Modeling Planning Tasks

Leliane Barros
Laboratory of Integrated Systems
University of São Paulo
Av. Luciano Gualberto 158 trav. 3
05508-900 São Paulo, Brazil
e-mail: leliane@lsi.usp.br

André Valente
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292, USA
e-mail: valente@isi.edu

Richard Benjamins
University of Amsterdam
Dept. of Social Science Informatics
Roetersstraat 15
1018 WB Amsterdam, Netherlands
e-mail: richard@swi.psy.uva.nl

Abstract

The analysis of planning systems in terms of the different algorithms they use needs to be complemented by a study of how these systems represent and use available knowledge — in other words, of their *problem solving methods*. The goals of such *knowledge-level* analysis are complementary to that of algorithms: instead of finding which algorithm is the most efficient, we can find the *role* of each piece of knowledge in the system. This is an essential aspect in finding out how to engineer a planning system and obtain a desired performance in an application and its engineering. One analysis of this type has been made in (Valente 1995). However, that analysis was incomplete, because it concentrated on the organization of the domain knowledge used by the system. Another equally important aspect is the way this knowledge is used in the planning process. These two are complementary aspects in the description of a problem-solving method. In this paper, we will depart from and complement Valente's analysis by providing a more detailed discussion on planning tasks, providing a general, high-level task decomposition for planning.

Keywords: classical planning, comparison of planning problem-solving methods, knowledge-level analysis

Introduction

There is a large amount of research in the planning community on the design and analysis of planning algorithms, with particular emphasis on their efficiency. This has been the main axis used to study and compare planning systems. Lately, several important results have been achieved in this direction (Minton, Bresina, & Drummond 1994; Kambhampati, Knoblock, & Yang 1995). Despite the importance of these efforts, there are, however, other aspects of planning systems which must be taken care of, especially if we want to bridge the gap between planning theory and applications. Sometimes the key features that make a planning application successful have less (if anything) to do with the power of its algorithm and more with less studied features such as the adequacy of planning methods to domain features or still to the expressiveness of the language used to represent knowledge about the domain and the plans. For instance, (Drummond 1994) claimed that, while most of the literature on the analysis of planning algorithms has concentrated on precondition achievement planning, the most important capabilities of some well-known successful applications (based on e.g. SIPE (Wilkins 1988)) were the capabilities to (i) represent hierarchical operators (i.e. using some sort of HTN planning), (ii) allow the user to document a plan's causal structure (e.g. by specifying sub-plans or skeletal plans), and (iii) represent and use plan resources. Drummond pointed out that these characteristics, commonly connected to the definition of sub-plans or skeletal plans, in practice, hardwired most of the work which was supposed to be done by the precondition achievement techniques.

This seems to indicate that there is a gap in the analysis of planning systems. Algorithms and their efficiency must surely be a central concern, but if we are to engineer successful applications there must be also a more abstract analysis that highlights the capabilities of the system and the way it represents and uses knowledge, away from the details of how it is to be implemented. Such analysis is not only useful for applications, but it may provide an interesting perspective in analyzing and comparing different systems. We claim that a valuable way to do such high-level analysis is to study how these systems represent and use available knowledge to solve the problem — in other words, what *problem-solving methods* they use. The goals of such *knowledge-level analysis* are complementary to that of algorithms: instead of finding which algorithm is the most efficient, we can find what is the role of each piece of knowledge in the system, which is an essential aspect in finding out how to engineer a planning system and obtain a desired performance in an application and its engineering.

