Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

**Research
Report**
RR-94-06

An Adaptive Deductive Planning System

Dietmar Dengler

February 1994

**Deutsches Forschungszentrum für Künstliche Intelligenz
GmbH**

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341

An Adaptive Deductive Planning System

Dietmar Dengler

DFKI-RR-94-06

This work has been supported ba a grant from The Federal Ministry for Research and Technology (FKZ ITWM-9000 8).

ISSN 0946-008X

An Adaptive Deductive Planning System

Dietmar Dengler
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
phone: +49 (681) 302-5259
fax: +49 (681) 302-5341
e-mail: dengler@dfki.uni-sb.de

Abstract

A generic planning system is introduced which allows for custom building of planners able to generate plans for different plan consumers in the context of intelligent support systems. All planners are adapted to the pecularities of different plan consumers, to their domain knowledge, their typical behavior, their preferences, and their utilization of plans.

The necessary knowledge sources of the generic planner are fixed in order to enable it to produce plans of a certain specificity. Its control strategy is described in a formal specification language containing constructs which allow for the configuration of characteristic parts of the control strategy. The customized planners are defined by executable specifications.

An application of the approach to deductive planning based on a modal temporal logic is shown. It is shown in an example how needs of different plan consumers in an intelligent help system can be met by a deductive planner.

Contents

1 Introduction

As the complexity of interactive computer software increases in terms of large amounts of information to be conveyed and a wide range of task structures, intelligent user support systems become necessary to guide and assist users in accomplishing their tasks, e.g., by instructing them how to make full use of the functionality of the software. The aim is to give the user support depending on the context and relative to his knowledge and experience. The tasks of a planner in such a context are multi-faceted: it must be able to generate plans which serving as a basis for the monitoring of the user's performance and representing user-oriented optimal solutions for recognized suboptimal behavior. The plans are the input to automatic execution facilities or may serve as the basis for a tutoring or advice-giving component. Thereby, the results of the planner have to fulfil different quality requirements with respect to specificity and resource consumption and must be adapted to the current needs of the particular *plan consumers*, e.g., to their domain knowledge, their capabilities, preferences, and behavior. Additionally, tutorial aspects and the computational costs of planning must be considered.
To tackle the problem of generating plan variants and to serve all types of plan consumers effectively requires high flexibility in the planner. In our approach we take as a basis a generic planning system from which a wide range of particular planners can be customized appropriate to generating plans of a certain quality. The control strategy of the planner is described in a formal specification language containing constructs which allow for the configuration of characteristic parts of the control strategy.
We have realized our approach by building a deductive planner based on a modal temporal logic. Since planning is done there by proving formal plan specifications, adaptation means to guide the proof process accordingly and to restrict the knowledge base of the logical calculus used.

Aspects of intelligent user support systems which justify the approach of an adaptive planning system are mentioned in section 2. Section 3 describes necessary knowledge sources for the planner according to the application domain considered. In section 4 the approach of how to realize the configuration of a planner is explained. Finally, an application of the approach to deductive planning is described in section 5.

2 Intelligent User Support

A scenario where the flexibility and adaptability of a planning system is actually needed occurs in the context of intelligent help systems for application software (cf. [Tat92], [BBD$^+$93]). Help systems are required both to instruct users in how to operate software and to promote some understanding of the workings of the application system. The users of interactive software systems differ in the fact that they pursue different tasks, and have different experiences, preferences, and levels of knowledge. Support provided by the system should therefore be adapted to the individual user [Ben93].
A planner can be seen as a central component of the support system since all activities of this system which concern plans or procedural processes are connected with the planner. Figure 1 emphasizes the position of a planner in an intelligent user support environment. The output of the planner must be adapted to the requirements of certain plan consumers. The following consumers can be fixed:

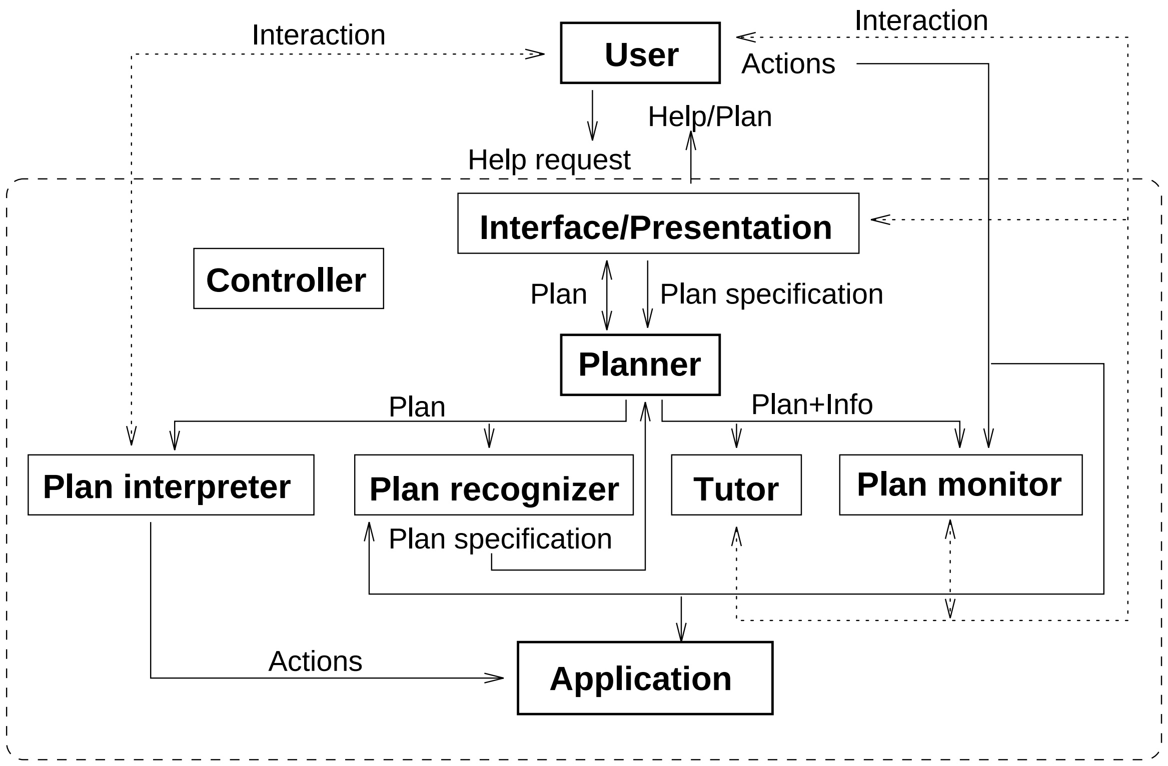

Figure 1: Plan flow in a user support environment

- The user of the application system; he asks for concrete or abstract plans, would like to have an optimized plan for a given suboptimal one, or wants the planner to verify the correctness of a plan he proposes.
- A reasoning system as a part of the support system, e.g. a plan recognizer needs plans as a basis to observe the user.
- A plan interpreter; it tries to execute plans automatically and the plans don't need to contain any user-specific overhead. For example, it is initiated by a request of the user who wants that a specific goal is reached, or by an offer of the plan recognizer for semantic plan completion. That component can transform abstract plans, e.g. containing loops, into concrete sequential plans. Thereby, it can happen that an interaction with the user is necessary, e.g. if the truth value of a condition in a complete case analysis cannot be decided with respect to the current domain axioms.
- Either an adice-giving or tutor component; plans at various levels of detail and difficulty are needed to provide appropriate help to the user. Additionally, the usage of user-unknown domain knowledge must be under complete control and justifications for the introduction of new material to the user must be available.
- An execution monitoring component; if the user has asked for a concrete plan reaching his current goal which was also presented to him then a monitoring process can start which supervises the execution of the plan. The user, now executing his plan, can opportunistically interrupt his plan or can interleave it with the execution of another plan. The plan used for monitoring should therefore be abstracted in some way, e.g. it

should allow safe interruptions. A user's try to disturb the safe execution of the plan can cause an active help action.

Various possible requirements for plan quality are listed below:

1. A plan should reach its goal with limited resources, e.g. with few basic actions, or there is a limit on resources which should not be exceeded, e.g. the knowledge extension potential of a user regarding tutorial aspects of a plan. That is a point where general optimization techniques can be used during planning to guide the search appropriately. Dependent upon the complexity of the domain and the current requirement on the rank of reached optimality different methods can be used (cf. e.g. [Kor90, Kor93]).

2. There are restrictions and/or preferences with respect to the actions and control structures occurring in a plan. For example, if plans has to be generated as a solution to a goal-directed request of a user then the look of the plan should consider the user's knowledge and level of expertise about the domain, and his usual utilization of actions. If he is a novice then he doesn't know all concepts and actions which are available. If the domain allows to reach certain goals by the use of different actions then the user has certainly some preferences deciding the choice between these actions.
   Concerning control structures it can be desirable from a tutorial point of view that a plan uses specific control structures which demonstrate relations between actions, e.g. the mutual independence expressed by nonlinearity.

3. A plan shows a certain degree of abstraction, or an increasing abstraction corresponding to the temporal course of the plan, e.g. a procedural advice can be better understood if it is detailed at the beginning and more abstract at the end [GJ90]; a plan recognizer prefers an abstract plan as a hypothesis for the user observation since it describes a large class of concrete plans in a compact way.

4. A plan contains supporting actions (e.g. information-providing actions) adapted to the knowledge of the consumer. If plans has to be generated where their use of actions should be transparent for the consumer then additional actions can be necessary. They reflect the basis of decisions in choice making with respect to domain objects and their properties.

5. A plan has been generated using a certain problem solving strategy which is mirrored by structural properties of the plan, e.g. the grouping of subgoals in a conjunctive goal concerning the same objects can result in a grouping of actions in the resulting plan. Besides on a syntactic level, the structural properties can also be on a rather semantic level. The positive interactions between actions in a plan, i.e. one action produces as an effect the preconditions of another action, can be utilized since a corresponding relation between their specified goals was known and used to order subgoals (cf. [CI89, MP93]).

6. A plan considers in its look the previous plan behavior of a user. If there are some actions or subplans in the application domain which are mutually independent then a user can have some preferences about the order of their execution. Such habits can be considered by the generation of a user-specific plan. Futhermore, the use of specific actions or subplans by a user can be influenced by his personal view on the domain objects, i.e. if a domain object has a certain individual property then the user associates other plans with that object as it would be the case if the object shows not this property.

