Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document
D-92-07

DFKI Workshop on Planning

Kaiserslautern, February 5, 1992

Proceedings

Susanne Biundo, Franz Schmalhofer (Eds.)

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
D-6750 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

DFKI Workshop on Planning

Susanne Biundo, Franz Schmalhofer (Eds.)

D-92-07

DFKI Workshop on Planning
Proceedings

Susanne Biundo and Franz Schmalhofer (Eds.)

Kaiserslautern, February 5, 1992

List of Participants

Elisabeth André
Mathias Bauer
Ansgar Bernardi
Ralph Bergmann
Hans-Jürgen Bürckert
Alastair Burt
Susanne Biundo
Andreas Dannenmann
Dietmar Dengler
Dirk Fedeler
Klaus Fischer
Christoph Globig
Matthias Hecking
Andrea Hemprich
Knut Hinkelmann
Christoph Klauck
Jana Koehler
Norbert Kuhn
Otto Kühn
Ralf Legleitner
Michael Marlburg
Gaby Merziger
Jürgen Müller
Bernhard Nebel
Thomas Rist
Franz Schmalhofer
Gabriele Schmidt
Andreas Schroth
Sonja Weber
Wolfgang Wilke
Otto Kühn
Gabriele Schmidt

Preface

This document contains the papers presented at the first DFKI workshop on planning. The workshop was organized in order to bring together people from different projects working on different planning issues. Almost all projects contributed a paper, indicating that planning currently plays a central role at DFKI.

There were thirteen talks in five sessions. Most of them were concerned with special aspects of planning, like representation, recognition, validation, generation, and modification. Two of the talks addressed planning in manufacturing and distributed planning, respectively.

One main point of discussion was devoted to different notions of planning: planning being the refinement of already existing skeletal plans (plan schemata) versus planning as the construction process of (instantiations of) these plan schemata. Important questions refered to feasibility and efficiency aspects of the resulting plans as well as to the efficiency and flexibility of the planning processes themselves.
As a result it seems worth to investigate the connections between these two different views of planning in order to find general concepts realized in both or to make methods used in one available for the other and vice versa.

Another observation was the increasing interest in plans containing control structures like conditionals and loops. They could be found in classical as well as in deductive planning approaches. According to the different views of planning the formation of these plans was addressed in different ways. Here exists a close connection to various program synthesis paradigms.

In summary, the workshop has provided a general survey of current work on planning at DFKI and thus constitutes an enabling factor for establishing tighter cooperations among the various projects. Future workshops about special planning issues should be envisaged as work proceeds in the particular projects.

We would like to thank the authors for their effort in contributing many interesting papers and we are greately indebted to all participants for many lively and fruitful discussions.

Susanne Biundo Franz Schmalhofer

Contents

Plan Representation

Andreas Schroth
AKA-MOD
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3
6600 Saarbrücken 11
Germany
e-mail: schroth@dfki.uni-sb.de

Abstract

The paper suggests a representation formalism for plans of autonomous cooperating agents in a changing environment. A model of case-based, opportunistic, hierarchical and reaktive planning is developed. A special feature of the approach is, that reasoning steps are viewed as actions and thus can be planned. A plan for cooperating forklifts in a loading-dock scenario is given as a small example.

1 Introduction

This paper suggests a representation formalism for plans of autonomous cooperating agents in a changing environment. A special feature of the approach is, that reasoning steps are viewed as actions and thus can be planned. The presented formalism will be further developed in the DFKI project AKA-Mod.

2 Aims of the Model

Cooperating agents act in a **changing environment**. On the one hand we have to assume that an agent's knowledge about the world is incomplete. This means that the world does not always behave as the agent anticipates. On the other hand the world state is sometimes affected by other agents without the agent directly realizing this. The agent has to recognize the changes later and to update his knowledge accordingly.
Under those circumstances an agent needs the ability of **reactive behaviour**. An agent's plan has to be flexible enough to permit him to react on unexpected conditions. The process of planning or reasoning in general must not take up so much of the agent's time that he can't react spontaneously on dangerous situations any more.
Nevertheless complex **plans** should be **represented explicitly**, (the agent's behaviour should not just be determined by independent rules,) so that the representing structure is available for inferences and moreover intentions concerning the plan can be represented.
Reasoning steps should be treated **as actions**. Then the reasoning process and especially the planning process can be planned, so that meta-planning (cf. [Wilensky83]) becomes feasible. This requires the reasoning process to be split up into modules which are able to change the agents knowledge. These reasoning actions can be executed opportunistically. For example more or less time-consuming inferences can be choosen depending on the time the agent has left, and resulting in more or less optimized plans.
Some examples for reasoning activities that should be treated as described are: plan generation, efficiency analysis of intended plans, plan optimization, conflict detection, deadlock control, plan reorganization after a failure, reasoning about causality and about resources, and inferences on knowledge. More examples can easily be found.
In a changing environment **hierarchical planning** (cf. [Wilkins84]) is appropriate. The agent makes available a not fully specified (i.e. abstract) plan for all of his goals. Parts of the plan which are likely to be executed soon are expanded (particularized) first, other ones not before the necessary information has been learned. In doing so the agent is able to specify his plan according to the actual situation.
Thereby **planning and plan execution are interlinked**, planning is done at execution time when this is opportune.
Since actions are expanded depending on contextual information and at the appropriate time, **opportunistic planning** is supported both for reasoning steps and

for 'normal' actions. (For the original approach to opportunistic planning refer to [Hayes-Roth85].)
According to the fundamental idea of **case-based reasoning** we treat planning as beeing based on standard plans. In the normal case the agent will know a plan to achieve his goals, he will not have to construct a plan from scratch. As a rule, the agent will retrieve a (possibly abstract) plan and specify or modify it appropriately to the circumstances. (cf. [Hammond89])
Altogether we gain a model of case-based, opportunistic, hierachical and reactive planning.

3 Plan Representation

Due to the limited space only a brief summary of the representation formalism will be given here. The formalism is conceptually related to [McDermott91].
<predicate>
refers to an expression that can easily be evaluated in respect of the agent's knowledge base. Complex evaluations should be formalized as actions (see below).
Special actions are writen as
(<action-name> {<parameter>}*)
and will be expanded from the agent's knowledge base.
Some examples are:
• (succeed) , (fail) ,
representing the always succeeding / failing action;
• (wait <time-period>) , (wait-until <predicate>)
i.e. suspend the plan's presently active branch and continue with another one beeing declared as an alternative by an **and**-construct (see below);
• complex evaluations should be executed as special actions
(evaluate <function>) ;
• (set <variable> <function>)
assigns a value to a variable being either local with respect to the plan or global in the agent's knowledge base;
• (achieve <predicate>)
calls the planner that will make available a plan to achieve the goal <predicate>.
Actions are combined to more complex actions by structuring constructs like:
• if <predicate> then <action> else <action>
enables conditional plans;
• or ({<action>}+)
chooses randomly or by means of a more complex strategy one of the <action>s;
• and ({<action>}+)
executes all of the <action>s in turn; if the currently active action is suspended by a `wait`-action, an alternative action becomes active;
• seq ({<action>}+) , simult ({<action>}+)
executes actions sequentially / simultaneously;
• loop while <predicate> <action> , loop until <predicate> <action>

repeats actions.
Agents with incomplete knowledge have to be able to reorganize their plan in case of a failing action. The failure of a subaction of a hierachical plan does not necessarily mean that every superordinate (more abstract) action has become impracticable and also will have to fail. Furthermore, a plan's abstraction hierarchy defines a causal relation among more concrete and more abstract actions. A cleaver error handling procedure will choose an appropriate level of abstraction within the hierarchical plan structure that constitutes the action to be replanned and (if possible) to be executed. We regard the knowledge that controls this selection as depending on and belonging to the plan and therefore represent it in the plan itself.
This is done by specifying an error handling strategy for appropriate actions.

```
    do <action1> if-fail <action2>
```

will catch the failure of a subaction of <action1>, execute the error-handling action <action2> and thereafter reenter the whole do-construct.

4 An Example

Imagine a loading-dock scenario where some forklifts have the task to transport some boxes from a store to a truck. Each forklift is assumed to have the top-level plan

```
    seq ( (ascertain Now-Situation)
          do (main-action)
             if-fail (ascertain Now-Situation)
                     %%% and thereafter reenter the do-construct
        ) .
```

`main-action` is expanded at planning time to the loop

```
    loop until (Now-Situation = Goal-Situation)
      seq ( (choose-some-boxes-to-transport-first)
            (search-one-of-these-boxes)
            (move-the-found-box-on-truck)
            (ascertain Now-Situation) ) ,
```

where the low-level actions will be expanded opportunistically at execution time. Any failure at this hierarchical level will result in a repetition of `main-action`, i.e. in updating the forklifts knowledge and then trying to execute the whole main-action loop again. A more sophisticated error handling strategy would be to continue with the most promising subaction of main-action, depending on contextual information. This behaviour had to be formulated with conditional failure treatment clauses, i.e.

```
    if-fail if <contextual-information> then <action> else (fail) .
```

References

[Hammond89] Hammond, K.J.: Case-Based Planning. Academic Press, Boston, 1989.

[Hayes-Roth85] Hayes-Roth, B.: A Blackboard Architecture for Control. Art. Int., 26, pp 251-321, 1985.

[McDermott91] McDermott, D.: A Reactive Plan Language. Yale CSD Report 864, 1991.

[Wilensky83] Wilensky, R.: Planning and Understanding. Addison-Wesley, Reading, MA, 1983.

[Wilkins84] Wilkins, D.E.: Domain-Independent Planning: Representation and Plan Generation. Art. Int., 22, pp 269-301, 1984

Knowledge Acquisition for Hierarchical Skeletal Plan Refinement

Otto Kühn Gabriele Schmidt
ARC-TEC
German Research Center for Artificial Intelligence (DFKI)
Postfach 2080
6750 Kaiserslautern
Germany
e-mail: kuehn@dfki.uni-kl.de schmidt@dfki.uni-kl.de

Abstract

Plans which were constructed by human experts and have been repeatedly executed to the complete satisfaction of some customer in a complex real world domain contain very valuable planning knowledge. In order to make this compiled knowledge re-usable for novel situations, it is suggested to construct a hierarchy of nested problem description modules of different grain sizes. With these problem description modules, hierarchically structured problem classes are defined so that appropriate skeletal plans can be associated. The thus selected skeletal plans have to be refined with respect to the actual situation. This approach is termed hierarchical skeletal plan refinement.

An analysis of the real world domain of mechanical engineering revealed that such problem classes and associated skeletal plans can be constructed for production planning problems. It was shown how hierarchical skeletal plan refinement can be accomplished with an integrated knowledge acquisition procedure that is supported by three coordinated tools. The first knowledge acquisition tool CECoS has been developed for delineating and defining hierarchies of problem classes from a number of selected cases and hierarchies of operator classes or partial plans from a number of concrete plan steps. The knowledge acquisition tool COKAM$^+$ is applied to acquire interactively formal descriptions of the operators and the appropriate task-related and common sense knowledge from text. With the tool SPGEN a skeletal plan with application conditions is then constructed for each problem class with an explanation-based learning procedure. These skeletal plans consist of a set of general operators and their dependencies.