There are several efforts in what we regard as a knowledge-level analysis of planning. For example, a large part of the research in formal models of planning can be seen in this perspective. Further, efforts in the specification of ontologies of planning (e.g. the KRSL standardization effort[1]) have the same flavor. This paper expands on and complements the analysis of conceptual models of planning methods made by (Valente 1995). His work is based on the representation of problem-solving methods in conceptual terms, in the tradition of KADS (Schreiber 1992). Valente focused his analysis on what types of domain knowledge are used in planning systems, and how this knowledge is structured. However, another equally important aspect which was only briefly discussed in (Valente 1995) is the way this knowledge is *used* in the planning process. These two are complementary aspects in the description of a problem-solving method — respectively the *domain* and *task* knowledge. In this paper, we extend this work by providing a more detailed discussion on *models of planning tasks*. We provide a general, high-level task decomposition for planning, and show how different planning systems (i) instantiate this decomposition by executing its steps using different methods, and (ii) impose different control structures to control the inferencing process. It is important to note that we maintain the same focus of (Valente 1995) in that we are not modeling all possible types of planning systems (e.g. re-planning, planning agents, reactive planning systems), but only systems that elaborate a plan to achieve a number of goals (or a final state) — this is what we mean by "planning" in the remainder of this article.

Ingredients of a Task Analysis

Approaches to knowledge-level analysis and modeling include Role-limiting Methods (Marcus 1988) and CommonKADS (Schreiber 1992). Their common characteristic is to define a set of high-level components which are used to express problem-solving knowledge. CommonKADS (also known in earlier versions as KADS) is a methodology for developing knowledge-based systems, created during a multi-year research effort that has included several ESPRIT projects. Briefly, the components used to represent problem-solving knowledge in KADS are:[2]

Tasks are represented using a functional perspective, i.e. specified by input/output relations which is called in KADS a *function*. A function also includes a *goal* i.e. a specification of *what* needs to be achieved. For example, the goal of planning task is to come up with an ordered sequence of operations that transforms a given initial state into a goal state. We represent the input and output by roles that knowledge plays in the problem-solving process. For instance, an input or output role like hypothesis means that some domain structure or element plays the role of hypothesis. Static roles are the ones whose contents do not change during problem solving (they correspond roughly to a knowledge base in a knowledge-based system); dynamic roles are the ones that do change.

Problem-solving methods (PSMs) describe *how* the goal of a task can be achieved. A method has input and output roles, and either decomposes a task into subtasks (composite method), or is primitive. Tasks and PSMs can be organized in a task-method decomposition structure, where a task may be realized by alternative methods, each of which consists of subtasks or is primitive. Fig. 2 presents such a diagram for planning. A primitive method can usually be implemented by a general or weak method (e.g. constraint propagation). A composite method specifies the allowed data flow (in terms of their roles) between the subtasks, and the control knowledge over them. Control knowledge is represented by a **control structure**, which determines the execution order and iteration of the subtasks. In the specification of a control structure there is no concern for precision (e.g. defining what is the actual content of symbolic variables). Instead, we only wish to describe the overall control regime. Without being committed to a particular control structure, the subtasks of a PSM form a **function-structure**, that captures the data flow between the subtasks. We will show an example of a function-structure and a control structure later in the paper.

Domain Ontologies provide a vocabulary that can be used to describe a certain domain. An ontology specifies the concepts, relations, etc. independently from the specific system that is reasoned about. Examples of domain ontologies in planning domains are the formalisms to represent the world, time and actions, such as the situation calculus or the event calculus. We will not deal further with domain ontologies in this paper. **Domain models** are structures built using elements of the domain ontology to be used in problem-solving methods. Typical examples are models that specify causal dependencies between symptoms and diseases in medical domains (causal model), or models that specify structural hierarchies for physical devices (structural models). We will discuss typical domain models for planning tasks in the following section.

[1] See the Web page http://www.aiai.ed.ac.uk/~bat/krsl-plans.html

[2] For clarity, the original KADS terminology has been simplified in some places.

Knowledge Roles in Planning

The basic problem solved in planning tasks is to generate a sequence of actions (a plan) whose execution achieves a given goal state. A planning problem is characterized by two inputs: an initial state and a goal state description of the world. One of the critical elements in the analysis of a planning method is to specify what types of knowledge it employs — in other words, which *roles* knowledge can play. In this section we will present (i) a set of four general dynamic roles in planning tasks; (ii) a part-of organization of static roles, originally defined in (Valente 1995). These two sets give a high-level view on what types of knowledge are used in planning, both in the sense of the knowledge which is input or output to the task and which is used as part of the knowledge base.