7. A plan is only a partial solution to a specified planning problem (i.e. the planner has to work incrementally). In a cooperative connection between a planner and a plan recognizer in an intelligent help system where the planners results are used as hypotheses for the observation of a user's plan behavior incremental planning can avoid unnecessary generation effort. From a set of delivered plans typically only a subset can be maintained as plan hypotheses after the next user action. So, if complete plans has been delivered there is an obvious generation overhead.

Considering the plan features just mentioned it is obvious that a kind of user modelling (cf. [Kob93, McT93]) is necessary in order to be able to produce plans which are adapted to the current user of the application system.
Pursuing the aim of providing intelligent support can also imply the necessity for interactive planning. A plan can be developed cooperatively with the user. The user can participate in choice making, e.g., since the planner has a lack of information which action to choose out of a set of alternatives. The user can also be given the opportunity to choose between alternative subplans whereby the planner forecasts the implications of each alternative.

3 Knowledge Sources

In order to generate plans which are adapted to a particular plan consumer, structuring and access methods with respect to the domain knowledge must be available.
The basic entities of plans are basic actions. The set of all available basic actions can be structured along different lines. Synonymous actions are collected to equivalence classes SC_i and preferences in such a class are expressed by a partial order $<_i$.
The family $BS = (\langle SC_i, <_i \rangle | i \in I)$, I an index set, describes a basic structuring of the set of actions. Classes SC_i can be compounded to equivalence classes AC_{R_z} with respect to a relation R_z. Again, these classes are structured by a partial order $<_{R_z}$ on their elements.
Instances of equivalence relations on **BS** relevant for a tutorial plan consumer, e.g. to extend pedagogically a user's action knowledge of the domain (cf. [Wol93]), are:

- reaching of exactly the same effects by presupposing the same preconditions (to express synonymous actions): $R_{synonym}$
- reaching common goals
- reaching similiar irreversible effects
- presupposing similiar hardly-reachable preconditions
- reaching the same number of effects
- causing similiar costs
- no significant relation

For I an index set, an indexed action set $A_{R,I}$ is defined as a family
$A_{R,I} = (\langle AC_z, <_{R_z} \rangle | AC_z$ subclass of $AC_{R_z}, z \in I)$.
By a structural change of I (and of R respectively) a specialized indexing can be defined:

- The elements of I are atomic goals; one gets a map of atomic goals to realizing actions.

- The elements of I are sets of atomic goals; an indexing of actions with respect to commonly reachable goal sets is specified.
- The elements are pairs (S_G, S_C), S_G, S_C sets of atomic goals; they represent conditioned indexes meaning that if goals S_G has to be reached by the simultaneous presence of goals S_C then $R_{(S_G,S_C)}$ specifies an equivalence class on a part of **BS** with respect to the goals S_G. For example, a conditioned index can be used to specify the usage of positive effects of some actions.

Control structures are an instrument to assemble basic actions to plans. One can differentiate between three sets: *explicit* control structures CS_e appear directly in the application domain, *conceptual* control structures CS_c are only implicitly contained in the domain (e.g. various forms of iteration), and *abstract* control structures CS_a occurring in abstract plans are often needed to communicate with non-human plan consumers. Preferences in the different sets are again expressed by a partial ordering.

Typical plan behavior can be considered as an extension of the action set structuring. Let P_G be an equivalence class of plans with respect to the relation "reaching the goal G", I an index set of pairs (G, C) with G a goal description and C a constraint, $<_{(G,C)}$ a partial order. The typical plan behavior is then described by $TP = (\langle P_G, <_{(G,C)}\rangle | (G, C) \in I)$. The constraints C can be both situation dependent (e.g. conditions about the current system state) and domain dependent conditions (e.g. user-specific conditions about the objects of the domain). Examples of typical plan behavior are e.g.

- The user wants to reach goals g_1 and g_2 while holding the constraint *true* (means situation independence). Then the user prefers to use a plan which first reaches the goal g_2 and then the goal g_1, i.e. he has a preferential order on goals and corresponding plans.
- A goal $g(x)$ should be reached while holding the constraint $\phi(x)$ describing a condition on the object x specified in the goal description. The user prefers to reach that goal by a plan $P_g(x)$ followed by an additional plan $P_\phi(x)$ which is not necessary to reach $g(x)$ but a reactive behavior of the user evoked by the condition $\phi(x)$.

Special indexes can be derived from **TP**, e.g. which action to use in a specific situation, or how to order subgoals to simulate the user's behavior.

There is some knowledge about the human-computer interaction in the application domain considered. The assumption is that the user has that information about the system state which he can see on the screen. Since some actions of the application domain can be parameterized the user must be aware of possible instantiations. For parameter instantiations which are domain inherent it is assumed that there are selection actions to access them. The planner has knowledge about how the user can access the missing information and how to generate plans which pay attention to that.

Besides different kinds of indexes there is problem solving knowledge in the form of heuristics for subgoal ordering and there is a collection of resource-specific optimization strategies.