1 Hierarchical Skeletal Plan Refinement for Production Planning in Mechanical Engineering

The problem of production planning in mechanical engineering consists of finding an adequate production plan for a given workpiece which is to be manufactured in some factory [TSR91]. For the manufacturing of a rotational part, the production plan specifies the sequence of chucking and cutting operations by which the workpiece can be manufactured. An adequate planning method for this problem must take into account the enormous complexity of this real world domain. Traditional planning methods, such as generating and testing various sequences of actions or pure hierarchical planning, are bound to fail due to the exorbitant number of possible operations and the various requirements which a good plan must fulfill. The planning method of hierarchical skeletal plan refinement not only meets the mentioned requirements, but at an abstract level it also reflects the expert's problem solving which typically consists of two phases: the selection of a skeletal plan and its subsequent refinement.
The selection of a skeletal plan is based on abstract features of the problem description and of the available resources specified in the environment and context description. The selected skeletal plan is an abstract sketch of the intended manufacturing process. An executable production plan is obtained by refining the skeletal plan with respect to the concrete data given in the problem and environment descriptions.
The general structure of the expert system which is being developed can be described by the model of expertise [BW89]. ¿From the concrete description of the workpiece and the available manufacturing environment more abstract feature descriptions are first constructed. These abstractions are then associated with a skeletal plan from a hierarchy that has been stored in the knowledge base. The skeletal plan is finally refined with the help of the workpiece and the factory description into the concrete production plan.
The model of expertise [KS92] specifies what kind of knowledge has to be acquired for the expert system, namely abstraction rules, refinement rules and hierarchies of skeletal plans which are associated with features of the problem description.

2 Integrated Knowledge Acquisition Method

An integrated knowledge acquisition method [SKS91] is developed to acquired the different knowledge which is relevant for solving the problem according to the model of expertise. The knowledge is acquired from three different sources of information (texts, cases and the expert's respective memories) with the help of three tools.
First the tool CECoS (Case-Experience Combination System) [BS91] is applied. Through an application of CECoS a hierarchy of problem or operator classes is delineated and formally defined.
COKAM$^+$ [Sch92] acquires preconditions and consequences from texts for the operators and the operator classes given by CECoS. Furthermore task-related engineering

and common sense knowledge can be obtained from texts which is then formalized step by step.
The formalized problem classes and feature descriptions obtained through CECoS and the formalized task-related engineering and common sense knowledge, preconditions and consequences supplied by COKAM+ can then be utilized to automatically construct skeletal plans and associated application conditions through the explanation-based learning procedure SPGEN (Skeletal Plan Generation Procedure) [SBKS91].

3 Acquisition of Problem and Operator Classes with CECoS

With the interactive tool CECoS [BS91] a hierarchically structured set of problem classes is obtained from a set of prototypical cases and human expert judgements. The problem classes are defined so that a useful skeletal plan will exist for each problem class. From explicit and implicit memories, the expert first establishes an extensional definition of the various problem classes with respect to selected prototypical cases. The so established production classes are then intensionally and thereby generally defined by feature descriptions given by the expert.
CECoS can also be applied for delineating and defining hierarchies of operator classes from a number of operators which are parts of a problem solution, i.e plan steps. At the leaves of the constructed hierarchy trees concrete operators are located. The nodes at higher levels are general operators which are abstractions of their related lower operators. The expert can term such general operators and also uses them during the problem solving process.

4 Acquisition of Operator Definitions with COKAM+

In order to acquire the knowledge with COKAM+ which is needed to generalize plans into skeletal plans with the tool SPGEN, it is useful not only to consider simple operators, but also the general operators or specific sequences of operators (often called macros in mechanical engineering).
For defining individual operators, each operator is presented to the expert. The expert then searches the informal knowledge base which he has interactively acquired from texts with COKAM+ and selects all knowledge units from the knowledge base which specify relevant preconditions and consequences of the particular operator. If relevant preconditions cannot be found in the informal knowledge, the expert is to add new knowledge units. This procedure is the best way to really find all the relevant preconditions and consequences. By combining theoretical knowledge from text with the expert experiences both gaps in the theoretical knowledge as well as gaps in the experts memories are likely to be discovered.
Explanations structures are acquired for the concrete operators as well as for the generalization of operators which are provided by CECoS by constructing different operator classes. The preconditions or consequences are related to the more general

operator, if they referred to all operators which are subsumed by this general operator. These definitions of generalized operators are essential for the construction of more or less general skeletal plans, which can be applied to different problem types in the hierarchy of problem classes.

5 Generation of Skeletal Plans with SPGEN

SPGEN is based on explanation-based generalization as described by [MKKC86]. The domain and common sense knowledge acquired with COKAM+ is thereby used as domain theory and the hierarchy of problem classes is employed to specify operationality criteria. Depending upon the selected problem class and the respective operationality criteria, a more or less general skeletal plan will be obtained from a given case.
A skeletal plan is constructed by SPGEN in four phases:
In the first phase the execution of the source plan is simulated on the basis of the available domain theory and explanations for the effects of the individual operations are constructed. If the domain theory is sufficient, a complete explanation of the plan will be obtained. The proofs that exist for the applicability of each operator can now be seen as an explanation of each effect that depends on operator attributes as well as world state attributes, from the initial or intermediate states.
In the second phase the generalization of these explanations is performed with respect to a criterion of operationality that specifies the vocabulary for defining abstract operators for the skeletal plan. The operationality criteria are provided by the features of the problem classes which were acquired from the human expert with the knowledge acquisition tool CECoS.
In the third phase, a dependency analysis determines which previous operations (or givens in the initial state) achieved the prerequisites for the subsequent plan steps. The skeletal plan thus accounts for the interactions between the various concrete operations of the plan at an abstract level.
In the forth phase the descriptions for the abstract operators of the skeletal plan are formed by collecting and normalizing the important constraints for each operation that were indicated by the dependencies. The dependencies referring to the problem description (the mold, the goal workpiece or the manufacturing environment) specify the class of problems for which the skeletal plan can be used, i.e. they define application conditions for the skeletal plan. The skeletal plan itself consists of a set of operator classes and dependencies which specify the possible sequences in which the operators may be applied.
The skeletal plans and application conditions constructed with SPGEN, provide a combination of knowledge-based and heuristic abstractions of a concrete plan. For novel problems, which satisfy the application conditions, the skeletal plan will provide a knowledge-based partitioning of the novel problems into appropriate subproblems, which can then be solved more easily.

References

[BS91] R. Bergmann and F. Schmalhofer. Cecos: A case experience combination system for knowledge acquisition for expert systems. *Behavior Research Methods, Instruments and Computers*, 23:142–148, 1991.

[BW89] Joost Breuker and Bob Wielinga. Models of expertise in knowledge acquisition. In Giovanni Guida and Carlo Tasso, editors, *Topics in Expert System Design, Methodologies and Tools*, Studies in Computer Science and Artificial Intelligence, pages 265 – 295. North Holland, Amsterdam, 1989.

[KS92] Otto Kühn and Franz Schmalhofer. Hierachical skeletal plan refinement task and inference structures. In *proceedings of the 2nd KADS user meeting, Siemens AG Munich, February 17 - 18*, 1992.

[MKKC86] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1:47–80, 1986.

[SBKS91] Franz Schmalhofer, Ralph Bergmann, Otto Kühn, and Gabriele Schmidt. Using integrated knowledge acquisition to prepare sophisticated expert plans for their re-use in novel situations. In Thomas Christaller, editor, *GWAI-91 15th German Workshop on Artificial Intelligence*, pages 62 - 71. Springer-Verlag, 1991.

[Sch92] Gabriele Schmidt. Situated knowledge acquisition from text in a complex domain, June 1992. to appear in Proceedings of the 5th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems.

[SKS91] Franz Schmalhofer, Otto Kühn, and Gabriele Schmidt. Integrated knowledge acquisition from text, previously solved cases and expert memories and expert memories. *Applied Artificial Intelligence*, 5:311 - 337, 1991.

[TSR91] Jörg Thoben, Franz Schmalhofer, and Thomas Reinartz. Wiederholungs-, varianten- und neuplanung bei der Fertigung rotationssymmetrischer Drehteile. DFKI Document D-91-16, DFKI, 1991.

Iterative Plan Recognition

Matthias Hecking
Mathias Bauer Gaby Merziger
PHI
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3
6600 Saarbrücken 11
Germany
e-mail: hecking@dfki.uni-sb.de

Abstract

Help systems support the users of application systems. This support can be considerably improved if help systems are provided with plan recognition and plan generation capabilities. Most of the known plan recognizers work with plan libraries. In the PHI project, we are combining plan recognition and plan generation to realize a more flexible way of plan recognition. A sequent calculus the basis for the recognition and generation. In this paper, we show how the iterative plan recognition process is structured. The consistency test is used to determine whether a plan hypothesis is still valid after processing the current observation.

1 Plan Recognition in Help Systems

Help systems aim at supporting users of application systems, e.g., cf. [NWW92], [HKN+88]. This support can be considerably improved if help systems are provided with plan recognition and plan generation capabilities. In this context, *plan recognition* is used to identify the users goals and thus forms the basis for providing active help (cf. [Fin83], [Hec87]).

There are many implemented plan recognizers in various domains and considerations of different plan recognition issues (e.g., [Fin83], [Gen79], [Hec87], [HKN+88], [Let88], [Lit85], [Moo88], [Pol86], [SSG78], [SI81]). One of the first theories of plan recognition was H. Kautz logical theory of plan recognition (cf. [Kau87]). Other theoretical approaches are based on attribute grammars (cf. [HH87]) or situation semantics (cf. [Wob88]). M. Vilain (cf. [Vil90]) shows one possible method for connecting the recognition of plans through context-free parsing methods with the theory of H. Kautz.

Whereas previous approaches have worked with separated plan recognition and plan generation components, the aim of the PHI[1] project (**P**lan-based **H**elp Systems, cf. [BBD+91]) is to realize different *cross-talk modes* between both.

Plan recognition as well as planning is done in a *deductive* way and is based on the **L**ogical **L**anguage for **P**lanning (LLP). A sequent calculus forms the basis for the deduction. The different application systems that should use the PHI kernel force the plan recognition component to work iteratively.

2 Iterative Plan Recognition

To realize a general plan recognition system for different applications the plan recognizer of the PHI system must work incrementally, i.e., after each received command the recognizer must determine whether the valid plan hypotheses are still valid or not or whether a hypothesis is completely recognized. This means, that the plan recognition process is an *iterative process for selecting plan hypotheses that account for the observed actions.* After each recognition step the following results can be obtained: (1) If the hypothesis accounts for the command, the command becomes an element of the already recognized parts of the hypothesis, and parameters are bound in the not yet recognized actions of the hypothesis, (2) if the hypothesis doesn't account for the command the hypothesis is not valid any longer, or (3) if the hypothesis accounts for the command and there are no unrecognized actions left, than the hypothesis is successfully recognized.

At the beginning of the iterative process a set of possible plan hypotheses $\triangle_0$ is provided by the plan generation component. Together with the observed action $\texttt{EX(command}_1\texttt{)}$ the plan recognizer determines the set of hypotheses $\triangle_1$ so that every member of $\triangle_1$ accounts for the observed commands, or more formally: $\triangle_0 \cup \{\texttt{EX(command}_1\texttt{)}\} \vdash_{\mathrm{PR}} \triangle_1$ ($\vdash_{\mathrm{PR}}$ means that plan recognition specific inferences

[1]This work has been supported by a grant from The Federal Ministry of Research and Technology (FKZ ITW-9000 8)

are used). If a sequence of observations $\mathtt{EX(command_1)};\ldots;\ \mathtt{EX(command_n)}$ must be processed, the recognition process can be abstractly described as follows ($\mathtt{O^i}$ means that the command is executed in the $\mathtt{i}$-th state):

$$\triangle_0 \cup \{\mathtt{EX(command_1)}\} \vdash_{PR} \triangle_1$$

$$\ldots$$

$$\triangle_i \cup \{\mathtt{O^i EX(command_{i+1})}\} \vdash_{PR} \triangle_{i+1}$$

$$\ldots$$

During this iterative process completely recognized plans can be deleted from $\triangle_i$ and are reported to the application system, and if no hypothesis can explain the observed actions, an adapted set $\triangle_0$ of generated possible hypotheses must be delivered by the generation component.