Dynamic Roles

Current state The role current state is initially filled by a description of the world in the beginning of the plan, and later it is modified to contain intermediary states in the plan.

Goal The role goal describes the world state that has to be achieved by the plan. The content of goal is a set of *conditions*. Initially this role points to the original goal to be achieved. During the planning process the content of the role is dynamically modified by establishing new subgoals and deleting achieved goals.

Plan The dynamic knowledge role plan is a composite role whose content is constantly modified during the planning process until a solution is found. It consists of:

(1) **Plan-steps** which correspond to unique action instances, along with their pre-conditions and the conditions that they can achieve..

(2) **Ordering constraints** over the plan-steps, such as that one action precedes another. The type of order imposed on the plan-steps in the plan (e.g. partial or total) depends on the static role plan structure (which will be described later) employed by the planner.

(3) **Variable bindings** which keep track of how variables of plan-steps are instantiated with domain knowledge such as objects, resources and agents.

(4) **Causal links** that represent causal connections between the plan-steps and the conditions they achieve. We define a causal link by (i) a condition in the plan that has to be true (e.g. a goal condition), (ii) a plan-step that needs this condition to be true and (iii) another plan-step that makes this condition true. In schema: $plan\text{-}step_1 \xrightarrow{condition} plan\text{-}step_2$. A causal link defines an interval of truth value for a condition. Not all planning methods use causal links in their representation of plans, as we will discuss later.

Conflict The role conflict contains the result of checking the plan with respect to the validity of a condition. Each time when a condition is unexpectedly false, a conflict is detected. When a plan has to achieve multiple goals (i.e to make true multiple conditions), conflicts easily arise because of interacting actions. The conflict role points to a plan-step that violates some interval of truth value of a condition. A conflict can also mean an inconsistency in the plan constraints. This knowledge role is only used by some planners, in particular the ones that explicitly modify generated plans.

Static Roles: the Plan Model

The part-of organization of static roles proposed in (Valente 1995) is shown in Fig. 1. The rationale behind this part-of tree of static roles is twofold. First, it means that all planning methods use a certain configuration of these roles. The minimal case is to have a single domain model to play the role of plan model, but more specific configurations may occur as well. Second, the reason why these roles represent a part-of tree is because they are interdependent, and thus should maintain coherence. They represent parts of the description of the same whole: the terms or structures defined in one part are used or referred to in another.

The plan model defines what a plan is and what it is made of. It consists of two parts: the world description, which describes the world about which we are planning, and the plan description, which describes the structure and features of the plan being generating. Below is brief description of these roles and their sub-roles.

World Description The world description role comprises two sub-roles, state description and state changes. The state description comprehends the knowledge necessary to represent or describe the state of the world. Examples of domain models to play this role in planning systems are a set of first order predicates (STRIPS-like) or a set of fluents from the Situation Calculus. The state changes role comprehends all the information connected to the specification of changes in the state of the world. This is also the specification of the elements a plan is composed of (but not *how* they are composed, see plan composition below). Examples of domain models to play this role in planning systems are the add-delete lists (ADL) used in STRIPS, or SOUP procedures as in NOAH.

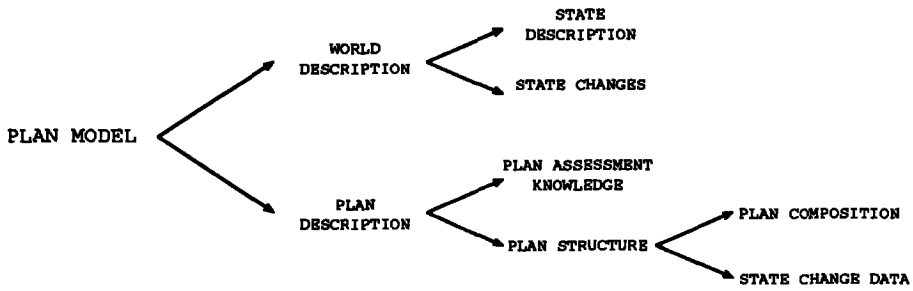

Figure 1: Part-of tree of static roles in planning.

