Planning in Concurrent Domains

Subbarao Kambhampati[*]
Center for Design Research
and Computer Science Department
rao@cs.stanford.edu

Jay M. Tenenbaum
Center for Integrated Systems
and Computer Science Department
marty@cis.stanford.edu

Stanford University
Stanford, CA 94305-4026

Abstract

In many real-world situations, a planner is part of an integrated problem-solving environment and must operate concurrently with other planners and special purpose inference engines. Unlike the traditional AI planners, planners operating in such concurrent environments have to contend with an evolving problem specification, and should be able to interact and coordinate with the other modules on a continual basis. This in turn poses several critical requirements on the planning methodology. In this paper we identify the ability to incrementally accommodate the changes necessitated by the externally imposed constraints into the existing plans, and the ability to understand and reason about the rationale behind externally imposed constraints at an appropriate level of detail as two crucial requirements for planning in such environments. We then explore directions for extending classical hierarchical planning framework to handle those requirements.

1 Introduction

In many real-world situations, a planner is part of an integrated problem-solving environment and must operate concurrently with other planners and special purpose inference engines (collectively referred to as SDMS in this paper). The SDMS concentrate on different specialized considerations and interact with the planner to post various feasibility and optimality driven constraints on the evolving plan. Consider, for example, the problem of generating process plans in concurrent design environments [Cutkosky and Tenenbaum, 1990, Kambhampati and Tenenbaum, 1990, Kambhampati and Cutkosky, 1991].[1] A goal of concurrent design is to do as much manufacturing planning and analysis as possible during design evolution, rather than waiting for the design to be complete. A typical planner in this environment will have to operate concurrently with SDMS specializing in such considerations as geometric analysis, fixturing, tolerancing etc., all of which influence planning. Another example is the DARPA logistics planning scenario where several disparate specialized planners (such as evacuation planners, mission planners etc.) are required to cooperate with each other to make a global transportation plan.

Planning in these environments a continual rather than one-shot process as the constraints imposed by the SDMS on the plan force the planner to contend with a continually evolving problem specification. The planner needs to respond to the specification changes in a continual rather than batch driven mode (i.e., the planner cannot wait for all the changes to be complete) as the feedback from the planner often guides the behavior of the other modules in the environment. Further more, the planner needs to understand the rationale behind the externally imposed constraints so that it can play an active role of coordinating its actions with the SDMS .

The classical planning paradigm [Chapman, 1987] fails to adequately handle the requirements of these environments as it considers planning as a one-shot task of constructing a partially ordered sequence of actions for achieving a given set of goals. The planner works under the assumption that it is an isolated module with all knowledge relevant to plan generation at its disposal. This paradigm only accounts for the intra-plan interactions, and ignores the interactions between the planner and the other modules of a problem-solving environment.

To deal with the special requirements of planning in concurrent environments, the planning methodology needs to be extended to provide the following capabilities:

- *Incrementality and accommodating external constraints*
  The planner needs to respond to the changes in its specification and externally imposed constraints by updating its plan efficiently and conservatively. During this update process, it needs to be incremental and reuse as much of its existing plan as possible, as starting from scratch every time some constraint changes can be prohibitively expensive. Thus, to function effectively in a concurrent environment and interact with other modules efficiently, a planner needs to be *incremental* in its operation.
- *Coordination and understanding rationale behind externally imposed constraints*
  To anticipate external interactions and to coordinate its actions with other modules in the environment, a planner in a concurrent environment needs to be able to understand and reason about the rationale behind

[*]We acknowledge the support of Office of Naval Research under contract N00014-88-K-0620.

[1]A process plan specifies the sequence of setup, fixturing and machining operations for manufacturing a given part (typically described as a set of machinable features).

the externally imposed constraints. To facilitate this, a systematic interface methodology between the planner and the other heterogeneous modules of the environment needs to be developed.

Intuitively, the planner needs to be incremental in order even to survive as a passive module in a concurrent environment, while it needs the ability to coordinate its actions with the SDMS to play an active role in guiding the global problem-solving activity.