4 Configuration of a Planner

We will now describe our approach to the configuration of a planner. The idea is to have a two-level specification for the control strategy of the planner. A generic planner is defined

by its specification in a specification language $\mathcal{L}_S$. Besides other constructs, $\mathcal{L}_S$ contains *choice points* cp_i as a language construct, whose possible instantiations are elements of $\mathcal{L}_S$. A choice point represents a place in a control strategy where planning-specific decisions are necessary. To abstractly describe the whole set of possible instantiations there is a language $\mathcal{L}_{CP_i}$ to specify a choice point. A term of this language is then part of a configuration description of the planner. A specific set of choice point descriptions is intended to describe the current requirements for plan quality which the planner has to reach. A concrete planner arises by replacing all choice points in the specification of the generic planner according to the configuration description. The planner then corresponds to a specification in a language $\mathcal{L}'_S$ which does not contain choice point constructs. We have chosen $\mathcal{L}'_S$ in a way such that specifications can easily be executed, i.e. planning is done by executing the specification of the planner. The customized planner is then a special-purpose planner able to generate plans of a specific quality.
The approach sketched above has the advantage that a customization of a planner can be automated and is even possible under certain circumstances during the execution of the planner, i.e. the planner can change its strategy during planning.

The specification language $\mathcal{L}_S$ is a tactic language in the tradition of metalogical frameworks (cf. [Pau89]) and is formalized by an algebraic specification, i.e. $\mathcal{L}_S$ is given by a term algebra $T_\Sigma(X)$ with signature Σ and variable set X:

$\Sigma =$ *sorts:* *specification, tactic, tactic_proc, string, ...*

opns: *basic_step*$_1$*,..., basic_step*$_n$*: specification specification* $\rightarrow$ *tactic*
true, false: tactic
then: list(tactic) specification specification $\rightarrow$ *tactic*
orelse: list(tactic) specification specification $\rightarrow$ *tactic*
if_tac: tactic tactic tactic $\rightarrow$ *tactic*
repeat: tactic specification specification $\rightarrow$ *tactic*
define_tac: string tactic specification $\rightarrow$ *tactic_proc*
call_tac: string specification $\rightarrow$ *tactic*
choice_tac: string $\rightarrow$ *tactic*
$\vdots$

eqns: $\forall$ *s:string A,C:tactic B,D:specification*
define_tac(s,A,B) $\wedge$ *define_tac(s,C,D)* $\rightarrow$ *A=C* $\wedge$ *B=D*
$\vdots$

The assumption is that a planning problem and its solution can be formalized in a certain logical language and they can be represented by a formal description of type *specification*. The basic entities of the tactic language are the constructs *basic_step*$_i$ which represent elementary planning tasks. It is assumed that the input/output behavior of these tasks is given by known functions. There exist constructs of this kind of more than the type structures mentioned here, e.g. to describe access to different knowledge sources.
There are some control structures on tactics, like sequencing, choice making, iteration, and recursion. A special construct is *choice_tac* which identifies an adaptable choice point in the control strategy. A choice point *choice_tac(s)* is described by a language $\mathcal{L}_{CP_s}$ defined maximally by the following algebra:

$$\begin{array}{lll}\Sigma_s = & \textit{sorts:} & \textit{feature} \\ & \textit{opns:} & f_1, \ldots, f_n\textit{: feature} \\ & & \textsf{\textit{is\_preferred\_to}}\textit{: feature feature} \rightarrow \textit{feature} \\ & & \textsf{\textit{then}}\textit{: feature feature} \rightarrow \textit{feature} \\ & & \textsf{\textit{and}}\textit{: feature feature} \rightarrow \textit{feature} \\ & & \textsf{\textit{or}}\textit{: feature feature} \rightarrow \textit{feature} \end{array}$$

A *feature* f_i abstractly denotes planning specific methods, strategies, or knowledge access. The various knowledge sources described in section 3 appear as features for choice points. For example, for a choice point *"choose an action to reach an atomic goal"* there can be features: take_user_action_knowledge, take_system_action_knowledge, consider_user_preferences,
take_action_avoiding_negative_interactions, etc.
The intended meaning of the operators of the choice point algebra is:

- f_i is_preferred_to f_j means deterministic choice: try first feature f_i, if not successful try f_j
- f_i then f_j means sequencing: use first f_i, if successful then use f_j
- f_i and $\wedge$ f_j means intersection: describes that only solutions are admitted that arise according to feature f_i and feature f_j
- f_i or f_j denotes a nondeterministic choice: decide nondeterministically to take one feature, if it cannot be successfully applied the other one is tried

There is a homomorphism $h_s : \mathcal{L}_{CP_s} \rightarrow T_\Sigma$ which translates abstract choice point descriptions into terms of the tactic language $\mathcal{L}'_S$.

Let $T_{\Sigma'}(X)$ be the subalgebra of $T_\Sigma(X)$ where only the operation *choice_tac* is missing. Then, a *planning tactic* **PT** is described by a set of terms of $T_{\Sigma'}(X)$ where:

1. $\mathbf{PT} = T \cup \{call\_tac(planner, S)\}$, where $T \subset \{\sigma(t_1, t_2, t_3) | \sigma = define\_tac\}$
2. For all $pt \in PT$ holds: there is subterm $pt' = call\_tac(s, S)$ of pt iff there is a term $define\_tac(s, U, S) \in PT$