Plan description The plan description role comprises two sub-roles: plan structure and the (optional) plan assessment knowledge:

(1) **Plan structure** This role specifies how the parts of a plan (actions, sub-plans) are assembled together. It also specifies (indirectly) how the plan is to be executed. There are several varieties in the structure of plans that can be identified in the literature. They can be described by two main knowledge roles: the plan composition role contains the description of the plan with respect to how the state changes are arranged in order to make up a plan. This includes, for instance, whether the plan will be a partial or a total ordering of a set of state changes, or whether it includes iteration or conditional operators. The composition may also be hierarchical: plans are composed of *sub-plans*, and so on up to *atomic* plans, which are normally state changes. The state change data role contains the plan information besides the structure of state changes. For example, important state change data are interval constraints for binding the variables involved in the state changes. It is also possible to assign different *resources* to each state change or sub-plan. Two particularly important resources are agents and time.

(2) **Plan Assessment Knowledge** determines what kind of factors make a certain plan (or sub-plan) better then another, and how they are to be taken into account (e.g. weighted). Based on this knowledge a plan can be either criticized or modified. An example of plan assessment knowledge is the *truth-criterion*, which is used to find out if a condition is true at some point in the plan. TWEAK (Chapman 1987) uses a modal truth criterion (MTC) which defines when a particular condition in a plan is necessarily or possibly true by formally stating all possible interaction problems. Another example of a domain model for "hard" plan assessment knowledge is the *causal-link protection* used in SNLP (McAllester & Rosenblitt 1991) (see discussion below).

A Framework for Modeling Planning Tasks

Based on an analysis of many classical planning systems, we have identified several relevant tasks and problem-solving methods. They can be organized into a task-method decomposition structure (see Fig. 2), where a method consists of (solid lines) subtasks and a (sub)task can be realized by alternative (dashed lines) methods. Ellipses represent tasks and rectangles methods.

We claim that the class of planners we are dealing with share a general, high-level problem-solving method which we call propose-critique-modify (PCM). That is, all planners contain in one way or another these three basic tasks: (i) *propose expansion*, (ii) *critique plan* and (iii) *modify plan*. Of course, planners differ in how they achieve each of these steps; that is, in what methods they use to perform these three tasks. These differences also reflect their choices in terms of how planning knowledge is represented. Fig. 2 shows a set of PSMs we have found in the literature to realize these tasks[3]. The tasks presented in Fig. 2 can be described as follows:

Propose expansion has the goal of generating a new step in the planning process in order to achieve a selected goal. The input knowledge roles for this task are: world description[4], plan structure and plan assessment knowledge. To realize this task, we found a method called *propose I*, which can be decomposed into three sub-tasks: *select goal*, *select operator* and *test for unachieved goals*.

[3]It is interesting to note that the idea of proposing a single high-level method for a class of problems is not new. (Chandrasekaran 1990) proposes a similar method, also called propose-critique-modify, to solve design problems.

[4]Saying that this knowledge role is an input implies that the knowledge roles that have a "part-of" relation with this role, are also inputs for the task.