To facilitate incremental modification of plans in response to evolving specifications, the planner should be able to analyze the plan at multiple levels of detail and focus modification at the appropriate level. Further, it should use least-commitment strategies during planning, as over-constrained plans do not lend themselves well for flexible reuse [Kambhampati, 1989, Kambhampati, 1990a]. The criterion for accommodating external constraints is for the planner to be able to incorporate the changes imposed by those constraints while reusing as much of the existing plan as possible and preserving the correctness of the overall plan.[2] This requires the planner to reason about the effect of the external constraints on the correctness of the existing plan and make minimal modifications to it to regain correctness.[3] To analyze the effect of external constraint on the correctness of the plan, as well as to modify the plan conservatively, the planner needs to represent and reason about the internal dependencies of its plans in a systematic fashion.

Since, hierarchical nonlinear planning (e.g. NOAH [Sacerdoti, 1977], NONLIN [Tate, 1977], SIPE [Wilkins, 1984]) is the prominent method of abstraction and least commitment in domain independent planning, in this paper we will explore ways of extending it to handle the incrementality requirements of concurrent domains. In section 2, we will develop a precise characterization of the correctness of the plan in terms of a representation of its causal dependency structure called the *validation structure* and introduce a framework of incremental modification based on it. In section 3, we will present techniques to accommodate the external constraints which compute the ramifications of those constraints on the validation structure of the plan, and repair the plan to regain correctness.

Enabling the planner to reason about the rationale behind the constraints imposed by the SDMS poses special problems because of the heterogeneous nature of the modules in a concurrent environment. Previous research in distributed planning (e.g. [Durfee and Lesser, 1988, Corkill, 1979]), black board systems (e.g. [Hayes-Roth, 1987]), embedded systems (e.g. [Georgeff, 1990]) and multi-agent planning architectures (e.g. [Lansky, 1988]), mostly addressed the issues of coordinating the actions of homogeneous planners working on different aspects of a single problem where each planner can understand and reason about the rationale behind the constraints imposed by other planners. Thus they are not well-suited to real world concurrent domains with heterogeneous modules. In section 4, we propose a methodology of interfacing the planner and the SDMS that relaxes the strong assumptions made by these approaches. Our approach is to enable the planner to understand the rationale at some appropriate level of abstraction. Accordingly, the external constraints will be accompanied by explanations that constitute ''sufficient'' (rather than necessary and sufficient) conditions under which the constraint can be guaranteed to be required by the SDM imposing it. We will discuss the issues involved in generating and coordinating with such window of applicability explanations.

[2]This is in contrast to approaches such as [Hayes, 1987], that allow externally imposed constraints to enter the plan, but do not reason about the effect of those constraints on the correctness of the plan.

[3]It should be emphasized here that we are not concerned with the absolute correctness of the plan, but rather its correctness with respect to the planner's own model of the domain (see [Kambhampati, 1990a]).

2 Incrementality

In this section we discuss the issues involved in making planning ''incremental''. We will start with a characterization of correctness of plans in the hierarchical planning paradigm in terms of the plan validation structure. This characterization is used to develop a methodology for modifying a plan to regain correctness. The resultant incremental modification framework forms the basis for accommodating various externally imposed constraints into an existing plan (see next section).

2.1 Validation Structure and Plan Correctness

Hierarchical planning can be seen as a process of reduction of abstract tasks into more concrete subtasks with the help of domain specific task reduction schemata, and resolving any harmful interactions by introduction of additional partial ordering relations among tasks or backtracking over previous decisions [Tate, 1977, Sacerdoti, 1977]. Given a planning problem $[\mathcal{I}, \mathcal{G}]$ where $\mathcal{I}$ is the specification of the initial state and $\mathcal{G}$ is the specification of the desired goal state (given as conjunction of literals to be satisfied), we use a structure called hierarchical task network (HTN) to represent the status of the plan at any moment. A HTN is a 3-tuple

$$\langle\ \mathcal{P} : \langle T, O, \mathcal{V}\rangle\ , T^*\ , D\ \rangle$$

where $\mathcal{P}$ is a partially ordered plan such that

- T is the set of tasks of the plan (with two distinguished tasks $t_\mathcal{I}$ and $t_\mathcal{G}$ to denote the input and goal state specifications respectively)
- O defines a partial ordering over T (with elements of the form ''$t_i \rightarrow t_j$'', signifying that t_i is a predecessor of t_j)
- $\mathcal{V}$ is a set of conditions with specification about the ranges where those conditions should be held true, and the applicability conditions of tasks of $\mathcal{P}$, or goals of the overall plan those conditions are supporting. Individual elements of $\mathcal{V}$ are called *validations*. They are represented by 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 from task t_s to t_d to support the applicability condition C of task t_j.
- T^* is the union of tasks in T and their ancestors
- D defines a set of parent, child relations among the tasks of T^* (if t^p is the parent of a task t, then t was introduced into the HTN because of the reduction of t^p)