A current planning problem is solved by evaluating a term *call_tac(planner,plan_specification)* into the standard boolean algebra. That may only result in *true* if the planner has worked successfully, i.e. a plan has been found which fulfills the plan_specification. The evaluation process is described by a homomorphism between $T_{\Sigma'}(X)$ and the boolean algebra and can be done by meta-interpretation.
A *configurable planning tactic* is described by a tupel

$$CT = (PT, \{(T_{\Sigma_{s_i}}, h_{s_i}) | s_i \text{ choice point in } PT\})$$

where PT a planning tactic with respect to $T_\Sigma(X)$. A *configuration description* is given by a set $CD = \{t_i | t_i \in T_{\Sigma_{s_i}}, choice\_tac(s_i) \text{ subterm of elements of } \mathrm{PT}_{CT}\} \cup Macro$,
$Macro$ is a set of macro symbols for choice point specifications.
A configurable planning tactic originates from an appropriate planning strategy refinement process done by the planning system engineer. This process can be supported by abstract

interpretation methods (cf. [CC92]) and the efficiency of the individual planners can be improved by partial evaluation and transformations (cf. [Fuc92],[Sah93]).

Configuration process

Given **CT** and **CD** a planning tactic **PT** has to be built. First, a complete and coherent configuration description is needed. There is a set of transformation rules which transform the set **CD** into a set **CD'**. Thereby macros are expanded, terms are rewritten, deleted, or added, e.g. since some simultaneously specified configuration parameters make no sense (incremental generation together with global resource-oriented optimization), or missing choice point specifications are supplemented using defaults. That process works as follows: Let $\{s_1, \ldots, s_n\}$ be the set of choice points in **CT**, $CD = \{sp(s_1), \ldots, sp(s_k), m_1, \ldots, m_l\}$ a configuration description, $m_i \in Macro$.
In a first step macros are expanded, i.e. there are rules of the kind

$$m_i\{sp'(s_1), \ldots, sp'(s_m)\}$$

If $m_i \in CD$ holds the application of such a rule works as:

- m_i is deleted from **CD**
- for all $sp'(s_i)$ the following is done: if there is a $sp(s_i) \in CD$ then it is deleted from **CD** and a choice point specification $sp(s_i)$ *is_preferred_to* $sp'(s_i)$ is added to **CD** otherwise $sp'(s_i)$ is added.

In a second step elimination rules are applied to the changed set **CD**:

$$SpSp' \ , \ Sp' \subset Sp$$

They work as follows: if $Sp \subset CD$ holds then Sp is replaced by Sp' in **CD**, i.e. particular specification parameters can be ignored, e.g. because they make no sense in a specific context.
In a last step default specifications for unspecified choice points can be added according to the application of rules:

$$\exists s_i : s_i \text{ choice point in } CT \text{ and } sp(s_i) \notin CDCD := CD \cup \{sp_{default}(s_i)\}$$

For all possible choice points s_i in **CT** there are predefined default specifications $sp_{default}(s_i)$. All the transformation rules just mentioned must have been predefined by the planning system engineer.

A set of rewrite rules RR specifying how *choice_tac*(s)-terms have to be replaced by the corresponding tactics can be inferred according to the known homomorphisms h_s. Applying RR on the configurable tactic in **CT** results in a specific planning tactic **PT**.

5 A Deductive Planning System

We will now sketch how the approach of an adaptive planning system fits into the field of deductive planning. The system in which our approach is embedded is called PHI (cf. [BDK92], [BBD+93]). It is a logic based tool for intelligent help systems and provides both a plan recognizer and a planning component which work in close cooperation. These two components considerably improve the performance of a help system. Regarding an active help mode—the help process is initiated by the system—the planning component can generate plans which serve as hypotheses for the observation task of the plan recognizer. Plan recognition in this sense means that the user's actions observed have to be matched against a set of plans which represent possible user behavior with respect to the application system. The plan recognizer is among other things able to propose semantic plan completion if it is highly probably that the user pursues a particular plan [BP93]. Furthermore, the planning component can generate more optimal solutions for recognized plans known to be suboptimal. Regarding a passive help mode the planning component can directly generate optimal plans.

The logic LLP [BDK92] is the basis for deductive planning. Plan specifications as well as plans are represented by certain classes of formulas in this logic. It is an interval-based modal temporal logic combining features of both traditional programming and temporal logics.
It provides the modal operators $\circ$(next), $\diamond$ (sometimes), $\square$ (always), and the binary modal operator ; (chop) expressing the sequential composition of formulas. Apart from these operators, control structures are also available, as in programming logics.
Our deductive approach to plan generation can be summarized as follows: given a formal plan specification formula which describes the relation of preconditions and effects with respect to the plan which has to be found, the plan arises from a constructive proof of its specification. Thereby, *plan specifications* are LLP formulas of the following form

$$[\textit{preconditions} \land \mathsf{Plan}] \rightarrow \textit{goals}$$

i.e., if the *preconditions* hold in a situation where we carry out Plan then we will reach the *goals*, a formula containing temporal modalities. Plan is a metavariable for a plan formula. Its instantiation is obtained if the constructive proof of the specification formula ended successfully.
The basic actions are given by axiom schemata where instances are able to describe the primary effects of actions as well as their invariants. Control structures of plans are represented by logical and temporal control structures defined in LLP. The knowledge about these structures is represented by sets of deduction rules handling the introduction and elimination of these structures.
Planning is done by proving formulas using a specific calculus for LLP, namely a sequent calculus. The tactic language introduced in the last section can be used in a straightforward way to guide the necessary proofs. The most important basic entity of the language $\mathcal{L}_S$ becomes *apply_rule* performing the application of a sequent rule. Most of the operators of $\mathcal{L}_S$ are in the style of tactical theorem proving systems, e.g. cf. [Pau89]. A current (deductive) planner is then given by the specification of a particular proof strategy for the underlying logic LLP.
The different indexes on knowledge sources mentioned in section 3 provide restrictions, preferences, and constrained access to the nonlogical axioms and rules of the sequent calculus used.