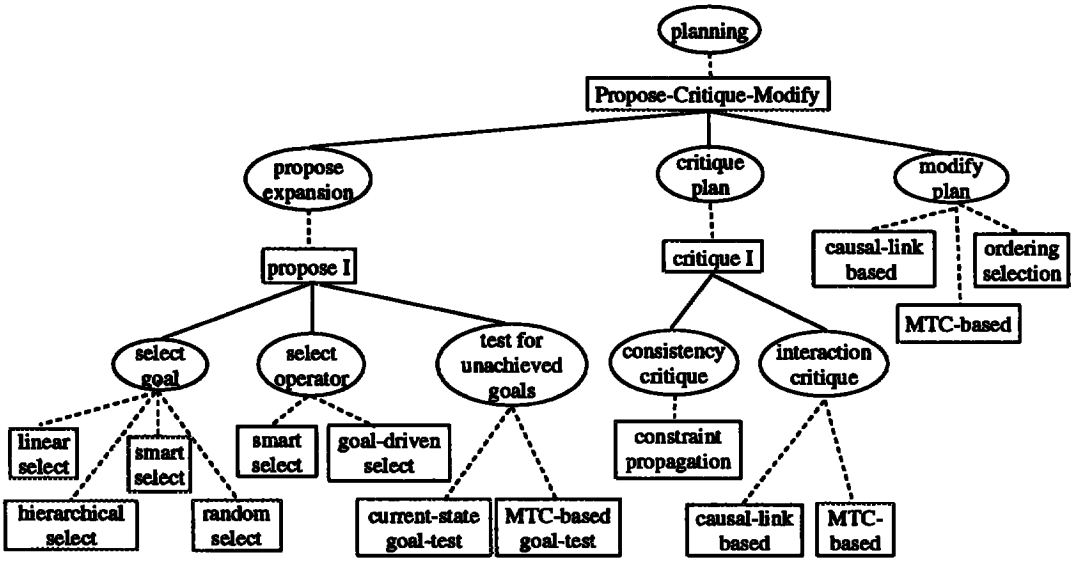

Figure 2: A task-method decomposition structure for planning.

Select goal This task selects a goal from the set of goals to be achieved. For *select goal* four methods can be used: *linear select*, *hierarchical select*, *random select* and *smart select*. STRIPS chronologically selects the last goal established by its search strategy (Fikes & Nilsson 1971), which we call *linear select*. NONLIN (Tate 1977) selects goals depending on some hierarchical levels assigned on the preconditions. SNLP (McAllester & Rosenblitt 1991) randomly selects goals from the set of pending goals. It is also possible to select a goal in a more intelligent manner which minimizes possible future modifications (smart select) (Drummond & Currie 1989). In this case, the method uses the static role plan assessment knowledge to select a goal. An example of smart select was proposed by (Kambhampati, Knoblock, & Yang 1995), where a goal-selection strategy is described called MTC-based goal selection, which selects a goal only when it is not necessarily true according to the modal truth criterion.

Select Operator *Select operator* takes the selected goal and the plan-step for which the goal is a precondition, and selects an operator. This can be a new operator which is included in the plan. But it can also be an operator which is already in the plan, in which case an ordering constraint is added in such a way that the operator is before the plan-step that needs the goal. When a new operator is included in the plan, its preconditions are added to the set of goals. The select operator task can be realized by two methods. The *goal-driven select*, selects an operator whose effect includes the selected goal, constraining the order to be necessarily before the selected goal. This method is used by SNLP and REFINEMENT SEARCH planning algorithms. The *smart select* method is applied by STRIPS and PRODIGY which are implemented by means-end analysis, and exploits the difference between the goal state and the current state to select an operator.

Test for unachieved goals This task checks the current plan for unachieved goals, and records them in the dynamic role goal. It also tests if the preconditions (sub-goals) of an operator are already achieved in the current state (when the plan composition is total-order). We identified two methods to realize this task: the MTC-*based goal-test* and the *current-state goal-test*. Causal-link based planners do not realize this task, since they ensure that goals once achieved, are preserved.

Critique Plan The task *critique plan* checks for conflicts and the quality of the plan generated so far, using plan assessment knowledge. The role plan assessment knowledge can point to "hard" constraints (plan interaction and the satisfiability of the plan constraints) and "soft" constraints (the factors that define when a given plan is better than another. We identified one method to realize this task which we called *critique I*. This method can be decomposed into two subtasks: *consistency critique* and *interaction critique*.