We define the correctness of the plan $\mathcal{P}$ in terms of its set of validations in the following way:
A partially ordered plan $\mathcal{P}$ is considered a correct plan for a problem $[\mathcal{I}, \mathcal{G}]$ iff

- For each goal $g \in \mathcal{G}$ of the plan, and each applicability condition of a task $t \in T$, there exists a validation $v \in \mathcal{V}$ supporting that goal or condition. If this condition is not satisfied, the plan is said to contain *''missing validations.''*
- None of the plan validations are violated. That is, $\forall v : \langle E, t_i, C, t_j \rangle \in \mathcal{V}$, (i) $E \in effects(t_i)$ and (ii) $\nexists t \in T$ s.t. $\Diamond(t_i \prec t \prec t_j) \wedge effects(a) \vdash \neg C$ (where, the relation ''$\prec$'' is the transitive closure of the relation ''$\rightarrow$''). If this constraint is not satisfied, then the plan is said to contain ''*failing validations*.''

In addition, we introduce a condition of non-redundancy as follows

- For each validation $v : \langle E, t_s, C, t_d \rangle \in \mathcal{V}$, there exists a chain of validations the first of which is supported by the effects of t_d and the last of which supports a goal of the plan.[4] If this constraint is not satisfied, then the plan is said to contain ''*unnecessary validations*.''

A plan that is correct by this definition is said to have consistent validation structure. The missing, failing and unnecessary validations defined above are collectively referred to as *inconsistencies* in the plan validation structure.

Notice that this definition of correctness is concerned exclusively about the validations that the planner has established. The plan may be correct by this definition and still be inapplicable from the point of view of some SDM . This is in consonance with the philosophy that the planner in a concurrent environment should primarily be concerned about keeping the plan correct from its perspective. When changes are necessitated by the externally imposed constraints, the planner should preserve correctness with respect to its validation structure while accommodating those changes.

2.2 Annotating Validation Structure

To facilitate efficient reasoning about the correctness of the plan, and to guide incremental modification, we represent the plan validation structure as annotations on the individual tasks constituting the HTN [Kambhampati, 1990c]. In particular, for each task $t \in T^*$, we define the notions of *e*-conditions , *e*-preconditions and *p*-conditions as follows: The *e*-conditions of a task represent the set of validations that it or its descendents in the HTN provide to the rest of the plan. If $R(t)$ represents the set of tasks consisting of t and its descendents in the HTN (also called sub-reduction)[5], $e-$conditions(t) is given by the set

$$\{v \mid v : \langle E, t_s, C, t_d \rangle \in \mathcal{V} \ \wedge\ t_s \in R(t) \ \wedge\ t_d \notin R(t)\}$$

[4]More formally, $\forall v : \langle E, t_s, C, t_d \rangle \in \mathcal{V}$ there exists a sequence $[v^1, v^2 \ldots v^k]$ of validations belonging to $\mathcal{V}$, such that (i) $v^k : \langle E^k, t_s^k, C^k, t_{\mathcal{G}} \rangle$ supports a goal of the plan (i.e, $C^k \in \mathcal{G}$) (ii) $v^{k-1} : \langle E^{k-1}, t_s^{k-1}, C^{k-1}, t_s^k \rangle$ supports an applicability condition of t_s^k, v^{k-2} supports an applicability condition of t_s^{k-1} and so on, with v^{k-1} supported by an effect of t_d.

