell.*, 1994).
[15] S. Kambhampati, M. Cutkosky, J. M. Tenenbaum, and S. Lee, "Combining specialized reasoners and general purpose planners: A case study," in *Proc. 9th AAAI*, 1991.
[16] M. Klein, "Supporting conflict resolution in cooperative design systems," in *Proc. 10th Workshop on Distributed Art. Intell.*, 1990.
[17] S. Lander, V. Lesser, and M. Connell, "Knowledge based conflict resolution for cooperation among expert agents," in *Computer-Aided Cooperative Product Development* (D. Sriram, R. Logher, and S. Fukuda, eds.). Berlin: Springer-Verlag, 1991.
[18] A. Lansky, "Localized event based reasoning for multiagent domains," *Comp. Intell. J.*, vol. 4, no. 4, 1988.
[19] S. H. Lee, M. R. Cutkosky, and S. Kambhampati, "Incremental and interactive geometric reasoning for fixture and process planning," in *Issues in Design/Manufacture Integration 1991* (A. Sharon, ed.), pp. 7-13, ASME Winter Annual Meeting, ASME, Dec. 1991.
[20] S. Miller and L. Schubert, "Using specialists to accelerate general reasoning," in *Proc. 7th Nat. Conf. Art. Intell.*, Aug. 1988.
[21] J. Pan and J. Tenenbaum, "Toward an intelligent agent framework for enterprise integration," in *Proc. 9th Nat. Conf. Art. Intell. (AAAI-91*, 1991.
[22] E. Sacerdoti, *A Structure for Plans and Behavior*. New York: Elsevier, 1977.
[23] R. Simmons and R. Davis, "Generate, test, and debug: Combining associational rules and causal models," in *Proc. 10th Int. J. Conf. Art. Intell.*, Aug. 1987.
[24] K. Sycara, "Resolving goal conflicts via negotiation," in *Proc. 7th Nat. Conf. Art. Intell.*, pp. 245-250, 1988.
[25] A. Tate, "Generating project networks," in *Proc. 5th Int. J. Conf. Art. Intell.*, p. 888-893, 1977.
[26] J. Tsang, "Propel: An expert system for generating process plans," in *Proc. SIGMAN Workshop on Manufacturing Planning* (IJCAI-89), Detroit, MI, 1989.
[27] D. Wilkins, "Domain independent planning: Representation and plan generation," *Art. Intell.*, vol. 22, pp. 269-301, 1984.
[28] R. Wilson and J.-F. Rit, "Maintaining geometric dependencies in an assembly planner," in *Proc. 1990 IEEE Int. Conf. on Robotics and Automation*, 1990.

Subbarao Kambhampati received the B.Tech. degree in electrical engineering (electronics) from Indian Institute of Technology, Madras, India, and the M.S. and Ph.D. degrees in computer science from the University of Maryland, College Park, MD, in 1985 and 1989, respectively.

From 1989 to 1991, he was a research associate with the Center for Design Research and Department of Computer Science at Stanford University, Stanford, CA. He is currently an Assistant Professor in the Department of Computer Science at Arizona State University, Tempe, AZ. His current research interests include automated planning, machine learning, analogy, and AI based methods for design and manufacturing.



Mark R. Cutkosky received the Ph.D degree from Carnegie-Mellon University, Pittsburgh, PA.

He was with the Robotics Institute at Carnegie-Mellon University, employed as a design engineer at ALCOA, Pittsburgh, PA. In 1985, he joined the Design Division, Department of Mechanical Engineering, Stanford University, Stanford, CA, where he is an Associate Professor.

Dr. Cutkosky has been active in the area of concurrent engineering since coming to Stanford and is the principal investigator of a research project on Concurrent Product and Process Design at Stanford's Center for Design Research. His other research areas include dextrous manipulation with robotic hands and tactile sensing. He is an NSF Presidential Young Investigator and the Anderson Faculty Scholar at Stanford. He is a member of ASME, SME, and Sigma Xi, and an Associate Technical Editor for the ASME Transactions: *Journal of Engineering for Industry*.

Jay M. Tenenbaum received the B.S. and M.S. degrees in electrical engineering from M.I.T. in 1964 and 1966, respectively, and the Ph.D. degree in electrical engineering and computer science from Stanford University in 1970.

From 1972 to 1980, he led to the research program in machine intelligence at SRI's Artificial Intelligence Center. In 1980, he co-founded the Fairchild Laboratory for Artificial Intelligence Research (FLAIR), a forerunner of Schlumberger Palo Alto Research, serving as its director from 1983 to 1986. In 1986, Dr. Tenenbaum returned to active research, as a Schlumberger Fellow and Professor of Computer Science (Consulting) at Stanford University. From 1988 to 1990, he also served as Director of Advanced Research Projects for Schlumberger Technologies. At Stanford, his research has focused on applications of AI in design and manufacturing. In January 1990, Dr. Tenenbaum established Enterprise Integration Technologies Corporation, an R&D and consulting organization that develops sophisticated information architectures for information sharing, decision coordination, workflow automation, and electronic commerce.

Soo-Hong Lee received the B.S. and M.S. degrees in mechanical engineering and mechanical design/system from Seoul National University, South Korea, in 1981 and 1983, respectively, and the Ph.D. degree in mechanical engineering, design division, from Stanford University, CA, in 1991.

He was a Research Associate at the CAD/CAM Lab, Korea Institute of Machinery and Metals from 1983 to 1985. Since January 1986, he has been at the Department of Mechanical Engineering, Stanford University, CA, where he is currently a consultant working on a concurrent development system for cable harnesses for Lockheed Missiles & Space Company, Inc. His research interests include manufacturing automation, automated flexible fixturing, concurrent product/process design, and design knowledge representation.

Dr. Lee is a member of the American Society of Mechanical Engineers.