Consistency critique This task checks the consistency of the constraints imposed on the plan generated so far. The constraints concern the action ordering and the co-designation of variables. More complex planning systems can also check the consistency of the assigned resources and agents. We identified the *con-*

	Static role	Domain knowledge	Task	Realizing method
STRIPS	world description plan composition	ADL-based total-order	select goal select operator test for unachieved goals	linear select smart select current-state goal-test
PRODIGY	world description plan composition	ADL-based total-order	select goal modify plan select operator test for unachieved goals	random select ordering-selection smart select current-state goal-test
NONLIN	world description plan composition plan assessment	HTN-based partial-order causal-link protection	select goal select operator critique plan modify plan	hierarchical select goal-driven select causal-link based causal-link based
TWEAK	world description plan composition plan assessment	ADL-based partial-order truth-criterion	select goal select operator critique plan modify plan test for unachieved goals	random select goal-driven select MTC-based critique MTC-based modification MTC-based goal-test
SNLP	world description plan composition plan assessment	ADL-based partial-order causal-link protection	select goal select operator critique plan modify plan	random select goal-driven causal-link based causal-link based

Table 1: Mapping of some main planners into the proposed framework. Listed are the static roles used, the domain models which fill these roles ("ADL" stands for "add-delete list"), the tasks and their realizing problem-solving methods.

straint propagation method to realize this task.

Interaction critique When checking for conflicts, this task verifies whether the selected operator for achieving the goal would interact with other goals in the plan, and, whether the other operators already in the plan would interact with the intended goal to achieve. Although these two verifications seem the same, the second checking is sometimes included in the task *select operator*. Note that this task involves explicit reasoning about interactions. For realizing the interaction critique task, we identified two methods: (i) the *causal-link-based critique*, which checks if the plan-step interacts with the conditions of the causal links; and (ii) the MTC-*based critique*, which uses the modal truth criterion to check the existence of a step that possibly interacts with the selected goal.

Modify plan The *modify plan* task is responsible for modifying the plan with respect to the results of the critique plan task (a conflict). By using plan assessment knowledge a modification can be done by adding ordering, binding or secondary preconditions to the plan until the possible conflict (violation) is solved. We identified three methods to realize this task: the *causal-link-based*, the MTC-*based* and the *ordering-selection* plan modification. The causal-link-based method adds causal links to the plan which prevent the conditions contained in conflict from being undone. The MTC-based method adds constraints to the plan in such a way that the conditions in conflict are necessarily true. Both methods are used in partial-order planners. The ordering selection is used for total-order plans to select a point in the plan where to insert an operator such that the conflict can be avoid.

Applying the Framework

In the previous sections, we proposed a framework for modeling (at a high level) all methods used in planners. This implies that it should be possible to describe a planner on three dimensions: (i) the static knowledge roles it uses (Fig. 1), (ii) a set of domain models to fill these roles, and (iii) a function-structure generated through the selection of some tasks and methods from Fig. 2. In this section, we will justify the above claim by showing this mapping for a number of well-known planners.

Mapping Planners into the Framework

Table 1 shows how our framework characterizes a number of important planners by mapping their choices in each of the three dimensions (static knowledge roles, domain knowledge, tasks and methods). The table contains the mappings for five different planners: STRIPS (Fikes & Nilsson 1971), PRODIGY (Carbonell & the PRODIGY Research Group 1992), NONLIN (Tate 1977) SNLP (McAllester & Rosenblitt 1991), and TWEAK (Chapman 1987).

The second column of Table 1 gives the static roles used in the planners. All planners use world description

and plan description, but only the planners that explicitly reason about interactions use the plan assessment knowledge role. The third column shows that STRIPS and PRODIGY are total-order planners, whereas the other three are partial-order planners. Except for NONLIN which is a HTN based planner, all planners use ADL operators (third column). As can be seen in the fourth column, partial-order planners use dedicated tasks for critiquing and modifying the plan (total-order planners solve this by backtracking). Due to space limitation, we refer to the fifth column of the table for the different PSMs used to realize the tasks.