[5]note that $R(t) = \{t\}$ if $t \in T$

The *e*-preconditions represent the set of validations that the task or its descendents consume from the rest of the plan. $e-$preconditions(t) is given by the set:

$$\{v \mid v : \langle E, t_s, C, t_d \rangle \in \mathcal{V} \ \wedge\ t_s \notin R(t) \ \wedge\ t_d \in R(t)\}$$

Finally, the *p*-conditions represent the validations that should necessarily be preserved by the effects of the task and its descendents to guarantee the correctness of the plan. $p-$conditions(t) is given by the set

$$\left\{ v \,\middle|\, \begin{array}{l} v : \langle E, t_s, C, t_d \rangle \in \mathcal{V} \ \wedge\ t_s, t_d \notin R(t) \ \wedge \\ \exists t_i \in T \ \ s.t.\ t_i \in R(t) \ \wedge\ \Diamond(t_s \prec t_i \prec t_d) \end{array} \right\}$$

The *e*-conditions , *p*-conditions and *e*-preconditions (referred to collectively as task annotations) encapsulate the role played by each task in the HTN of the plan in ensuring the correctness of the plan.

2.3 The PRIAR Modification Framework

Based on the notion of validation structure, we have developed a framework for flexible modification of plans in hierarchical planning called PRIAR [Kambhampati, 1990c, Kambhampati, 1989]. In PRIAR framework, a plan is modified in response to inconsistencies in its validation structure. The repair actions depend on the type of inconsistency, and the type of validation involved in that inconsistency. They involve removal of redundant parts of the plan, exploitation of any serendipitous effects of the changed situation to shorten the plan, and addition of high level (refit) tasks to re-establish any failing validations. For example, if the inconsistency is a failing validation $v : \langle E, t_s, C, t_d \rangle$, then depending upon whether or not C can be achieved by the planner, PRIAR either adds a task t_r to the HTN re-establish the failing validation, or replaces the reductions that are dependent on C. The annotations on the individual tasks of the HTN (defined in section 2.2) provide a systematic framework for locating the parts of the plan that need to be removed or replaced. Finally, any refit-tasks introduced during the repair process are reduced by the hierarchical planner, which employs various validation structure based control strategies to localize this reduction [Kambhampati and Hendler, 1989, Kambhampati, 1990b].

3 Accommodating External Constraints

In this section, we discuss how the planner accommodates various types of changes necessitated by by externally imposed constraints. Without loss of generality we will assume that the planner has an initially correct plan which it wants to modify to accommodate the changes and while preserving the correctness of the plan. We will also assume at this point that the changes imposed by the external constraints are non-negotiable, in that they would have to be necessarily accommodated by the planner. In the previous section, we have shown that the planners notion of correctness of its plan is intimately tied to the consistency of the plan validation structure. Thus, the general strategy followed for accommodating changes necessitated by externally imposed constraints is to compute the inconsistencies in the validation structure that result from those changes and use the PRIAR framework to modify the plan to remove the detected inconsistencies. In the following sections we discuss how changes in the specification the problem, and ordering relations among the plan steps are handled through this strategy.

3.1 Changes in Input and Goal Specifications

Often during planning the specifications of the planing problem are modified to reflect the changes in the state of the world and the overall goals of the system. From the definition of correctness of the plan in section 2.1, it should be clear that the changes in problem specifications may give rise to missing, failing or unnecessary validations in the plan validation structure. For example if some of the goals of the plan become unnecessary as a result of changes in the specifications, the validations supporting those goals become unnecessary; if the changes necessitate new goals for the plan, that would lead to a missing validation and finally if the changes delete some assertions of the initial state then the validations supported by the effects of $t_{\mathcal{I}}$ will fail. These inconsistencies in the validation structure can be located by simply examining the validations supported by the assertions in the initial state, and goal state. Any detected inconsistencies are then repaired with the help of PRIAR modification operations.

3.2 Changes in ordering relations

Often external constraints translate into addition or deletion of ordering relations among the steps of the plan. Such changes may be a result of feasibility considerations (for example, in process planning, the geometric modeler might rule out some infeasible feature orderings [Hayes, 1987]), or of optimality considerations (eg. SDMS may group several steps of the plan together for efficient execution etc., and such groupings may be inconsistent with the existing ordering relations [Kambhampati and Philpot, 1990]). In the following we provide methods for efficiently analyzing the ramifications of addition and deletion of ordering relations on the correctness of the plan.

3.2.1 Deletion of Ordering Relations

Suppose an ordering relation $o_d : t_a \rightarrow t_b$ is to be deleted as a consequence of some externally imposed constraint. This leads to a change in the partial ordering of the plan and some of the tasks which were previously ordered with respect to each other will now become unordered (parallel).