Assume that until now an action sequence $\Phi = \mathtt{EX(a_1)};\ldots;\mathtt{EX(a_{n-1})}$ was observed and that each of the plan hypotheses in $\triangle_{n-1} = \{\mathtt{P_1,P_2,\ldots,P_m}\}$ could explain those observations. Let $\mathtt{a_n}$ be the next observed action. Then the plan recognition process selects those hypotheses $\mathtt{P_i}$ that also account for the action sequence $\Phi' = \Phi;\mathtt{EX(a_n)}$. For each such $\mathtt{P_i}$ this means: (a) Φ' contains a parameter binding compatible to the one demanded in $\mathtt{P_i}$, (b) there is a suitable concrete domain command in Φ' wherever the plan hypothesis contains an abstract command, (c) for every nondeterministic choice in $\mathtt{P_i}$, Φ' contains exactly one of the alternative actions, and (d) Φ' induces a temporal structure compatible with the initial part of $\mathtt{P_i}$. For each plan hypothesis $\mathtt{P} \in \triangle_i$ the iterative step can be described as follows. Let $\mathtt{EX(command_i)}$ be the formula describing the last observed action. Then the plan recognizer tries to derive a new hypothesis $\mathtt{P^1}$ which will become a member of $\triangle_{i+1}$ through "$\mathtt{P} \wedge \mathtt{O^i EX(command_i)} \vdash_{PR} \mathtt{P^1}$ "where $\mathtt{P}$ and $\mathtt{P^1}$ are related in the following way. There is a way to split $\mathtt{P}$ into different command segments. $\mathtt{Init_P}$ is the initial segment and contains that part of the hypothesis that was already recognized. It exactly corresponds to the sequence of observed actions of former recognition steps. $\mathtt{Mid_P}$ describes just that part of $\mathtt{P}$ considered in the current recognition step. $\mathtt{Rest_P}$ is the part that will be considered in the next step if the current recognition step is successful. Thus we have $\mathtt{P} = \mathtt{Init_P;Mid_P;Rest_P}$ and $\mathtt{P^1} = \mathtt{Init_P;EX(command_i);Rest_{P1}}$ where $\mathtt{Rest_{P1}}$ results from $\mathtt{Rest_P}$ by substituting formal parameters bound in the last step. If $\mathtt{Rest_{P1}}$ becomes empty, the plan corresponding to this hypothesis was successfully recognized. In the next recognition cycle the first command of $\mathtt{Rest_{P1}}$ will become the new $\mathtt{Mid_{P1}}$ part.

The current recognition step is successful if we can show that $\mathtt{Mid_P}$ and $\mathtt{EX(command_i)}$ fulfill the requirements (a) - (d) listed above. This is the case if the recognizer deduces whether the following holds: $\not\vdash \mathtt{EX(command_i), Mid_P} \rightarrow$. That means that if the observed command and the $\mathtt{Mid_P}$ part are consistent the $\mathtt{Mid_P}$ part accounts for the command and the instatiated observation becomes part of the $\mathtt{Init}$ part of the new hypothesis.

References

[BBD+91] M Bauer, S. Biundo, D. Dengler, M. Hecking, J. Köhler, and G. Merziger. Integrated Plan Generation and Recognition - A Logic-Based Approach. Report No. RR-91-

26, German Research Center for Artificial Intelligence, Stuhlsatzenhausweg 3, W-6600 Saarbrücken 11, Germany, 1991

[BD92] S. Biundo and D. Dengler. The Logical Language for Planning (LLP). Research report, German Research Center for Artificial Intelligence Inc., 1992. forthcoming.

[Fin83] T. W. Finin. Providing help and advice in task oriented systems. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 176–178, 1983.

[Gen79] M. R. Genesereth. The role of plans in automated consultation. In *Proceedings of the 6th International Joint Conference on Artificial Intelligence*, pages 311–319, 1979.

[Hec87] M. Hecking. How to Use Plan Recognition to Improve the Abilities of the Intelligent Help System SINIX Consultant. In *Proceedings of the Second IFIP Conference on Human-Computer Interaction, held at the University of Stuttgart, Federal Republic of Germany, 1-4 September, 1987*, pages 657–662, 1987.

[HH87] M. Hecking and K. Harbusch. Plan Recognition through Attribute Grammars. Memo No. 17, Dept. of Computer Science, University of Saarbrücken, W.Germany, 1987.

[HKN+88] M. Hecking, C. Kemke, E. Nessen, D. Dengler, M. Gutmann, and G. Hector. The SINIX Consultant - A Progress Report. Memo No. 28, Dept. of Computer Science, University of Saarbrücken, W.Germany, 1988.

[Kau87] H. A. Kautz. A formal theory of plan recognition. Report No. TR 215, University of Rochester, Department of Computer Science, 5 1987.

[Let88] S. Letovsky. Plan analysis of programs. In *AAAI-88 Workshop on Plan Recognition*, St. Paul, Minnesota, 1988.

[Lit85] D. J. Litman. Plan recognition and discourse analysis: An integrated approach for understanding dialogues. Technical Report TR 170, Department of Computer Science, The University of Rochester, Rochester, NY, 1985.

[Moo88] R. J. Mooney. Explanation-based learning of plans for plan recognition. In *AAAI-88 Workshop on Plan Recognition*, St. Paul, Minnesota, 1988.

[NWW92] P. Norwig, W. Wahlster, and R. Wilensky. *Intelligent Help Systems for UNIX - Case Studies in Artificial Intelligence*. Springer, Heidelberg, 1992.

[Pol86] M Pollack. A model of plan inference that distinguishes between the beliefs of actors and observers. In *Proceedings of the ACL-86*, pages 207–214, 1986.

[SI81] C. Sidner and D. Israel. Recognizing intended meaning and speakers plans. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 293–298, 1981.

[SSG78] C. F. Schmidt, N. S. Sridharan, and J. L. Goodson. The plan recognition problem: An intersection of psychology and artificial intelligence. *Artificial Intelligence*, 11:45–83, 1978.

[Vil90] M. Vilain. Getting serious about parsing plans: a grammatical analysis of plan recognition. In *Proceedings of the 8th National Conference of the American Association on Artificial Intelligence, Boston, MA*, pages 190–197, 1990.

[Wob88] W Wobcke. A logical theory of plan recognition. In *AAAI-88 Workshop on Plan Recognition*, St. Paul, Minnesota, 1988.

On the Computational Complexity of Temporal Projection and Plan Validation[1]

Bernhard Nebel
WIP
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3
6600 Saarbrücken 11
Germany
e-mail: nebel@dfki.uni-sb.de

Christer Bäckström
Department of Computer and Information Science
Linköping University
S-581 83 Linköping
Sweden
e-mail: cba@ida.liu.se

Abstract

One kind of temporal reasoning is *temporal projection*—the computation of the consequences of a set of events. This problem is related to a number of other temporal reasoning tasks such as *story understanding*, *planning*, and *plan validation*. We show that one particular simple case of temporal projection on partially ordered events turns out to be harder than previously conjectured. However, given the restrictions of this problem, story understanding, planning, and plan validation appear to be easy. In fact, plan validation, one of the intended applications of temporal projection, is tractable for an even larger class of plans.

[1]A more complete description of this work is presented in [NB91,NB92]

1 Temporal Projection

The problem of *temporal projection* is to compute the consequences of a set of events. Dean and Boddy [DB88] analyze this problem for sets of *partially ordered events* assuming a propositional STRIPS-like [FN71] representation of events. The motivation behind this analysis is that the *validation of non-linear plans* and *story understanding* tasks seem to be based on such a form of temporal reasoning.
Dean and Boddy investigate the computational complexity of a number of restricted problems and conclude that even for severely restricted cases the problem is NP-hard, which motivate them to develop a tractable and sound but incomplete decision procedure for the temporal projection problem.
Among the restricted problems they analyze, there is one they conjecture to be solvable in polynomial time. This problem can be characterized as follows. Each STRIPS-rule has only one precondition, which also appears as the only element in the delete-list, and the add-list also contains only one element. Additionally, the set describing the state is also restricted to contain only one (positive propositional) element. Although these restrictions sound as if all temporal reasoning tasks must become completely trivial, it turns out that even in this case temporal projection is NP-hard, as is shown in [NB91].

2 The Relation between Plan Validation and Temporal Projection

The above result is somewhat surprising, because *planning*, for instance, seems to be easily solvable given the restriction of this temporal projection problem. Indeed, planning under this restriction is tractable [NB91,Byl91].
This observation casts some doubts on whether temporal projection is indeed the problem underlying plan validation and story understanding, as suggested by Dean and Boddy [DB88]. It seems natural to assume that the *validation of plans* is not harder than planning. Our NP-hardness result for the simple temporal projection problem seems to suggest the contrary, though.
The most problematical point in the definition of the temporal projection problem by Dean and Boddy seems to be that event sequences are permitted to contain events that do not affect the world because their preconditions are not satisfied. In a planning context, however, we would consider such sequences that contain unexecutable actions as illegal.
If we define the plan validation problem in a way such that all possible event sequences have to contain only events that affect the world, plan validation is tractable for the class of plans containing only unconditional events, a point already suggested by Chapman [Cha87].
In fact, deciding a conjunction of temporal projection problems that is equivalent to the plan validation problem appears to be easier than deciding each conjunct in isolation. The main reason for this fact is that for plan validation purposes we can stop to test as soon as if we find an illegal sequence with the result that the plan is invalid. Computing the temporal projection of a set of events, however

using the definition of Dean and Boddy – requires that we also consider illegal event sequences.
Summarizing, the problem decomposition by Dean and Boddy, namely, to decompose the plan validation problem into a number of temporal projection problems, seems to be conceptually right. From a computational point of view, however, this decomposition does not make sense.
Additionally, it turns out that the tractable and sound decision procedure for temporal projection fails on plans that could be validated in (low-order) polynomial time.

3 Conclusion

Although many forms of temporal reasoning seem to require substantial computational resources, i.e., temporal reasoning problems are generally intractable, many forms of this kind of reasoning are nevertheless easier. The analysis of temporal projection by Dean and Boddy [DB88] suggests that this task belongs to the class of reasoning tasks that are difficult. However, as we have shown, the form of temporal projection as defined by Dean and Boddy does not arise in the applications they envisioned. To the contrary, plan validation is tractable for a large class of non-linear plans, namely, all unconditional plans.

References

[Byl91] Tom Bylander. Complexity results for planning. In John Mylopoulos and Ray Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 274–279, Sydney, Australia, August 1991. Morgan Kaufmann.

[Cha87] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377, July 1987.

[DB88] Thomas L. Dean and Mark Boddy. Reasoning about partially ordered events. *Artificial Intelligence*, 36(3):375–400, October 1988.

[FN71] Richard E. Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving as problem solving. *Artificial Intelligence*, 2:198–208, 1971.

[NB91] Bernhard Nebel and Christer Bäckström. On the computational complexity of temporal projection and some related problems. DFKI Report RR-91-34, German Research Center for Artificial Intelligence (DFKI), Saarbrücken, 1991. Also published as Research Report LiTH-IDA-R-91-34, Department of Computer and Information Science, Linköping University, Linköping, Sweden.

[NB92] Bernhard Nebel and Christer Bäckström. On the computational complexity of temporal projection and plan validation. In *Proceedings of the 10th National Conference of the American Association for Artificial Intelligence*, San Jose, CA, July 1992. MIT Press. To appear.

Learning Plan Abstractions: Formal Model and Method

Ralph Bergmann Franz Schmalhofer
KIWI
German Research Center for Artificial Intelligence (DFKI)
Postfach 2080
6750 Kaiserslautern
Germany
e-mail: bergmann@dfki.uni-kl.de schmalho@dfki.uni-kl.de

Abstract

Plan abstraction is most important for plan recognition tasks as well as for the acquisition of planning knowledge for hierarchical planning. Knowledge intensive machine learning procedures such as explanation-based learning have shown to be capable of producing generalizations of plans, but no approach has been proposed for constructing plan abstractions. In this paper, generalization is distinguished from abstraction. A general formal model of plan abstraction is proposed, in which the problem of plan abstraction is decomposed into finding a state abstraction mapping and a sequence abstraction mapping. A generic abstraction theory is applied for generating deductively justified plan abstractions. Lastly, a five-phase procedure is outlined which supports the automatic construction of an abstract plan from a concrete plan. This procedure demands, that a concrete and an abstract planning space is defined as a STRIPS- system. Additionally, a generic abstraction theory is required which defines the sentences of the abstract world in terms of the concrete world.