Based on the table, we can identify different types of planners depending on how they relate to the following characteristics: linear or non-linear; total-order or partial-order; truth-criterion or causal-link-protection. In the following, we will concentrate on the most interesting types which are also discussed in the planning literature.

Total-order/linear (STRIPS). Such planners use a total-order domain model for the role plan composition; a *linear select* method for the goal selection task; the *current-state goal-test* method for the task "test for unachieved goals"; the *smart selection* method for the task "operator selection"; and they do not use plan assessment knowledge.

Total-order/nonlinear (PRODIGY). Such planners use a total-order domain model for plan composition; the *random select* method for the select goal task; the ordering-selection method for modify plan task and the *current-state goal-test* for the test for unachieved goals task. They usually do not use the role plan assessment knowledge.

Partial-order/truth criterion (TWEAK, NONLIN). Such planners use a partial-order domain model for plan composition; the MTC-based domain model for plan assessment knowledge; the "test for unachieved goals" task with the MTC-*based* method. They do not make use of causal-links for bookkeeping.

Partial-order/causal-link-protection (SNLP). Such planners use the causal-link dynamic role to keep track of how the goals have been established; the causal-link-based domain model for filling the static role plan assessment; they use the tasks "critique plan" and the "modify plan", but not "test for unachieved goals".

An Example: Mapping TWEAK For illustration, we detail here how TWEAK can be mapped onto the proposed framework. TWEAK is a partial-order planner that solves conflicts by explicitly reasoning about them and modifying the plan. Fig. 3 shows its function-structure, where individual functions or tasks are represented as rounded rectangles, with their input and output roles represented as normal rectangles. Arrows indicate the direction of knowledge flow (thin arrows connect dynamic roles, and bold arrows static roles). In this function-structure, the current plan (in the initial situation it contains just the goals to be achieved) is tested to see whether there are unachieved goals. This task updates the role goals. If there are unachieved goals, one of them is selected and with respect to that one an operator is selected, either from state changes (which is part of the role world description), or from the proper plan. This results in an expanded plan, which is then tested for conflicts. If they are found some modification action is required. "Test for unachieved goals", "critique plan" and "modify plan" need the role plan assessment knowledge, and "modify plan" additionally needs the roles world description and plan structure.

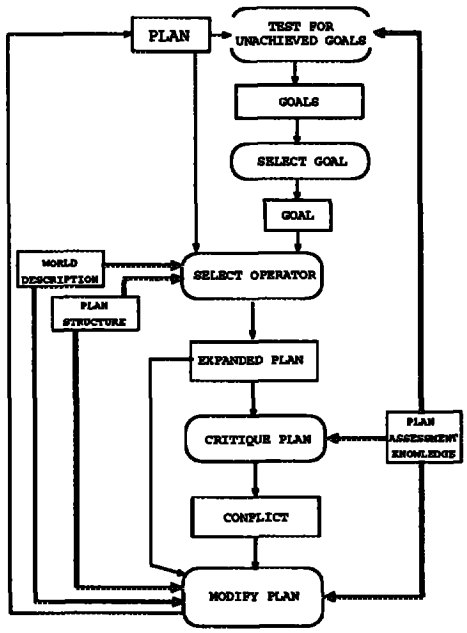

Figure 3: A function-structure for TWEAK according to the proposed mapping. Rounded rectangles represent tasks (or functions) and normal rectangles roles. Bold lines connect static roles and thin lines dynamic roles.