From the definition of correctness in section 2.1, it should be clear that failing validations are the only type of inconsistencies that can result from the deletion of an ordering relation.[6] Since checking each validation $v \in \mathcal{V}$ for failure is expensive, we use the following method to check only those validations which can fail because of the deletion of o_d.

First o_d is removed from O and the ''$\prec$'' relation on the tasks of the plan is recomputed. The predecessor-successor fields of the tasks of $\mathcal{P}$ before and after the deletion are compared. For each task $t_i \in T$ of $\mathcal{P}$, we collect the tasks of the plan that become unordered with respect to it. Suppose t_i becomes unordered with respect to tasks $\{t_{i_1}, t_{i_2} \ldots t_{i_m}\}$. From the definitions of section 2.2, we can show that when previously ordered tasks become parallel to t_i, the annotations of those tasks will also become p-conditions of t_i.[7] In other words, the new p-conditions of t_i are given by

$$p\text{–conditions}^{old}(t_i) \cup \bigcup_j annotations(t_{i_j})$$

By definition p–conditions(t_i) comprises the validations of the plan that must be preserved by the effects of t_i, Thus, to locate the validations that are violated by the effects of t_i it is sufficient to check the annotations of the tasks $\{t_{i_1}, t_{i_2} \ldots t_{i_m}\}$. If any validations are actually found to be violated, the PRIAR modification methodology is again used to suggest repair actions. Similar analysis is done for each task in the plan that becomes unordered with respect to other tasks because of the deletion of o_d.

3.2.2 Addition of Ordering Relations

Suppose the externally imposed constraint leads to the addition of a new ordering relation $o_n : t_a \rightarrow t_b$. We have two possibilities in this situation:

Case 1. *The added ordering is consistent with the current orderings.* As long as $t_b \not\prec t_a$ in the current plan, we can add $t_a \rightarrow t_b$ without causing any inconsistencies. From the definition of the correctness $\mathcal{P}$, we can see that a consistent new ordering *will not* affect the correctness of the existing plan. Thus nothing need be done in this case.

Case 2. *The added ordering is* not *consistent with the current ordering.* That is $t_b \prec t_a$ in the current plan. In this case, there are some ''cycles'' in the ordering O on $\mathcal{P}$. Let

$$t_b \rightarrow t_{i_1} \ldots t_{i_m} \rightarrow t_a \overset{o_n}{\rightarrow} t_b$$

be a cycle [8]. Since o_n has to be necessarily accommodated, the only way of regaining correctness is to break such cycles by removing some other previously held ordering relations in this cycle. Removal of an ordering relation may cause inconsistencies in the validation structure and those need to be located and repaired as discussed in section 3.2.1. The analysis of section 3.2.1 can also be used to decide which ordering relation will necessitate least amount of repair work upon deletion, and choose to delete that ordering relation.

4 Coordination through reasoning about the rationale behind externally imposed constraints

The techniques discussed in the previous two sections enable the planner to play a passive role of efficiently accommodating

[6]Deletion of ordering relations increases the parallelism in the plan and this may lead to violation of some established validations of the plan.

[7]If a validation $v : \langle E, t_s, C, t_d \rangle \in \mathcal{V}$ belongs to the annotations of a task $t_j \in T$, then, given that $\forall t \in T\ R(t_j) = \{t_j\}$, we have (see section 2.2)

$$\Diamond(t_s \prec t_j \prec t_d) \lor (t_s = t_j) \lor (t_d = t_j)$$

Now suppose t_j became unordered with respect to $t_i \in T$. We have

$$\Diamond(t_j \prec t_i) \land \Diamond(t_i \prec t_j)$$

From the two relations above, we can see that if $v : \langle E, t_s, C, t_d \rangle$ belongs to the annotations of t_j then it will also satisfy the relation

$$\Diamond(t_s \prec t_i \prec t_j)$$

which in turn means that v is a p-condition of t_i.