1 Introduction

Plan recognition may be seen as analyzing a sequence of operations or programming statements of a computer program in order to apply a user's application- oriented terminology for constructing a corresponding logically sound abstract plan. This abstract plan is much more closely related to the user's thinking than the operations or programming statements of the concrete plan or program. For example, recognition of an assembler program involves analyzing a sequence of machine-level instructions to identify the use of higher programming concepts such as data representations or abstract data manipulating operations. Abstract plans are very important for reducing the complexity of planning tasks. Friedland and Iwasaki [FI85] have proposed skeletal plans which they defined as a sequence of abstract operations which can be refined to a solution of a concrete problem. Korf [Kor88] has recently shown that with an appropriately defined hierarchy of plan abstractions, planning problems can be reduced in their complexity from an exponential search to a linear search problem under certain conditions. In this paper we propose a theoretical framework for describing plan abstraction in general. A knowledge intensive learning method which supports the automatic construction of abstract plans from concrete plans is finally sketched.

2 The Problem of Learning Plan Abstractions

Michalski and Kodratoff [MK90] have recently pointed out, that abstraction must be distinguished from generalization. While generalization transforms a description along a set superset dimension, abstraction transforms a description along a level of detail dimension which usually involves a change in the representation space. A plan or program consists of a description which specifies some operations to be executed in a specific order. Each operation changes the state of the system in some way. Therefore plan abstraction has two independent dimensions: The first dimension describes a change in the level of detail for the representation of single states and the second dimension requires a change in the level of detail by reducing the number of states contained in a plan. As a consequence plan abstraction must perform a change of the representation of the state descriptions and a change of the operations which describe the state transitions. Knowledge-intensive learning mechanisms such as explanation- based learning (EBL) can construct plan generalizations [FHN72, MCK$^+$89,Ber92b], but have not yet been shown to be capable of forming truly abstract plans. Since EBL procedures form operational specializations of a domain theory with respect to a given example, the operational specializations are at an intermediate level of generality. They are more general than the example but cannot be more general than the sentences of the underlying domain theory. In order to construct an abstract plan, the underlying domain theory must be enhanced with knowledge about abstract states and abstract operations.

3 A Formal Model of Plan Abstraction

Since the goal of abstraction is to transform a plan according to a level of detail dimension by performing a change in representation, two planning spaces, a concrete and an abstract space, must be defined. We assume, that each of these planning spaces or worlds is represented as a STRIPS-system $W = (R, T, Op)$, where R is a set of essential sentences [Lif87] which describe the dynamic aspects of a state of the world. T is a static theory which allows to deduce additional properties of a state in the world and Op is a set of operators described by a precondition list, an add list and a delete list. Let $W_c = (R_c, T_c, Op_c)$ be the STRIPS-system which describes the concrete world and $W_a = (R_a, T_a, Op_a)$ be the STRIPS-system which describes the abstract world. Note, that a totally different terminology can be employed to represent the concrete and the abstract world. The problem of plan abstraction can now be described as transforming a plan p_c from the concrete world into a plan p_a in the abstract world, with several conditions being satisfied. This transformation can formally be decomposed into two mappings: a *state abstraction mapping* a, and a *sequence abstraction mapping* b as follows:

Definition 1: A *state abstraction mapping* a: $S_c \rightarrow S_a$ is a mapping from S_c, the set of all states in the concrete world, to S_a, the set of all states in the abstract world, that satisfies the following conditions:
a) If $s_c \cup T_c$ is consistent then $a(s_c) \cup T_a$ is consistent.
b) If $s_c \cup s_{c'} \cup T_c$ is consistent then $a(s_c \cup s_{c'}) \supseteq a(s_c) \cup a(s_{c'})$.

Definition 2: A *sequence abstraction mapping* b: $N \rightarrow N$ relates an abstract state sequence $(sa_0, ..., sa_n)$ to a concrete state sequence $(sc_0, ..., sc_m)$ by mapping the indices i of the abstract states sa_i onto the indices j of the concrete states sc_j, such that $b(0) = 0$, $b(n) = m$ and $b(u) < b(v)$ if $u < v$.

Using these two mappings, plan abstraction can be defined as follows:
Definition 3: A plan p_a is an *abstraction* of a plan p_c if there exists a state abstraction mapping $a : S_c \rightarrow S_a$ and a sequence abstraction mapping $b : N \rightarrow .N$, such that: If p_c and an initial state sc_0 induce the state sequence $(sc_0, ..., sc_m)$ and $sa_0 = a(sc_0)$ and $(sa_0, ..., sa_n)$ is the state sequence which is induced by sa_0 and the abstract plan p_a, then $a(sc_{b(i)}) = sa_i$ holds for all $i \in N1, ..., n$.

This definition of plan abstraction requires that for each state which results from the abstract plan, a corresponding state in the concrete plan exists. The sequence abstraction mapping defines which concrete state corresponds to each abstract state. Note that only some of the concrete states have a corresponding abstract state. Concrete states which are not abstracted by the sequence abstraction mapping describe a kind of detail which is eliminated by the plan abstraction. While the sequence abstraction mapping reduces the level of detail of a plan by reducing the number of states contained in a plan, the state abstraction mapping changes the level of detail by changing the representation for each state from concrete to abstract. A good (i.e. useful) state abstraction mapping collects all those abstract sentences which support the main subgoals of the plan that has to be abstracted. In order to restrict the number of possible state abstraction mappings in this manner, a generic abstraction theory T_g [GRS91] can be applied. Such a theory consists of a set of axioms which describe the essential sentences r_a of the abstract world in terms of sentences of the concrete world. For a deductively justified state abstraction mapping a we require, that: If $r_a \in Na(s_c)$ then $s_c \cup T_c \cup T_g \vdash r_a$.

4 A Method for Constructing Plan Abstractions

The task of automatically constructing deductively justified abstractions by using know-ledge- intensive machine learning methods based on cognitively adequate generic abstraction theories can be achieved by the following five-phase procedure: In the first phase, the execution of the concrete plan p_c is simulated and the sequence of the induced states in the concrete planning space is computed. In the second phase, for each of these states, an abstract description is derived by trying to proof the essential sentences r_a of the abstract planning space through $s_c \cup HT_c \cup T_g \vdash r_a$. Here the generic abstraction theory is employed to construct state abstractions which are composed of justified sentences. In third phase, for each pair of abstract planning states (sa_u, sa_v), it is checked, if there exists an abstract operation from Op_a which is applicable in sa_u and which transforms sa_u into sa_v. A directed graph, where abstract states are represented as nodes and abstract operations as arcs, is finally construced in this phase. In phase four, a complete and consistent path from the initial abstract state to the final abstract state is searched. This path describes an abstract plan and defines a state abstraction mapping and a sequence abstraction mapping. In the fifth phase, explanation- based generalization is applied to generalize the sequence of abstract operations into an abstract plan with corresponding application conditions [Ber92b,Ber92a].

References

[Ber92a] R. Bergmann. Explanation-based learning for the automated reuse of programs. In *Proceedings of the IEEE-Conference on Computer Systems and Software Engineering, COMPEURO92.*, 1992.

[Ber92b] R. Bergmann. Knowledge acquisition by generating skeletal plans. In F. Schmalhofer, G. Strube, and Th. Wetter, editors, *Contemporary Knowledge Engineering and Cognition*, Heidelberg, 1992. Springer. (in press).

[FHN72] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[FI85] P. E. Friedland and Y. Iwasaki. The concept and implementation of skeletal plans. *Journal of Automated Reasoning*, pages 161–208, 1985.

[GRS91] A. Giordana, D. Roverso, and L. Saitta. Abstracting background knowledge for concept learning. In Y. Kodratoff, editor, *Lecture Notes in Artificial Intelligence: Machine Learning-EWSL-91*, volume 482, pages 1–13, Berlin, 1991. Springer.

[Kor88] R.E. Korf. Optimal path-finding algorithms. In L. Kanal and V. Kumar, editors, *Search in Artificial Intelligence*, pages 223–267. Springer, New York, 1988.

[Lif87] V. Lifschitz. On the semantics of strips. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, Timberline, Oregon, 1987.

[MCK+89] S. Minton, J. G. Carbonell, C.A. Knoblock, D. R. Kuokka, O. Etzioni, and Y. Gil. Explanation-based learning:a problem solving perspective. *Artificial Intelligence*, 40:63–118, 1989.

[MK90] R. S. Michalski and Y. Kodratoff. Research in machine learning: Recent progress, classification of methods, and future directions. In Y. Kodratoff and R. S. Michalski, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 3, chapter 1, pages 3–30. Morgan Kaufmann, San Mateo, CA, 1990.

The Design of Illustrated Documents as a Planning Problem

Elisabeth André Thomas Rist
WIP
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3
6600 Saarbrücken 11
Germany
e-mail: andre@dfki.uni-sb.de rist@dfki.uni-sb.de

Abstract

Not only the generation of text, but also the generation of multimodal documents can be considered as a sequence of communicative acts which aim to achieve certain goals. For the realization of a system able to automatically generate illustrated documents, a plan-based approach seems adequate. To represent knowledge about how to present information, we have designed presentation strategies which relate to both text and picture production. These strategies are considered as operators of a planning system. However, a conventional hierarchical planner for determining the contents and the rhetorical structure of a document has proven inappropriate to handle the various dependencies between content determination, mode selection and content realization. To overcome these problems, a new planning scheme has been developed that supports data transfer between the content planner and the mode-specific generation components and allows for revising an initial document structure.

1 Introduction

A basic assumption behind the design of the multimodal presentation system WIP (cf. [Wahlster et al. 91]) is that not only the generation of text and dialog contributions, but also the design of graphics and multimodal presentations are planning tasks (cf. [André and Rist 90]). A conventional hierarchical planner has, however, proven inappropriate to handle the various dependencies between content determination, mode selection and content realization. In the following, we will briefly sketch an plan-based approach that integrates content and mode selection and allows for interaction with mode-specific generators. A more detailed description may be found in [André and Rist 92].

2 The Basic Planning Scheme

To represent knowledge about how to present information, we have designed presentation strategies which relate to both text and picture production. These strategies are considered as operators of a planning system (cf. [André and Rist 90]). The basic idea behind the planning process is as follows: Given a presentation goal, try to find strategies whose effect matches the presentation goal and check for which variable bindings their applicability conditions hold. Then select a strategy, instantiate it and post the main and subsidiary acts as new subgoals or - in the case of elementary acts such as 'Depict' or 'Assert' - forward them to the mode-specific generators.
To ensure that document fragments in multiple modalities are smoothly tailored to each other in the document to be generated, one also has to consider various dependencies between content determination, mode selection and content realization. As a consequence, the process sketched above appears to be much more complicated with respect to flow of control and data between the presentation planner and the generators.

3 Interleaving Presentation Planning, Text and Graphics Generation

Previous work on natural language generation has shown that content selection and content realization should not be treated independently of each other. A strictly sequential model in which data only flow from the "what to present" to the "how to present" part has proven inappropriate because the components responsible for selecting the contents would have to anticipate all decisions of the realization components. This problem is compounded if, as in our case, content realization is done by separate components (currently a text and a graphics generator) of which the content planner has only limited knowledge.
It seems even inappropriate to sequentialize content planning and mode selection although mode selection is only a very rough decision about content realization.

Selecting a mode of presentation depends to a large extent on the information to be communicated. On the other hand, content planning is strongly influenced by previously selected mode combinations. E.g., to graphically refer to a physical object, we need visual information that may be irrelevant to textual references.
A better solution is to interleave content planning, mode selection and content realization. In the WIP system, we interleave content and mode selection using a uniform planning mechanism. This has become possible since the presentation strategies and metarules accessed by the planner contain not only knowledge about what to present, but also knowledge about adequate mode combinations. In contrast to this, presentation planning and content realization are performed by separate components that access disparate knowledge sources. This modularization enables parallel processing, but makes interaction between the single components necessary. As soon as the planner has decided which generator should encode a certain piece of information, this piece should be passed on to the respective generator. Conversely, the planning component should immediately incorporate the results of the generators. Therefore, the processing of all components has to be 'interrupted' at certain decision points to allow other components to react.