To demonstrate one aspect of consumer-specific planning we show in an example how supporting actions can be integrated in an abstract plan. The application domain where the example comes from is a subset of the UNIX operating system, namely its *electronic mail system.*
The planner has to generate abstract plans which serve as hypotheses for the observation

of the user by a plan recognizer. The plan specification is given by

$$\forall x \;\; pre(x) \wedge P \rightarrow \Diamond goal(x)$$

Abstract plans describe a class of concrete plans in a compact way. If the planner were to generate a concrete plan then it would do it as follows:
Two formulas must be proven, a modified plan specification

$$\exists x \;\; pre(x) \wedge P \rightarrow \Diamond goal(x)$$

and whether the plan is applicable in the current situation

$$pre(t), \qquad \text{for the substitution } \sigma = \{t/x\} \text{ from the proof of the specification.}$$

The prover has access to a complete description of the current situation and can "compute" the bounded range of values for the variable x. But, the plan should simulate the behavior of the user, i.e. the user must also be able to follow the value restriction of x. In general, this is only possible by executing additional *informative* actions. The plan generated must correspond to this behavior. Using our proving approach it means that the plan specification must be automatically extended by goals corresponding to appropriate informative actions. The extension must guarantee that the original specification is still valid.
It looks abstractly like

$$\forall x \;\; pre(x) \wedge P \rightarrow \Diamond(add\_goal(x) \wedge \Diamond goal(x))$$

and causes P to become a sequence P_{add_goal}; P_{goal} during planning.
The point is, that $pre(x)$ can contain a subformula expressing a value restriction on x which is however not a necessary precondition for a plan which fulfils the original specification.
Let's have a look at an example from the mail domain. The specification considered is:

$$\forall x : msg \;\; sender(x) = s \wedge delete\_flag(x) = 0 \wedge P \rightarrow \Diamond read\_flag(x) = 1$$

It describes that a message x, which is not deleted and has sender s, should be read. A simple plan which fulfils that specification is the abstract single action plan $EX(type(x))$ which only presupposes that x is not deleted.[1] But that plan doesn't simulate the behavior of the user, who has to know which message fulfils all specified preconditions. The plan could be used as a basis of a concrete plan for the consumer *automatic action execution*. However, the user usually initiates some additional informative action. The plan specification can be extended according to knowledge about human-computer interaction in the domain. Using the meta-knowledge

$$\forall x, y \;\; member(x, screen) \wedge type(x) = type(y) \rightarrow instantiation\_possible\_by\_user(y)$$

the specification becomes:

$$\forall x : msg \;\; sender(x) = s \wedge delete\_flag(x) = 0 \wedge P \rightarrow \Diamond(member(x, screen) \wedge \Diamond read\_flag(x) = 1)$$

A more reasonable instantiation for P is then the plan $EX(from(s));EX(type(x))$, where the *from* action just produces the missing information.

[1] EX is a predicate symbol which takes as its argument an action term. It describes in the underlying logic the execution of an action.

The plan can still be more sophisticated if typical user behavior is also considered. For example, it can be available that after the user has read a message, typically he may save it if it is interesting. Assuming that the system does not exactly know what defines the concept "interesting message" the instantiation for P can then be:

$EX(from(s));EX(type(x));EX(save(x,+x)) \vee EX(empty\_action)$.

The nondeterministic subplan appears since it is uncertain if the message is really interesting for the user.

Regarding an appropriate configuration of the planner in order to produce the different plans the choice point *work on subgoal* with respect to the last mentioned instantiation of P can be described as:

(*informative_action_intro* **then** *standard* **then**
(*plan_behavior_certain* **is_preferred_to** *plan_behavior_uncertain*)) **is_preferred_to** *standard.*

"standard" designates a simple backward-chaining strategy to find a plan with respect to the current subgoal.

A central control component of the support system can direct the configuration of a current planner according to requirements of the plan consumers. But it is also possible that the consumers themselves initiate a configuration.

Aspects of Incremental Planning

Our view on incremental planning is to let the planner only generate a prefix of a plan with respect to a given plan specification. Let $specification_{Plan}$ be a plan specification:

$$specification_{Plan} \equiv [preconditions \wedge Plan] \rightarrow goals$$

As an output of incremental planning we get an initial subplan $inst(Plan')$ and a specification for the rest of the plan:

$$specification_{Plan''} \equiv [preconditions_{Plan''} \wedge Plan''] \rightarrow goals_{Plan''}$$

The incremental process has divided $specification_{Plan}$ into two specifications $specification_{Plan'}$ and $specification_{Plan''}$:

$$specification_{Plan'} \equiv [preconditions \wedge Plan'] \rightarrow goals_{Plan'}$$

Plan is then the sequential compostion of $Plan'$ and $Plan''$ and $inst(Plan')$ an instantiation for the metavariable $Plan'$. The general problem is that it can be the case that $inst(Plan) \not\equiv inst(Plan');inst(Plan'')$, i.e. completely planning produces not the same result as sequential incremental planning, and in a worse case $inst(Plan'')$ cannot be found since $inst(Plan')$ could have some irreversible effects.