[8]cycles can be found by scanning the ''$\rightarrow$'' and ''$\prec$'' relations of the tasks of the plan

changes necessitated by any externally imposed constraints on its plan. As the specifications and the plan evolve, the external constraints also evolve and it might be possible to relax some of them under the new situation. Rather than waiting for SDMS to take the initiative, the planner should play an active role in guiding the global problem-solving activity anticipating external interactions and coordinating its actions with the SDMS . The simplest approach for this would require the planner to request all the modules about all the constraints imposed by them, when ever *anything* changes. However, this could be quite inefficient especially since the analyses performed by some SDMs may be computationally expensive. In addition, this approach also seriously undermines the autonomy of the planner as it cannot take any decisions without having them approved by all the modules.

To take an active role in the global problem-solving activity, the planner needs to understand and reason about the rationale behind the externally imposed constraints. This requirement is symmetric: when the planner makes any decisions that affect the outside modules, it should be able to associate a rationale for that decision. In other words, a major requirement for a module to work in a concurrent environment is to be able to provide rationale for the decisions that can be understood by other modules in the environment, and to be able to reason with the rationales provided by the outside modules.

Though the issue of coordination have been addressed previously in distributed planning and blackboard based approaches to planning, they generally assume that the individual planners are all identical and share a common representation (e.g. [Durfee and Lesser, 1988]) and that there is a single central knowledge base shared among all modules and planners [Hayes-Roth, 1987, Durfee and Lesser, 1988]. Because of these assumptions, coordination is accomplished in those architectures by allowing each planner to directly reason about the constraints imposed by other modules. This is controlled either through a central black-board mechanism [Hayes-Roth, 1987], a hierarchical organization of planners or through the use of localized representations that explicitly circumscribe the effects of the agent's actions [Lansky, 1988].

In concurrent domains, the assumption of homogeneous modules holds only in situations whose where the modules of a concurrent environment constitute a ''vertical'' decomposition of the domain, where each module takes all the specialized considerations independently. However, in concurrent domains, the environment typically consists of heterogeneous modules which specialize in various sub-tasks of the overall task. For example, in a domain like process planning where there are several specialized considerations (such as geometric, fixturing etc.) that need to be taken into account, if the planner itself were to take them all into account during planning, plan construction could become prohibitively expensive [Simmons and Davis, 1987]. Further more such an architecture will often be inconsistent with the natural structure of the domain. In process planning, even if the planner is designed to take into account all the geometric considerations during planning, duplicating all the geometric knowledge and inference formalisms into the planner would be very wasteful.

Because of this heterogeneity among the modules , it is not feasible to have each module understand the rationale behind the decisions of the other modules, *even* if it were provided to them. For example in process planning, it is infeasible for the planner producing machining sequence to understand all the details of constraints posted by modules specializing in fixturing and geometric reasoning etc.

Suppose an external constraint c_e (say an ordering relation) is imposed on the plan by the geometric modeler to ensure that there will be a clear access path to make a feature $feature_1$. There is no straightforward way for the geometric modeler to communicate to the planner that the the constraint c_e is imposed to ensure clear access path to $feature_1$ since the planner's domain model may not give it the ability to reason about the notion of ''clear access path.'' In the absence of any rationale accompanying c_e, the planner would have to rely on the geometric modeler to correctly update the constraints in the event that constraint is no longer needed. This approach, in as much as it requires frequent geometric checks, is inefficient.

Thus, for computational tractability and functional autonomy of individual modules, planner in a concurrent environment should be able to understand the rationale behind external constraints at some appropriate level of detail.

4.1 Window of Applicability (WOA) Explanation

Since the individual modules cannot reason about the rationale behind the externally imposed constraints directly, we propose that the rationale be presented as a set of sufficiency conditions called ''WOA (Window of applicability)'' explanation for the external constraint. These sufficiency conditions do not constitute the complete set of conditions leading to the imposition of the external constraint, but rather only an abstraction of those conditions that the planner can reason about.

Thus, each externally imposed constraint will be associated with a 3-tuple

$$\langle c_e, \text{woa}_{c_e}, \text{sdm}_{c_e} \rangle$$

where c_e is the constraint (changed specification or ordering relation) that the planner needs to accommodate into its plan, sdm_{c_e} is the SDM that is imposing that constraint, and woa_{c_e} is the WOA explanation associated with c_e. The semantics are that

- WOA consists only of predicates that the planner can reason about [9]
- As long as woa_{c_e} is holding, c_e should necessarily be accommodated into its plan
- If the planner finds at any point of time that woa_{c_e} is not holding, then it is possible that c_e is no longer required. The planner can then request the sdm_{c_e} to check if the constraint is still required. [10] (If the

[9]A proposition f is considered *reachable* if the planner can reason about the ramifications of its actions on the truth of the proposition. The WOA should only consist of reachable propositions. In TWEAK representation, which does not allow any domain axioms, a proposition is reachable iff f codesignates with some proposition in the add or delete lists of some operator template of the planner.