4 Propagating Data During Presentation Planning

Since every component has only limited knowledge of other components, data have to be passed from one component to the other. E.g., if a generator finds a better solution or is not able to satisfy a task, it has to inform the planner, which has to modify its initial plan. To ensure the consistency of the document, all changes have to be propagated to other branches of the plan structure.
Information flow is not only necessary between the content planner and the generators, data also have to be propagated from one generator to the other. Suppose the text generator has generated a referring expression for an object shown in a picture. If the picture is changed due to graphical constraints, it might happen that the referring expression no longer fits. Thus, the planner will have to create a new object description and pass this description on to the text generator, which will have to replace the initial referring expression by a new one.
Furthermore, the need for propagating data during presentation planning arises when dealing with dependencies between presentation strategies.

5 The Architecture of the Presentation Planner

The considerations above led to an architecture for the presentation planner as shown in Fig. 1. The basic planning module selects operators that match the presentation goal and expands the nodes to generate a refinement-style plan in the form of a DAG. The plan evaluation/revision module is responsible for evaluating and revising plans.To allow for alternating revision and expansion processes, WIP's presentation planner is controlled by a plan monitor that determines the next action and the next nodes to be expanded. All components of the presentation planner have read/write access to the document plan. The leaves of the document plan are connected to entries in the task queues of the mode-specific generators. Thus, a two-way exchange of information between the two generators is possible via the document plan.

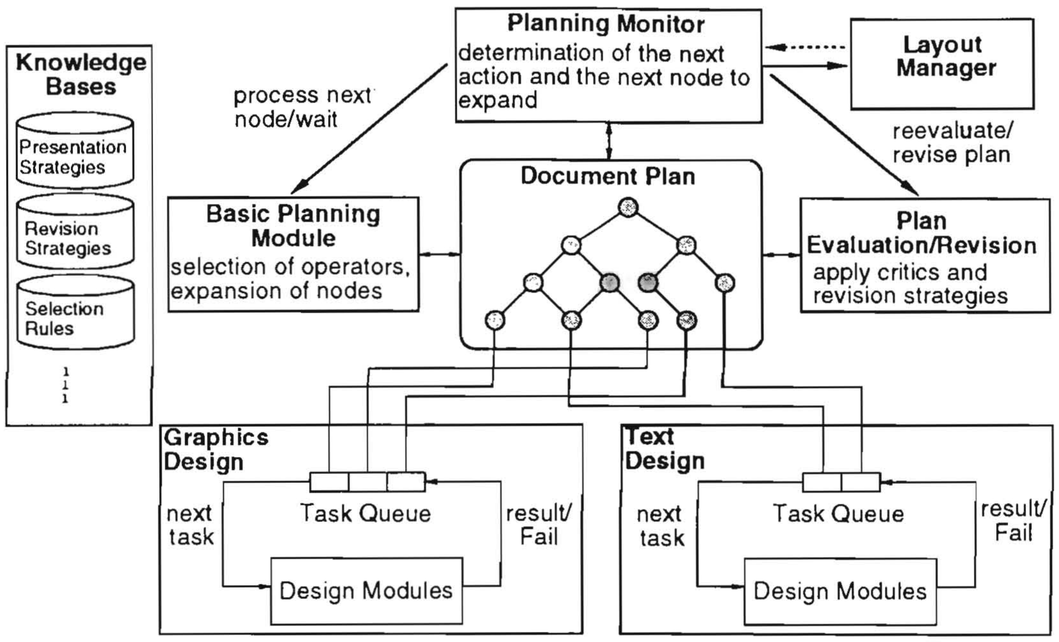

Figure 1: The Architecture of WIP's Presentation Planner

6 Summary

We have argued that not only the generation of text, but also the synthesis of multimodal documents can be considered as a communicative act which aims to achieve certain goals. For the realization of a system able to automatically generate illustrated documents, we have proposed a plan-based approach that supports data transfer between the content planner and the mode-specific generators and allows for global plan evaluation after each plan step. The modularization of presentation planning and mode-specific generation has led to the problem that the results provided by the generators may deviate from the initial presentation plan. Since such deviations have to be reflected in the presentation plan, the planning scheme has also to comprise restructuring methods.

References

[André and Rist 90] André, E. and Rist, T. 1990. Synthesizing Illustrated Documents: A Plan-Based Approach. In *Proceedings of InfoJapan '90* Vol. 2 (163–170).

*[André and Rist 92] André, E. and Rist, T. 1992. The Design of Illustrated Doc*uments as a *Planning* Task. German Research Center for Artificial Intelligence, DFKI Research Report.

[Wahlster et al. 91] Wahlster, W., André, E., Bandyopadhyay, S., Graf, W., Rist, T. 1991. WIP: The Coordinated Generation of Multimodal Presentations from a Common Representation. DFKI (University of Saarbrücken) Technical Report.

Deductive Planning in a Command Language Environment

Susanne Biundo   Dietmar Dengler
PHI
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3
6600 Saarbrücken 11
Germany
e-mail: biundo@dfki.uni-sb.de   dengler@dfki.uni-sb.de

Abstract

A new deductive approach to planning is presented. It has been developed to solve planning tasks in command language environments and mainly relies on programming logics.
We introduce a logical framework defining the interval logic LLP and a sequent calculus. The deductive planning system based on this logic provides several strategies for deriving sequential as well as conditional and *while*-plans. The system is currently being implemented using concepts from tactical theorem proving.

1 Introduction

The deductive approach to planning we introduce aims at solving planning tasks in an intelligent help system's context.
Intelligent help systems support users of complex application systems by the achievements of qualified experts. One aspect of this support is devoted to the generation of executable plans which, as a means of active help, are presented to the user who may follow them in order to reach his current goals. Another aspect concerns the synthesis of *abstract* plans. It plays a central role within the PHI system (cf. [BBD$^+$91]). PHI realizes the integration of logic-based plan recognition and plan generation components where plan recognition is done on the basis of abstract plans. The abstraction involves *object variables*, *abstract actions*, and *control structures* as well as *temporal abstraction*. The latter comprises to formulate plan specifications including propositions, like, for instance : "action a is carried out *at some time* during the execution of plan p".
The planning domain is the command language environment of an application system. Consequently, the basic actions correspond to the elementary statements of this command language. Together with the control structures provided for plans this strongly suggests to view *plans as programs*.

To realize deductive planning in this context we have developed the interval-based temporal logic LLP (Logical Language for Planning) that combines features of a *programming logic* [Krö87] with those of *interval logics* (cf., e.g., [RP86]). A sequent calculus for LLP provides the deductive basis for plan generation. Plans are generated by proofs of so-called plan specification theorems, which are special type LLP formulas. They represent *partial* or *total* correctness assertions or express *liveness properties* of plans, respectively.
Axiomatizing the planning domain using LLP provides a *representational* "solution" of the frame problem: Basic actions are represented like assignment statements in programming logics. As a consequence, only one axiom schema is needed for each action to describe its effects as well as the frame properties that are not affected by the action.

A classical application domain for intelligent help systems are operating systems. Hence, we have chosen the example domain for our system to be a manageably sized subset of the operating system UNIX, namely its *mail system*, where commands like *type*, *delete*, or *save* manipulate objects, like *messages* or *mailboxes*.

2 Deductive Planning Using LLP

The Logical Framework

LLP relies on a many-sorted first-order language and, besides the normal logical variables, provides a set of so-called *local variables* for each sort. The local variables are borrowed from programming logics where they correspond to program variables *whose values can change from one state to another. We use local variables in the* same way and describe the effects of basic actions by a change of values of certain local variables.

The modal operators provided by LLP are $\bigcirc$ (next), $\Diamond$ (sometimes), $\Box$ (always), and a sequential composition of formulas by the two-place modal operator ; (chop). Besides these operators, like in programming logics, also *assignments* and *control structures* are available. The conditional $\underline{if}\ \epsilon\ \underline{then}\ \alpha\ \underline{else}\ \beta$ for example, stands for the formula
$[\epsilon \rightarrow \alpha] \wedge [\neg\epsilon \rightarrow \beta]$. The *while*-operator is defined by the following axiom:
$\underline{while}\ \epsilon\ \underline{do}\ \alpha\ \underline{od}\ ; \beta \leftrightarrow [\underline{if}\ \epsilon\ \underline{then}\ [\alpha\ ; \underline{while}\ \epsilon\ \underline{do}\ \alpha\ \underline{od}\ ; \beta]\ \underline{else}\ \beta]$.
Basic actions are represented by atomic formulas using the predicate EX ("execute"). $EX(type(1, mbox))$, for example, represents the basic action of reading the first message in a mailbox $mbox$.
Certain formulas of our temporal logic are viewed as *plans*. Those *plan formulas* are

- all formulas $EX(c)$, where c is a term of type *command*,
- assignments of form $a := t$, where a is a *local variable* and t is a term,
- all formulas $\alpha; \beta$ where α and β are plan formulas,
- all formulas $\underline{if}\ \epsilon\ \underline{then}\ \alpha\ \underline{else}\ \beta$, where α and β are plan formulas and ϵ is a formula not containing any temporal operator or basic plan formula,
- all formulas $\underline{while}\ \epsilon\ \underline{do}\ \alpha\ \underline{od}\ ; \beta$, where α and β are plan formulas and ϵ is a formula not containing any temporal operator or basic plan formula.

Syntax, semantics, and a sequent calculus for LLP are given in [BD92].

Representing the Planning Domain

The application domain we choose for our examples is a mail system where the objects are "mailboxes" and "messages"; a mailbox is viewed as a list of one or more messages. During the activation of the mail system different aspects of messages can be changed by the commands the user executes: so, every command causes a state transition changing the content of the current mailbox. We deal with this behaviour by using local variables to represent objects of type *mailbox* or *message*, respectively.

The axioms describing mail commands are given like axioms for assignment statements in programming logics. As an example, the "type" command for reading a message is axiomatized by the following scheme:

$$\begin{array}{ll}\forall i: & integer \\ & [[\neg flag(i, Current\_mbox) \equiv \text{"d"} \wedge \\ & \mathsf{P}\ {}^{flag(i, Current\_mbox)}_{\text{"r"}}\ {}^{Current}_{Current+1} \wedge EX(type(i, Current\_mbox))] \rightarrow \bigcirc\mathsf{P}]\end{array}$$

The symbol P is a *metavariable* for formulas; the substitution instructions correspond to the *effect* of the "type" command: "type" does nothing else than changing the $flag$ of the i-th message in $Current\_mbox$ to "r" and increases the $Current$-counter by 1.
During the deductive plan generation process appropriate instances of these axiom schemata are used. The instances can either be frame axioms or axioms describing effects of the according actions.

Deductive Planning

The planning process starts from a plan specification formula. Specifications are formulas containing metavariables for plans. Deriving a plan from such a specification is done by constructing a sequence proof that provides appropriate instantiations for these variables. That means, based on the specification we develop a proof tree applying several sequence rules in turn until all leaves of the tree are closed.

The instantiations to be made for the plan meta variable are restricted to plan formulas. If we starting from the specification formula end up with a proof tree where no metavariables are left, the instantiation generated for the plan variable represents an *executable* plan that satisfies the given specification.

We distinguish between different types of plan specifications. Among them we have assertions about *intermediate states* (also called *liveness properties* [Krö87]). They read

$$\mathsf{Plan} \rightarrow [\phi_i \rightarrow \Diamond\phi_g]$$

stating that ϕ_g holds some time during the execution of Plan.

Suppose, our current plan specification is "Read any message of the mailbox C_mb and delete it". The input for the plan generation process is then a formula of the form:

$$\mathsf{Plan} \rightarrow [flag(x, C\_mb) \not\equiv \text{"d"} \rightarrow \Diamond[flag(x, C\_mb) \equiv \text{"r"} \wedge \Diamond flag(x, C\_mb) \equiv \text{"d"}]]$$

The symbol Plan is a metavariable for a plan formula and has to be appropriately instantiated during the proof.