The essential task to be done is now the division of the whole problem into appropriate subproblems. In the case of conjunctive goals in the plan specification this means to order the subgoals in some way.

A plan recognition component as a consumer of incremental plans needs user-oriented plans as the basis for its observation process. The planner can then attempt to do the necessary ordering of subgoals with respect to the user's typical plan behavior. If no appropriate goal ordering information can be extracted then general domain-dependent knowledge for goal ordering can be taken. In a precompilation step the action axioms can be analysed to detect some relevant relations between individual goals (cf. [CI89, MP93]), e.g. it is known which realizations of subgoals produce positive or negative dependencies. One can also detect effects which are irreversible. Goals implying these effects can then be appropriately ordered in a given plan specification.

A choice point for subgoal division in the planner's control strategy must now be configured in a way that it considers the aspects just mentioned.

6 Related Work

Improving the usability of computer systems is an important research goal of human-computer interaction (cf. [Ben93]). Adaptive systems with their design variety help to approach this goal. In the case of a system adapting to a human, user modelling plays an important role (cf. [McT93, Kob93]). The system should be able to adapt to the individual characteristics and needs of its users. Human-computer interaction systems of that kind where a planner is an integrated part can be members of the following classes: intelligent help system (cf. [Bre90], [Tat92], [BBD$^+$93]), intelligent assistant system (cf. [GJ90], [Boy91], [SC92]), or intelligent tutoring systems (cf. [Nwa91], [EC93]). In these systems the planner's results have to be measured with respect to the user's ongoing task, his knowledge about the domain including his preferences, and his experience with the application system (cf. [Win90], [Kok91]). A plan is then optimal if it is both well adapted to the user's requirements and as short as possible according to the planner's ability.
With the integration of general optimization techniques in a planning system the complexity problem of planning must be considered (cf. [BN93]). Often, only approximate solutions are realistic, e.g. plan merging (cf. [YNH92]), transformational plan synthesis (cf. [Lin90]), and localization (cf. [Lan90]). It is desirable to place at the planning system engineer's disposal, a pool of optimization techniques from which the most promising candidate for the current application can then be taken.
Adaptation of the planning system to the domain model can improve its efficiency (cf. [Etz91], [MP92]), e.g. by automatically building appropriate indexes on the domain knowledge (see section 3).

Generic planning systems have the advantage that their flexibility and expressive clarity allow for the easy development, debugging, and maintenance of particular planners. A main feature of these systems is their strong separation between domain and control knowledge. There are some other approaches which partially consider aspects of a generic planning system.
TEST (cf. [RS90]) is a specification tool for planners. Its framework can be used as a prototyping tool for new planners. The approach relies on an analysis of planning as theory extension. Different planning systems can be achieved when different heuristics are used by the assumption manager in deciding which assumption to add to the knowledge base. This influences the way in which the spaces of possible courses of action are searched.
In the planning framework of Cranefield (cf. [Cra92]) it is possible to define various different temporal operators and corresponding inference rules for combining plans to form larger compound plans. Furthermore, one has complete control over the planning process since the planning strategy is defined by tactics.
The MRG system ([TCS92]) provides a general framework to handle plans. It uses a tactic language to represent basic actions, events, goals as well as activities, like plan generation,

plan execution, plan monitoring and interaction with the real world. A user of the system is able to use, combine, and integrate various planning techniques.
O-Plan2 ([Tat93]) is an agenda-based architecture providing different mechanisms to enable a planning and control system builder to select suitable implementation methods for describing choices, posting constraints to restrict choices, and triggering choices during planning.

7 Conclusion

We introduce a generic planning system which allows one to customize special-purpose planners easily for application in the intelligent support system context. Since the planners must be able to generate results which fulfil specific quality criteria we took the approach that a control strategy for the particular planners can automatically be configured out of an individually specified criteria adjustment with respect to the generic planner. All planners are defined by formal executable specifications. In this way the refinement of the generic planner can easily be supported by abstract interpretation methods and on the other hand the efficiency of the individual planners can be improved by partial evaluation and transformations.
The flexibility of the planning system makes it adaptable to the pecularities of different plan consumers. This was sketched by the application of the approach to deductive planning in the context of intelligent help systems where some aspects of consumer-specific planning have been described. Deductive planning is a straightforward application for the generic planning approach proposed but not the only one imaginable.

References

[BBD$^+$93] M. Bauer, S. Biundo, D. Dengler, J. Koehler, and G. Paul. PHI – a logic based tool for intelligent help systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambéry, France, August 1993. Morgan Kaufmann.

[BDK92] S. Biundo, D. Dengler, and J. Koehler. Deductive planning and plan reuse in a command language environment. In Neumann [Neu92], pages 628–632.

[Ben93] D. Benyon. Accommodating individual differences through an adaptive user interface. In M. Schneider-Hufschmidt, T. Küme, and U. Malinowski, editors, *Adaptive User Interfaces*, pages 149–165. North-Holland, 1993.