[10]The external constraints may have been voluntarily imposed by the external modules or may have been posted to satisfy some request by the planner. In the process planning example, either the geometric modeler may have imposed c_e because *it* wanted to ensure clear access path, or alternatively the planner requested it to

sdm$_{c_e}$ were to retract the constraint c_e, then the planner can use the methodology developed in sections 2 and 3 to appropriately modify the plan to accommodate the change.)

Note that this allows the planner to essentially ignore outside modules unless the WOA of the constraint posted by them are affected. This provides functional autonomy to the planner, by obviating the need to poll each and every SDM for every change in the environment.

In the process planning example discussed above, the planner might be given the WOA explanation

$$a \leq \mathit{length}(\mathit{feature}_i) \leq b$$

for the constraint c_e, with the semantics that as long as the length of the feature $\mathit{feature}_i$ is in the range $[a\, b]$, the constraint c_e would be required by the geometric modeler. When the WOA is violated, it does not mean that the c_e is automatically retracted. Instead, the corresponding SDM (in this case geometric modeler) will be requested to repeat the computation to see if c_e is still required in the current situation.

4.2 Classes of WOA Explanations

A critical issue about the WOA explanation concerns their level of detail: as mentioned above, they should be at such a level as to allow the planner to reason with the explanation, and they should constitute *sufficient* conditions for assuming that c_e is still needed. There may be a spectrum of such conservative sufficiency conditions for any given external constraint, with tradeoffs between the informedness of the WOA and the ability of the outside modules to reason with it.

For instance, in the process planing example above, the geometric modeler could return as the WOA of the ordering constraint a representation of the configuration space in which the $\mathit{ClearAccess}(h_1)$ predicate will remain. However, that WOA is likely to be at too low a level of detail to be directly useful to the planner. On the other hand it could return as the WOA a list of quantities whose values it used in arriving at c_e as the constraint to be imposed. In this case, the planner would have to reinvoke the geometric modeler any time any of those quantities change.

While the former WOA is expressive but difficult to reason with, the latter is easy to reason with, but very conservative. The correct level of abstraction of WOA depends ultimately on the degree of similarity between the domain models and the inference strategies of the planner and the SDMS . For example, if all the modules are identical, as in distributed planning, the WOA can include both the necessary and sufficient conditions for c_e. While in complex environments with heterogeneous modules, even a WOA that specifies the quantities on whose values c_e depends, would be of utility.

A promising strategy is to provide WOA explanations that are at multiple levels of abstraction. This will allow the planner to choose the abstraction that it can handle, while still allowing some other SDM to reason with the WOA at a deeper level of detail.

ensure clear access path. In the latter case, the planner may model clear access path as precondition $\mathit{ClearAccess}(\mathit{feature})$ of some task of $\mathcal{P}$ whose truth needs to be computed by an outside module (*cf.* ''compute-conditions'' of nonlin [Tate, 1977]). When asked to make such a condition true, the geometric modeler may then impose some constraints (such as c_e) on the current plan. For the purposes of our current discussion, this distinction is immaterial.

Careful investigation will be needed to find classes of explanations that satisfy the conservative (sufficiency) property of the window of applicability explanations, while at the same time allowing reasoning at a level suitable for other modules.

5 Summary

Planning in many real world situations requires concurrent operation between the planners and other modules of the environment. This poses several requirements that are not supported by the existing approaches to automated planning. In particular, we identified the ability to incrementally accommodate changes necessitated by the externally imposed constraints into the existing plans, and the ability to understand and reason about the rationale behind externally imposed constraints at an appropriate level of detail as two crucial requirements for planning in such environments. We have then explored ways of extending the hierarchical planning framework to handle these requirements. In particular, we discussed the techniques for making the planning incremental and for accommodating the externally imposed constraints by reasoning about their effect on the correctness of the plan. Next we proposed a methodology for coordination between planners and external modules that depends on attaching a set of sufficiency conditions rather than necessary and sufficient conditions as justifications to various externally imposed constraints. We have discussed the issues of generating and reasoning with such conditions.