A plan is derived from this specification by applying several sequence rules according to a strategy developed for the synthesis of sequential plans out of liveness properties statements. It includes an automatic classification of formulas resulting from rule applications: Some of them are subgoals leading to a proper instantiation of a plan metavariable, while others represent so-called *plan assertions*. They describe certain properties of the specified plan that cannot be proved until the plan synthesis has been completed.

The plan we obtain from the above specification reads:

$$EX(type(x, C\_mb)); EX(delete(x, C\_mb))$$

stating that the message has to be read and subsequently has to be deleted.

3 Conclusion

The logical framework we have introduced forms the basis for deductive planning in a command language environment. Plans are generated by sequence proofs of specification formulas. Specification formulas contain metavariables for formulas. During the proof these metavariables have to be replaced by plan formulas. If the proof succeeds these finally constitute the specified plan. Besides sequential ones plans can be generated that contain control structures like *if then else* and *while*. Proofs of specification formulas are carried out in a goal directed way using several derived rules and appropriate proof strategies.

References

[BBD+91] M. Bauer, S. Biundo, D. Dengler, M. Hecking, J. Koehler, and G. Merziger. Integrated Plan Generation and Recognition: A Logic-Based Approach. In *Proceedings of the 4. Internationaler GI-Kongress Wissensbasierte Systeme, München*, pages 266–277. Springer IFB 291, 1991.

[BD92] S. Biundo and D. Dengler. An Interval-Based Temporal Logic for Planning. Research report, German Research Center for Artificial Intelligence, Saarbücken, 1992.

[Krö87] F. Kröger. *Temporal Logic of Programs.* Springer, Heidelberg, 1987.

[RP86] R. Rosner and A. Pnueli. A Choppy Logic. In *Symposium on Logic in Computer Science*, Cambridge, Massachusetts, 1986.

Planning as Transformation of Declarative Representations in COLAB

Knut Hinkelmann
ARC-TEC
German Research Center for Artificial Intelligence (DFKI)
Postfach 2080
6750 Kaiserslautern
Germany
e-mail: hinkelmann@dfki.uni-kl.de

Abstract

Given the CAD-like geometry of a rotational-symmetric workpiece abstract NC macros for rough-turning the workpiece on a CNC lathe machine are generated. A declarative representation of a production plan is derived from a declarative representation of the start and goal states by step-wise abstraction, association and refinement operations. All intermediate representations are declarative, too. The approach is implemented in COLAB, a hybrid-knowledge compilation laboratory which integrates the power of forward and backward reasoning (incl. functional programming), constraint propagation, and taxonomic classification. The declarative subformalisms of COLAB collaborate in a prototypical synergetic manner to perform the central subtasks of CAD-to-NC transformations.

1 Introduction

To reach a goal or solve a problem a rational or even intelligent agent will not simply try arbitrary actions but will first think of the problem. But to think about a problem, an internal representation is needed of the initial state from which to start and of the goal that should be reached. The result of this planning process is a set of actions, which must be performed to reach the goal state from the inital state. This set of actions itself has an internal structure for its own. For instance, the order in which the actions are performed may be significant. This means that the set of actions of a plan is at least partially ordered. Also the plan may contain loops and conditions, e.g. if there are incomplete informations.
The paradigm of planning as transformation of declarative representations is exemplified in the μCAD2NC system [BHH+91] for production planning of rotation-symmetric products on lathe-turning machines. A declarative representation of a production plan is derived from a declarative representation of the start and goal states by step-wise rewriting and association operations. All intermediate representations are declarative, too. The goal state is described by a drawing in a CAD system. The initial state is not explicitly represented but is implicitly considered in the planning process as a cylinder whose length and radius corresponds to the length and maximum radius of the workpiece. The resulting plan contains as its main part an NC program with instructions for chucking, tool change, and cutting the contour of the lathe work.

2 Planning in COLAB

The presented approach is implemented in the COLAB knowledge representation system. The COLAB system has been designed as a COmpilaton LABoratory aiming at a synergetic COlLABoration of different knowledge representation formalisms. Its architecture corresponds to terminological systems like KL-ONE separating taxonomic and affirmative (often called assertional) knowledge. The COLAB system is comprised of subsystems dealing with different kinds of knowledge. A hybrid knowledge base can contain items from *all* subsystems. Tags indicate the type of a knowledge item and determine how it has to be processed. Dynamic cooperation of the subsystems is organized through access primitives providing an interface to the respective reasoning services. For a more detailed description of COLAB and its subsystems see [BHHM91].

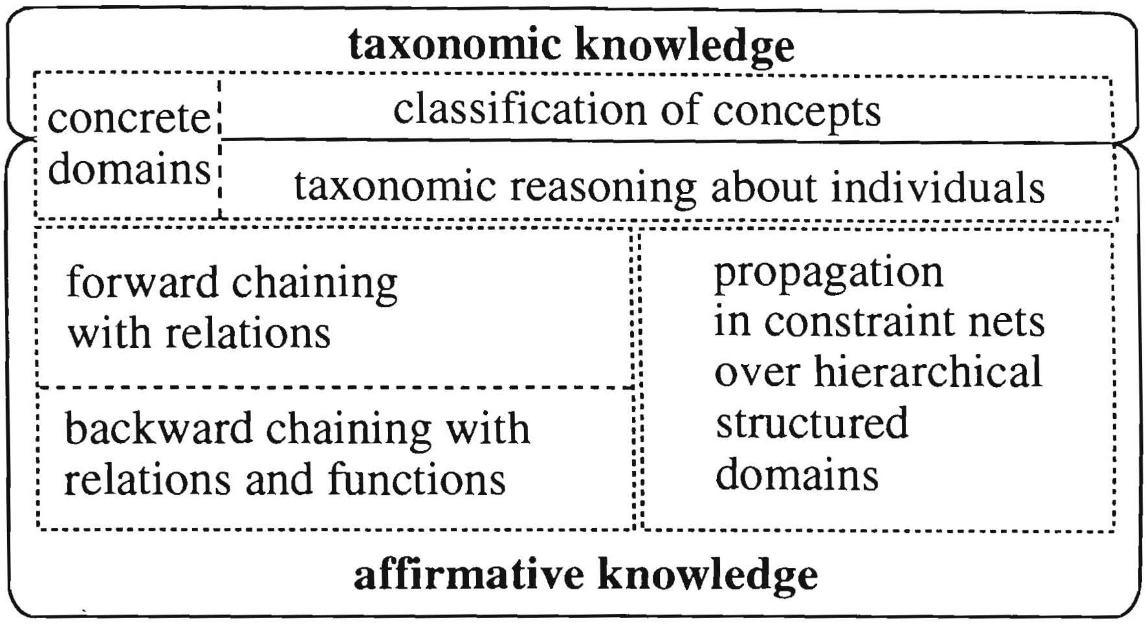
taxonomic knowledge
concrete domains
classification of concepts
taxonomic reasoning about individuals
forward chaining with relations
backward chaining with relations and functions
propagation in constraint nets over hierarchical structured domains
affirmative knowledge

Figure 2: The COLAB Representation Architecture

The starting point of our planning approach is a very 'elementary' description of a workpiece according to the boundary representation of a product model. Original CAD data can be transformed into this representation in a simple preprocessing step. In the example presented here we will concentrate on a geometrical description of a workpiece's surfaces and topological neighborhood relations omitting technological information for simplicity. The following sample knowledge items represent part of a workpiece:

```
...
(attrterm (ring rng42 (tup '(center1 110)
                           '(center2 110)
                           '(radius1 30)
                           '(radius2 20))))
(attrterm (cylinder cyl43 (tup '(center1 110)
                               '(center2 120)
                               '(radius1 20)
                               '(radius2 20))))
(attrterm (ring rng44 (tup '(center1 120)
                           '(center2 120)
                           '(radius1 20)
                           '(radius2 25))))
...
(fact (neighbor rng42 cyl43))
(fact (neighbor cyl43 rng44))
...
```

The first three attribute terms (indicated by the tag `attrterm`) describe the geometry of three surfaces. The `tup` constructor is comparable to the list constructor and is used here to group the attributes. The attributes correspond to co-ordinates and radii of the surface boundaries. The `neighbor` facts determine the topological relations of the surfaces.

Production planning with these input data would be very complex. Instead there exists a library of skeletal plans. Each of these skeletal plans is accessed with a more or less abstract description of a characteristic part of a workpiece, which is called a workpiece feature. Thus, the representation of an abstract NC program is derived from the representation of the goal state by first generating an abstract feature description of the workpiece, associating features to skeletal plans and then refining these skeletal plans to an NC program.
In the first abstraction phase a feature description is aggregated from the elementary workpiece data by a cooperation of the taxonomic and rule-based reasoning components of COLAB. Feature abstraction starts bottom-up with a collection of attribute terms by asserting features into the fact base. The rules are very general mentioning in the ideal case only the most general features (in the subsumption hierarchy) ranging over the corresponding number of surfaces. As soon as a new feature instance or information about an already existing instance is asserted, its most special concept association is computed using the realization service. This information gain can trigger rules to derive further features. If, for instance, two surfaces are aggreagted to a biconic, the realization service may find that it is actually a left shoulder. Then this new fact may trigger any rule with premises referring to left shoulder, shoulder, biconic, or any more general feature.
The result of this first abstraction phase is a number of facts describing all the features occuring in the workpiece. Since the resulting production plan will be represented by one single term, the first classification phase is completed by converting the aggregated features into a nested term structure, which is called classified workpiece. The resulting representation is less redundant and more analogical, because the structure of the term reflects the topological ordering of the features:

```
(cwp (tup 40) ...
     (nft (grv (flk (tup (rng 110 25 20)))
               (grd (tup (rng 110 20 20) (rng 120 20 20)))
               (flk (tup (rng 120 20 25))) )
          (tup (nft (lsh (flk (tup (rng 110 30 20)))
                         (grd (tup (rng 110 20 20))) )
                    (rsh (flk (tup (rng 120 20 25)))
                         (grd (tup (rng 110 20 20))) ) ) ... )))))
```

When the features are collected into one term, for each feature various skeletal plans are retrieved by COLAB's relational-functional component. A skeletal plan contains the tools, the direction into which the cut is directed (left, right, vertical) and the co-ordinates of the feature's contour. In particular, depending on the material, the angles of the surfaces, etc. only a small subset of available skeletal plans is really suitable. Therefore, the constraint system is called to reject skeletal plans with unfitting tools. The remaining skeletal plans are composed to an SAC plan, i.e. a (higher-order skeletal) plan consisting of a composition of sequential, alternative, and commutative subplans. In a last refinement step an abstract NC program is generated by qualitative simulation of the SAC plan in the relational-functional subsystem: In this phase the order of commutative subplans is fixed and the best of the alternatives is selected. The main optimization criteria are minimal number of cuts and minimal number of tool changes. The final ANC program consists of a sequence of macro calls performing the roughing part of contour cutting. Modern NC programming system can easily expand these kind of macros into real NC program code.

```
(tup ... (roughing (tool dnmm-71 tmaxp-pdl93)
                   left
                   (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25)))
         (roughing (tool dnmm-71 tmaxp-pdr93)
                   right
                   (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) ... )
```

3 Summary

Starting with a declarative representation of a workpiece an NC program is generated by successive abstraction, association and refinement operations. In the first abstraction phase a classified workpiece is derived by cooperation of the taxonomic and the forward reasoning subsystem of COLAB. In a second phase skeletal plans associated with the features of the classified workpiece are retrieved by the relational-functional component. Thereby the constraint system rejects unsuitable plans before they are composed to an SAC plan. The final abstract NC program is generated by qualitative simulation of the SAC plan, selecting best alternatives and fixing the order of operations.