[BN93] C. Bäckström and B. Nebel. Complexity results for SAS$^+$ planning. In *Proc. of the 13 th IJCAI*, Chambery, 1993. to appear.

[Boy91] G. A. Boy. *Intelligent Assistant Systems.* Academic Press, London, 1991.

[BP93] M. Bauer and G. Paul. Logic-based plan recognition for intelligent help systems. In C. Bäckström and E. Sandewall, editors, *Proceedings of the 2nd European Workshop on Planning*, 1993.

[Bre90] J. Breuker. *EUROHELP Developing Intelligent Help Systems.* EC, Kopenhagen, 1990.

[CC92] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computations*, 2:511–547, 1992.

[CI89] J. Cheng and K. Irani. Ordering problem subgoals. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 1989.

[Cra92] S. J. S. Cranefield. A logical framework for practical planning. In Neumann [Neu92], pages 633–637.

[EC93] M. Elsom-Cook. Student modelling in intelligent tutoring systems. *Artificial Intelligence Review*, 7:227–240, 1993.

[Etz91] O. Etzioni. STATIC: A problem-space compiler for PRODIGY. In *Proc. of AAAI-91*, pages 533–540, Anaheim, CA, 1991.

[Fuc92] N.E. Fuchs. Specifications are (preferably) executable. *Software Engineering Journal*, 1992.

[GJ90] G.N. Gilbert and M. Jirotka. Planning procedural advice. *Interacting with Computers*, 2:313–329, 1990.

[Kob93] A. Kobsa. User modeling: Recent work, prospects and hazards. In M. Schneider-Hufschmidt, T. Küme, and U. Malinowski, editors, *Adaptive User Interfaces*, pages 111–128. North-Holland, 1993.

[Kok91] A. J. Kok. A review and synthesis of user modelling in intelligent systems. *The Knowledge Engineering Review*, 6:21–47, 1991.

[Kor90] R. E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.

[Kor93] R. E. Korf. Linear-space best-first search. *Artificial Intelligence*, 62:41–78, 1993.

[Lan90] A. L. Lansky. Localized representation and planning. In J. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 670–674. Kaufmann, San Mateo, CA, 1990.

[Lin90] T. A. Linden. Transformational synthesis: An approach to large-scale planning applications. In *Proc. of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 126–136, San Diego, CA, 1990.

[McT93] M.F. McTear. User modelling for adaptive computer systems: a survey of recent developments. *Artificial Intelligence Review*, 7:157–184, 1993.

[MP92] L. McCluskey and J. Porteous. Model adaption: an approach to tackling the efficiency problem in general planning systems. In *UK PSIG Proc.*, 1992.

[MP93] T.L. McCluskey and J.M. Porteous. Two complementary techniques in knowledge compilation for planning. In *Machine Learning Workshop Proceedings*, Amhurst, Massechusetts, USA, 1993.

[Neu92] B. Neumann, editor. *Proceedings of the 10th European Conference on Artificial Intelligence*, Vienna, Austria, August 1992. Wiley.

[Nwa91] H.S. Nwana. User modelling and user adapted interaction in an intelligent tutoring system. *User Modeling and User-Adapted Interaction*, 1, 1991.

[Pau89] L. Paulson. The foundation of a generic theorem prover. *Automated Reasoning*, 5, 1989.

[RS90] H. Reichgelt and N. Shadbolt. A specification tool for planning systems. In L. C. Aiello, editor, *Proceedings of the 9th European Conference on Artificial Intelligence*, Stockholm, Sweden, August 1990. Pitman.

[Sah93] D. Sahlin. Mixtus: An automatic partial evaluator for full prolog. *New Generation Computing*, 12:7–51, 1993.

[SC92] M.H. Sraner and S. Carberry. Generating tailored definitions using a multifaceted user model. *User Modeling and User-Adapted Interaction*, 2, 1992.

[Tat92] C. Tattersall. A new architecture for intelligent help systems. In C. Frasson, G. Gauthier, and G. I. McCalla, editors, *Intelligent Tutoring Systems: Proc. of the Second International Conference (ITS'92)*, pages 302–316. Springer, Berlin, Heidelberg, 1992.

[Tat93] A. Tate. The emergence of "standard" planning and scheduling system components - open planning and scheduling architectures. In C. Bäckström and E. Sandewall, editors, *Proceedings of the 2nd European Workshop on Planning*, 1993.

[TCS92] P. Traverso, A. Cimatti, and L. Spalazzi. Beyond the single planning paradigm: Introspective planning. In Neumann [Neu92], pages 643–647.

[Win90] R. Winkels. User modelling in help systems. In D. H. Norrie and H.-W. Six, editors, *Computer Assisted Learning: 3rd International Conference, ICCAL '90, Hagen, FRG, June 1990*, pages 184–193. Springer, New York, 1990.

[Wol93] U. Wolz. Providing opportunistic enrichment in customized on-line assistance. In W. Gray, W. Hefley, and D. Murray, editors, *Proceedings of the 1993 International Workshop on Intelligent User Interfaces*, pages 167–174. ACM Press, 1993.

[YNH92] Q. Yang, D. Nau, and J. Hendler. Merging separately generated plans with restricted interactions. *Computational Intelligence*, 8:648–676, 1992.