Acknowledgements

We would like to thank Mark Cutkosky for many helpful discussions and comments on a previous draft of this paper. Andrew Philpot and Randy Wilson also provided helpful suggestions.

References

[Chapman, 1987] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3), 1987.

[Corkill, 1979] D.D. Corkill. Hierarchical planning in a distributed environment. In *Proceedings of Sixth IJCAI*, August 1979.

[Cutkosky and Tenenbaum, 1990] M. R. Cutkosky and J. M. Tenenbaum. A methodology and computational framework for concurrent product and process design. *Mechanism and Machine Theory*, 23(5), 1990.

[Durfee and Lesser, 1988] E.H. Durfee and V.R. Lesser. Predictability versus responsiveness: Coordinating problem solvers in dynamic domains. In *Proceedings of Seventh NCAI*, August 1988.

[Georgeff, 1990] M. Georgeff. Decision-making in an embedded reasoning system. In *Proceedings of Eleventh IJCAI*, August 1990.

[Hayes-Roth, 1987] B. Hayes-Roth. Dynamic control planning in adaptive intelligent systems. In *Proceedings of DARPA Knowledge-Based Planning Workshop*, December 1987.

[Hayes, 1987] C. Hayes. Using goal interactions to guide planning. In *Proceedings of 6th National Conference on Artificial Intelligence*, pages 224--228, July 1987.

[Kambhampati and Cutkosky, 1991] S. Kambhampati and M. R. Cutkosky. An approach toward incremental and interactive planning for concurrent product and process design. In *Proceedings of ASME Winter Annual Meeting on Computer Based Aproaches to Concurrent Engineering*, (To appear) 1991.

[Kambhampati and Hendler, 1989] S. Kambhampati and J.A. Hendler. Control of refitting during plan reuse. In *Proceedings of 11th International JointConference on Artificial Intelligence*, pages 943--948, August 1989.

[Kambhampati and Philpot, 1990] S. Kambhampati and A. Philpot. Incremental planning for concurrent product and process design. Technical report, Center for Design Research and Computer Science Department, Stanford University, CA, (In preparation) 1990.

[Kambhampati and Tenenbaum, 1990] S. Kambhampati and J.M. Tenenbaum. Towards a paradigm for planning in interactive domains with multiple specialized modules. In *AAAI-90 Workshop on Automated Planning for Complex Domains*, August 1990.

[Kambhampati, 1989] S. Kambhampati. *Flexible Reuse and Modification in Hierarchical Planning: A Validation Structure Based Approach*. PhD thesis, Center for Automation Research, Department of Computer Science, University of Maryland, College Park, MD 20742, October 1989.

[Kambhampati, 1990a] S. Kambhampati. A classification of plan modification strategies based on their information requirements. In *AAAI-90 Spring Symposium on Case-Based Reasoning*, March 1990.

[Kambhampati, 1990b] S. Kambhampati. Mapping and retrieval during plan reuse: A validation-structure based approach. In *Proceedings of 8th National Conference on Artificial Intelligence*, August 1990.

[Kambhampati, 1990c] S. Kambhampati. A theory of plan modification. In *Proceedings of 8th National Conference on Artificial Intelligence*, August 1990.

[Lansky, 1988] A. Lansky. Localized event based reasoning for multiagent domains. *Computational Intelligence Journal*, 4(4), 1988.

[Sacerdoti, 1977] E.D. Sacerdoti. *A Structure for Plans and Behavior*. Elsevier North-Holland, New York, 1977.

[Simmons and Davis, 1987] R. Simmons and R. Davis. Generate, test and debug: Combining associational rules and causal models. In *Proceedings of 10th International Joint Conference on Artificial Intelligence*, August 1987.

[Tate, 1977] A. Tate. Generating project networks. In *Proceedings of 5th International Joint Conference on Artificial Intelligence*, pages 888--893, 1977.

[Wilkins, 1984] D.E. Wilkins. Domain independent planning: Representation and plan generation. *Artificial Intelligence*, 22:269--301, 1984.