References

[BHH+91] Harold Boley, Philipp Hanschke, Martin Harm, Knut Hinkelmann, Thomas Labisch, Manfred Meyer, Jörg Müller, Thomas Oltzen, Michael Sintek, Werner Stein, and Frank Steinle. μCAD2NC: A declarative lathe-workplanning model transforming CAD-like geometries into abstract NC programs. Technical Report Document D-91-15, DFKI GmbH, November 1991.

[BHHM91] H. Boley, P. Hanschke, K. Hinkelmann, and M. Meyer. COLAB: A Hybrid Knowledge Compilation Laboratory. submitted for publication, December 1991.

A Deductive Approach to Plan Modification

Jana Koehler
PHI
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3
6600 Saarbrücken 11
Germany
e-mail: koehler@dfki.uni-sb.de

Abstract

We discuss a deductive approach to plan modification that integrates planning from second principles into a deductive planner. The logical formalism is reflected in a theorem proving approach in which the reuse component tries to prove a new plan specification using one of the generalized plan specifications stored in a plan library. If the proof succeeds the old plan can be used to satisfy even the new specification. If it fails the information for successfully modifying the old plan can be extracted from the failed proof.

1 Plan Reuse in a Deductive Planning Environment

Planning in complex domains is normally a resource and time consuming process when it is purely based on first principles. Once a plan is generated it represents problem solving knowledge which is generally lost in classical planning systems after the plan has been successfully executed. If such a planner has to solve the same problem again, it will spend the same planning effort and is not capable of "learning" from its "experience." Methods of *planning from second principles* try to reuse former problem solutions in order to make planning more efficient and flexible.
Besides planning from first principles as it is performed by the deductive planner [Biundo/Dengler], we integrate planning from second principles by incorporating a plan reuse component into it. As with plan generation we ground this plan reuse component on a deductive formalism [Biundo and Dengler, and Koehler92].
Planning and plan reuse interact in the following way: A formal plan specification (provided to the planner) is forwarded to the reuse component. If the reuse component succeeds in hunting up a plan from the library that (perhaps after minor modifications) can be reused to solve it the *plan modification* process starts. This process implements planning from second principles: It takes an existing plan together with its generation process (which in our case is a proof) out of the plan library. If the plan has to be modified, for example, by inserting additional actions, a formal subplan specification is generated and passed to the planner. The planner generates a subplan, which then is used to extend the already existing plan in such a way that it satisfies even the current specification. If no reuse "candidate" can be found the deductive planner has to generate a completely new plan out of the given specification.

2 A Four Phase Model of Plan Reuse

To formalize planning from second principles we have developed a four phase model of plan reuse reflecting the different tasks that have to be addressed [Koehler9]:

1. In the phase of *Plan Determination* a plan specification formula Φ is retrieved from the plan library to solve a new planning problem given as a plan specification formula Ψ. We presuppose that the plan library does not contain (user-)predefined plan entries, but is built up using information provided by the deductive planner, e.g., the specification formula, the generated plan, and the proof tree for the plan.

2. In the phase of *Plan Interpretation* the formula Φ has to be interpreted in the current planning situation by investigating whether Φ can be instantiated to Φ_{inst} such that Ψ is obtained.

3. In the *Plan Refitting* phase the instantiated plan specification Φ_{inst} is compared with Ψ and necessary refitting tasks for the planner are derived. Planner and plan reuse component interact in such a way that the reuse component generates subplan specifications for which the planner is activated to generate

the subplans which have to be deleted from or incorporated into the plan to be reused.

4. The reuse process finishes with a *Plan Library Update* in which the plan specification formula Ψ is generalized and compared with already stored plans. If Ψ is "worth" storing it is added to the plan library.

In the following we shortly describe how plan interpretation and refitting, summarized as *plan modification* are realized deductively.

3 A Deductive Approach to Plan Modification

The deductive formalism is worked out using the framework provided by an interval-based temporal modal logic, the so-called Logical Language for Planning LLP that is used by the planner the reuse component is interacting with [Biundo and Dengler92]. We assume that plan specification formulas given in LLP $[\mathsf{Plan}_\psi \rightarrow \psi]$ are of form $[\mathsf{Plan}_\psi \rightarrow [\psi_i \rightarrow \psi_g]]$, where the subformulas ψ_i and ψ_g describe the facts holding before executing the plan and the facts that have to be reached by it, respectively. Suppose, given a plan specification $[\mathsf{Plan}_\psi \rightarrow \psi]$ the plan determination process succeeds in finding an appropriate entry in the plan library and comes up with a specification formula $[\mathsf{Plan}_\phi \rightarrow \phi]$ and a plan formula P_ϕ that had been generated from this specification to replace the metavariable Plan_ϕ. To find out whether P_ϕ can be reused as a solution even for Plan_ψ in order to satisfy the current specification we try to prove the formula:

$$[\phi \rightarrow \psi]$$

This step is justified by the fact that $[P_\phi \rightarrow \psi]$ if $[\phi \rightarrow \psi]$, provided $[P_\phi \rightarrow \phi]$ holds. If the proof of $[\phi \rightarrow \psi]$ succeeds the "old" plan P_ϕ can be reused without any modifications. But in general, the proof of $\phi \rightarrow \psi$ will fail since the old plan will not be applicable without any modification to solve the new planning task. Therefore, we conduct the proof attempt in such a way, that from the failed proof sufficient information can be extracted to modify P_ϕ successfully.
$[\phi \rightarrow \psi]$ is attempted to be proved using a matrix calculus for the modal logic LLP based on results by [Bibel82,Wallen89]. The proof attempt consists of two steps: First the matrix corresponding to that formula has to be built and paths in the matrix are determined for which *simultaneously complementary literals* under an *admissible substitution* (meeting several criteria posed by the underlying logic) can be constructed. The result of the *plan interpretation phase*, i.e., the desired instantiation for Φ is provided by the substitution under which the number of these paths has a maximum.
Secondly, to formalize the *plan refitting phase* we analyze the remaining paths in the matrix. The formula described by the set of these paths represents refitting information from which we derive logical specification formulae representing the necessary refitting tasks for the planner. This unique characterization of refitting tasks allows to reduce the problem of plan refitting to *one* basic refitting strategy resolving the two possible plan failures, viz. *plan reduction* and *plan expansion* that have to be distinguished.

It finally has to be verified that the modification of P_ϕ leads to a correct, i.e., executable plan.

4 Conclusion

In this paper we discussed the ideas of a logical and domain-independent approach to the problem of plan reuse inside of deductive planning. The main idea is to base it on a theorem proving approach to provide the logical framework for a completely automated solution to the problem of plan modification. The developed modification method is purely based on information arising in the deductive planning process and does not require additional information supplied by a user.
Current investigations are related to the characterization of the proof attempt as a kind of subsumption test to provide the framework for further theoretical investigations of such important aspects as decidability of the test procedure, soundness, i.e., the modified plan is indeed a solution for the current goal and completeness, i.e., in general every plan can be modified to obtain a solution.

Acknowledgements

I want to thank Prof. Wolfgang Wahlster for his advice and support and my colleagues Dr. Susanne Biundo and Dietmar Dengler for their interest in my work and many fruitful discussions.

References

[Bibel82] W. Bibel(1982). *Automated Theorem Proving.* Vieweg, Braunschweig - Wiesbaden.

[Biundo and Dengler92] S. Biundo and D. Dengler (1992). An interval-based temporal logic for planning. Research report RR-92-12, German Research Center for Artificial Intelligence.

[Biundo and Dengler, and Koehler92] S. Biundo, D. Dengler, and J. Koehler (1992). Deductive planning and plan reuse in a command language environment. Research report RR-92-11, German Research Center for Artificial Intelligence.

[Koehler9] J. Koehler (1991). Approaches to the reuse of plan schemata in planning formalisms. Technical Memo TM-91-01, German Research Center for Artificial Intelligence.

[Wallen89] L. A. Wallen (1989). *Automated Deduction in Non-classical Logics.* MIT-Press, Cambridge, London.

Abduction and Planning

Alastair Burt
KIK-Teamware
German Research Center for Artificial Intelligence (DFKI)
Postfach 2080
6750 Kaiserslautern
Germany
e-mail: burt@dfki.uni-kl.de

Abstract

The goal of the KIK project is to investigate domain independent techniques for cooperation. We regard cooperation as an interactive process between agents, where agents are entities that carry out actions and plan their actions. Here we describe a general framework for planning, in which it is viewed as a form of hypothetical reasoning. Abduction is presented as a means to reason with planning hypotheses. We indicate how this influences our notion of inter agent cooperation.

1 Cooperation and Planning

Spurred by the increased networking of computers, cooperation is becoming the focus of attention in two ways:

- in *Computer Supported Cooperative Work (CSCW)*, where attempts have been made to formalise cooperative behaviour amongst people in the workplace in order enhance the computer support, and
- in *Distributed AI (DAI)*, where researchers are trying to find general mechanisms to coordinate software in loosely coupled systems.

A useful definition of cooperation is:

Cooperation == Distributed Planning of Actions + Coordinated Execution of Actions

An understanding of cooperation is therefore to be gained by

1. finding a general model of planning,
2. extending it to the case where several planners interact with one another. and
3. describing the relationship between planning and execution.

Below we describe a 1 and hint at how one may achieve 2 and 3.

2 Requirements for Planning

There are two key features which characterise general approaches planning:

- the means to describe state change and actions — here we use augmented event calculus, alternatives might be situation calculus or non-classical logics of action and time; and
- the means used to derive the actions that will lead to the goal state — here we use abduction, alternatives might be deduction or modifications of deduction.

This approach to planning has been developed by [Esh88] and [Sha89].

3 Event Calculus and State Change

Event calculus captures the notion of state change by pegging it onto events. One describes when events happen and what properties of the world these events initiate or terminate. The following two Horn clauses represent this:

```
holds-at(P, T) if
            happens(E) and
            time(E) < T and
            initiates(E, P) and
            not clipped(E, P, T)


clipped(E, P, T) if
            happens(E') and
            terminates(E', P) and
            not T ≤ time(E') and
            not time(E') < time(E)
```

We can capture the effects of an action by defining *initiates* and *terminates* appropriately. For example:

```
initiates(E, clear(Z)) if
            act(E, move(X, Y)) and
            holds-at(on(X, Z), time(E)) and
            Z ≠ Y

terminates(E, on(X, Z)) if
            act(E, move(X, Y)) and
            Z ≠ Y
```

Preconditions, which must hold in the world before an action can be executed, can be described in two ways:

- by extra conditions in the definitions of *initiates* and *terminates*, or
- by using the appropriate integrity constraints in the abductive scheme we outline below.

4 Abduction: A Form of Hypothetical Reasoning

Given a theory T and a goal formula G, abduction consists in finding the set of assumptions Δ such that:

$$T + \Delta \vdash G$$

There may be many possible values for Δ, in which case it will be useful to define which are good or optimal values. Possible criteria are:

- that it only contains instances of relations in a certain set α of *abducibles*.
- that it is *minimal*: that is there exists no Δ' s.t.
  $\Delta' \subset \Delta$ and
  $T \cup \Delta' \vdash G$, or
- the Δ meets the integrity constraints I, that is, $T \cup \Delta \cup I$ is satisfiable.

5 Abduction and Planning

Using abduction in the context of event calculus and planning we can restrict the abducibles to the relations *happens*, *act*, and the relations of temporal ordering. The goal G is the formula $\exists$ T *holds-at(P, T)*, where P is the property which wish to hold in the goal state. An appropriate criterion for the best Δ is that it contain the fewest instances of *happens*.
To obtain an algorithm to compute Δ we can use a extension of SLD resolution that does not fail when it cannot prove an instance of a relation in α but instead adds the instance to Δ. Complications arise when we interpret *not* as negation as failure, as is usual with event calculus. The most elegant way to handle this is to re-interpret negation as failure itself as a form of abduction.
What are the advantages of using abduction and event calculus for planning?