Below we give as an example the control structure of TWEAK that has to be applied on its function-structure (Fig. 3). Static roles are denoted by their abbreviated names: PAK for plan assessment knowledge, WD for world description, PS for plan-structure. The repeat-until loop in the critique-plan and modify-plan is due to the fact that for each modification, other conflicts could show up, and they have to be taken care of.

control-structure TWEAK

```
1 test-for-unachieved-goals(plan, PAK → goals);
2 if goals = ∅ then exit;
3 else select-goal(goals → goal); backtrack point
4  select-operator(plan, goal, WD, PS → expanded-plan);
5  repeat
6     critique-plan(expanded-plan, PAK → conflict);
7     modify-plan(conflict, expanded-plan, PAK,
         PS, WD → plan); backtrack point
8  until conflict = ∅
9  goto 1;
```

Conclusions

In this paper, we presented a knowledge-level framework for characterizing classical planners, including partial and total order planners. The framework analyzes planners in terms of the domain knowledge and problem-solving methods they use, and helps in understanding the differences and commonalties between the various planners.

Although our analysis has a different starting point than that of (Kambhampati, Knoblock, & Yang 1995), some of the objectives, and even their realizations, may be very similar. We provide a general knowledge-level (KL) model that unifies classical planners, while Kambhampati et al. provide a generalized algorithm (called refinement search), in which so-called "plan-space" planners can be expressed by instantiating this algorithm in different ways. Their claim is that a unified view facilitates separation of important ideas underlying individual algorithms from "brand-names", and thus provides a basis for understanding the design tradeoffs and for fruitfully integrating the various approaches. A detailed comparison between the two frameworks is outside the scope of this paper. However, it is important to stress that the main difference is the goals of the work: while (Kambhampati, Knoblock, & Yang 1995) provide a very detailed (and sometimes very difficult to understand) framework in order to be able to compare the performances of the planners, our emphasis is in providing a more abstract framework that can be used as an explanation of the differences between these planners, mostly for knowledge engineering purposes.

Because our analysis uses the CommonKADS methodology, we inherit its benefits. We identified the relevant domain knowledge and reasoning steps in various planners. For building a particular planner, one can select the appropriate problem-solving methods and fill their knowledge roles with the domain knowledge at hand. The knowledge acquisition process is thus supported. Another benefit is that the task-method decomposition structure (Fig. 2) can be considered as a high-level specification for an architecture that enables opportunistic planning. That is, a meta-reasoner could dynamically select which method to use for realizing a particular planning task. In this way, different planning methods can be interleaved to construct more intelligent planners.

References

Carbonell, J., and the PRODIGY Research Group. 1992. PRODIGY4.0: The manual and tutorial. Technical Report CMU-CS-92-150, School of Computer Science, CMU.

Chandrasekaran, B. 1990. Design problem solving: A task analysis. *AI Magazine* 11:59–71.

Chapman, D. 1987. Planning for conjunctive goals. *AI* 32:333–377.

Drummond, M., and Currie, K. 1989. Goal ordering in partially ordered plans. *IJCAI* 960 – 965.

Drummond, M. 1994. Precondition achievement planning: framework *sans* application? In Wilkins, D., ed., *Proceedings of the AAAI Workshop on Comparative Analysis of A I Planning Systems*, 1–9.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2.

Kambhampati, S.; Knoblock, C.; and Yang, Q. 1995. Planning as refinement search: a unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence* 76. special issue on planning and scheduling.

Marcus, S., ed. 1988. *Automatic knowledge acquisition for expert systems.* Boston: Kluwer.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proc. of AAAI-91*, 634–639.

Minton, S.; Bresina, J.; and Drummond, M. 1994. Total-order and partial-order planning: a comparative analysis. *Journal of Artificial Intelligence Research* 2:227–262.

Schreiber, A. T. 1992. The KADS approach to knowledge engineering. editorial special issue. *Knowledge Acquisition* 4(1):1–4.

Tate, A. 1977. Generating project networks. In *Proceedings IJCAI-77, Paris.*

Valente, A. 1995. Knowledge level analysis of planning systems. *SIGART Bulletin* 6(1):33–41.

Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann Publishers, Inc.