- It stays within well-understood first order classical logic.
- It maps easily onto an efficient extension of resolution.
- Since it stores assumptions on which a plan is based, it can be used to check that these assumptions still hold when the plan comes to be executed and to modify the plan when they do not.
- Event calculus is a very general means to reason about states, and can be extended to cope with periodic and continuous change. Moreover, given suitable definitions of *holds-at(P, T)*, P may be any reified logical formula.
- Abduction is a very general framework for hypothetical reasoning. It could incorporate the heuristics that have been suggested for efficient planning algorithms.

6 Cooperation and the Planning Model

In view of the above model of planning cooperation can be re-interpreted as:

Distributed Hypothetical Reasoning + Coordinated Realisation of Hypotheses

In the cooperative context there are several agents, $A_1, A_2, ..., A_n$ each associated with their own T_i, G_i, I_i, Δ_i and α_i. We then have a problem of how to localise the decision making when generating a particular set of assumptions. The Δ_i of agent should in some sense be acceptable to the other agents but we do not want to require that $\cup_{j=1}^{n} T_j \cup \Delta_j \cup I_j$ be satisfiable.
We must build on our model of planning in other ways to reflect the dynamic interactive nature of the multi agent system. Agent A_i not only generates Δ, but by carrying out actions ensures that the statements in Δ_i become true of the world. We would like to engineer the agent such that it demonstrably respects the commitments made in Δ_i. The search for suitable actions to carry out is not typically a once for

all affair. Rather, it is an ongoing process. The agent is constantly revising Δ, in the light of new input from the world. Our abductive architecture should handle this situation. These and related topics are the subject of current research.

References

[Esh88] K. Eshghi. Abductive planning with event calculus. pages 562–579, Cambridge, Massachusetts London, England, 1988. "MIT Press".

[Sha89] M. Shanahan. Prediction is deduction but explanation is abduction. In *Proceedings IJCAI*, 1989.

Concepts for Hierarchical Planning in a Flexible Manufacturing System

Klaus Fischer
AKA-MOD
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3
6600 Saarbrücken 11
Germany
e-mail: fischer@dfki.uni-sb.de

Abstract

The paper presents a concept for the design of the planning and controlling components of a flexible manufacturing system. The hierarchical planning structure consists of 6 layers: the production planning and control layer, the shop floor control layer, the task coordination layer, the task planning layer, the task execution layer and the machine control layer. The design of these layers is briefly described by the paper.

1 Introduction

CIM (computer-integrated manufacturing) is nowadays a catchword to which current research work is trying to give real contents. In the environment of a laboratory it is already possible to carry out a manufacturing order fully automatically, from the layout of the product to the manufacturing of the end product. At the beginning of the research work, success in special subjects, e.g. CAD (computer-aided design) systems, had been achieved. The main research work today is done on the integration of the units of a flexible manufacturing system (FMS). At first the main problem was the communication between these units. The ongoing connection of these units in local area networks and the definition of communication protocols has brought about great progress here. Now it is possible to concentrate on the real problems in the planning and controlling components of a FMS. These are the control of the flow of information and the control of the movement of the workpieces and tools in the manufacturing floor. In order to control the movement of the workpieces and tools, it must be determined how and when they have to be carried from one location to another. Controlling the flow of information requires ensuring that the data guiding the manufacturing process, e.g. how a workpiece has to be worked on, accompanies with the physical transport of the workpieces. The concept presented in this paper is a framework for the design of the planning and controlling components in a FMS.

2 The Production Planning and Control Layer

The production planning and control (PPC) system is the highest controlling body to reach the goals defined by the management of a company. The registration of client orders takes place in the PPC system where they are treated by economical criteria. Here the classical functions: *administration of client orders, data management, material management*, and *time management* are treated. The horizon of planning is a day up to a week. Planning of a specific client order is done with respect to the finishing date and the sequence of the work steps to be done. In doing so a rough estemation of the capacities of the resources needed for this client order is done [SFB331]. By an off-line planning step work plans are given to the PPC system which determine the manufacturing sequence for a certain end product. These work plans are passed as manufacturing orders to the shop floor control system where they are executed. All economical aspects in planning are handled by the PPC system and during the design of the working steps and work plans by a product engineer. For the deeper planning layers it is above all important to execute the production orders defined by the PPC system efficiently with respect to the time constraints specified by the PPC system. The more deterministic the execution of the production orders take place, the better it is possible for the PPC system to plan the execution of the client orders. Because the planning is done using statistical data, variations in the execution of the manufacturing orders due to autonomous planning decisions can be registered. Autonomous planning decisions can be necessary because of the occurrence of error and exceptional situations (e.g. damage of a tool or a workpiece, collision etc.) or because of the break down of resources (e.g. machine tools, robots etc.).

3 The Shop Floor Control Layer

The central problem to be solved in the shop floor control (SFC) system is the control of the flow of material. Controlling a FMS means reaching two conflicting goals: *minimal costs of inventory by minimizing the duration of the production order in the shop floor* and *a maximal rate of capacity utilization of the machines.* It is impossible to satisfy these two goals simultaneously [Müller70]. Therefore, a compromise has to be found between these two goals. By increasing the technological level the resource 'time' becomes more and more important for competitiveness, because advantages in time can become advantages in costs [Milberg90]. Hence, saving of time is an important aspect in a FMS and, therefore, it is common agreement that it is good to reach the first of the two goals and to get additionally a reasonable capacity utilization of the machines. The global goal here is to have a high flexibility in the design of products and in the production of these products in order to be able to react quickly to changes of the market.

4 The Task Coordination Layer

Looking at the hierarchical planning structure from above, the task coordination layer (TCL) is in the first planning layer which is present for each autonomous unit separately. The planning and controlling layers from the TCL to the machine control layer (MCL) group to a logical unit which is unified with the term 'autonomous unit' (AU). The SFC system passes the task to the TCL immediately when it is determined by the production plan that the task may be executed because all of the preceding working steps have been completed. The SFC system does not care if it is possible for a group of AUs to execute this task immediately or if they are currently engaged in the execution of a task. The task is just announced by the SFC and the AUs decide by themselves when it will actually be executed. By doing a specific task several AUs have to cooperate. Each AU has to play a part to solve a specific task. No AU may believe that it is the only one which wants to play a certain part for a specific task. Therefore, the AUs must coordinate their intentions of playing parts in different tasks on the TCL.

5 The Task Planning Layer

The TCL of a AU passes a single task to the task planning layer (TPL) of the AU which has to be solved in order to solve the global task given on the TCL. The planning of the task is done using the content of a global knowledge base. This global knowledge base contains an image of the current state of the real world. By solving a task the AUs have to cooperate with each other. For that reason, the complex task given by the TCL has to be decomposed into primitive actions each of which can be executed by the AU without interacting with other AUs. The task planner contained in the TPL of an AU therefore implements the function:

$$f : \mathcal{K}^* \times \mathcal{T} \longrightarrow \mathcal{P}^*$$

where $\mathcal{K}$ is the set of a all possible knowledge base states, $\mathcal{T}$ is the set of all tasks, and $\mathcal{P}$ is the set of primitive actions. The input to the task planner is a sequence of knowledge base states because the content of the knowledge base is changed while task planning is active. Hence, the task planner reacts to changes in the environment of the AU by reacting to the changes in the knowledge base. For each task function f is specified by a set of rules which we call *behaviour pattern*. A single rule of a behaviour pattern is called a *behaviour rule* [Fischer88,Fischer89].

6 The Task Execution Layer

The primitive actions derived in the TPL of an AU are executed in the task execution layer (TEL) of the AU. By executing primitive actions an AU has not to interact with other AUs. Nevertheless, such a primitive action may be itself a complex task to do, for example the sensor-guided insertion of a pin into a hole by a robot. [Hagg92] describes the concept of sensor actor networks with which the solution of such tasks may be specified.

7 The Machine Control Layer

Simple control commands are sent from the TEL to the machine control layer (MCL) where the actions are actually executed. The separation of the TEL and the MCL is done due to the systems which are available today. The controlling units of these robots and machine tools are simple computation units with weak computational power, which are therefore not able to do complex computations. In the future it is likely that the controlling units of such systems will offer the computational power of today's workstations and that therefore the TEL and the MCL will grow together.

References

[Fischer88] Fischer, Klaus: Regelbasierte Synchronisation zwischen Roboter und Maschinen, Technischer Bericht, Institut für Informatik, TU München.

[Fischer89] Fischer, Klaus: Knowledge-Based Task Planning for Autonomous Mobile Robot Systems, Proc. of the 2nd Inter. Conf. on Intelligent Autonomous Systems, Amsterdam, pp 761–771, December 1989.

[Hagg92] Hagg, Ernst: Realisierung von Multisensoranwendungen mit vernetzten logischen Sensoren und Aktoren, Doktorarbeit, Institut für Informatik, TU München, 1992.

[Milberg90] J. Milberg: Eine Woche lang CIM — CAD — CAM, Süddeutsche Zeitung, 22.10.1990.

[Müller70] Müller-Mehrbach, Heiner: Optimale Reihenfolgen, Springer-Verlag, Berlin 1970.

[SFB331] Sonderforschungsbereich 331 — Informationsverarbeitung in autonomen mobilen Handhabungssystemen — Arbeits- und Ergebnisbericht — Januar 1986 - Dezember 1988.

PIM - Planning In Manufacturing

Ansgar Bernardi Christoph Klauck Ralph Legleitner
ARC-TEC
German Research Center for Artificial Intelligence (DFKI)
Postfach 2080
6750 Kaiserslautern
Germany
e-mail: bernardi@dfki.uni-kl.de klauck@dfki.uni-kl.de
legleit@dfki.uni-kl.de

Abstract

In order to create a production plan from product model data, a human expert thinks in a special terminology with respect to the given work piece and its production plan: He identifies certain areas of interest, the so-called application features. The exact form of these features is influenced by his manufacturing environment (e.g. available tools) and by his personal experience. The expert associates the application features with fragments of a production plan. By combining these fragments, bearing in mind some general principles, he creates the complete production plan.

We present a set of representation formalisms which allow to model this approach very closely. Based on TEC-REP (TEChnological REPresentation), a general representation formalism for geometrical and technological information about the work piece, an expert's application features are defined. They are described using the language FEAT-REP (FEATure REPresentation) and represent his personal terminology. Skeletal plans (abstracted plans or fragments of plans), represented in the hierarchical formalism SKEP-REP (SKEletal Plan REPresentation), are associated with the features.

When an expert's knowledge about production planning has been formalised in terms of application features and associated skeletal plans, the generation of a production plan boils down to a sequence of abstraction, selection and refinement: The geometrical/ technological representation of a work piece allows the recognition of the relevant features. The associated skeletal plans are selected, merged and refined until a complete plan is created. This is demonstrated in the CAPP-system PIM (Planning In Manufacturing), which is currently developed as a prototype. .

The representation formalisms and the prototypical implementation have been designed with special focus on the possible integration into existing CIM-chains. We provide interfaces to CAD and to NC machines.

The approach sketched above formalizes the knowledge of the concrete expert (or the accumulated know-how of a concrete factory). By employing the expert's feature definitions it is possible to create planning systems which are especially tailored to the concrete manufacturing environment and optimally use the expert's knowledge. The close modeling of the expert's planning methods and his terminology should also lead to improved acceptance of the system.

1 TEC-REP

The representation formalism TEC-REP [BKL91]provides the necessary constructs to describe the geometrical and technological information of the work piece. The geometry of the workpiece is described by surface primitives. These predefined primitives include simple expressions for rotational symmetric parts, which are of special interest in the domain of manufacturing by turning. To be as universal as possible, primitives for non-symmetric surfaces also exist. The extensions of any surface are specified using a cartesian coordinate system. The concrete dimensions are specified for each work piece. Every surface of a work piece description can be identified by a unique identification number. The neighbourhood of surfaces is explicitly represented by a separate primitve. The technological informations are described by attributes to the surfaces they concern.
In summary, TEC-REP allows the description of every work piece we currently deal with by using a symbolic and attributed boundary representation.

2 Features and FEAT-REP

Features represent a concrete expert's knowledge about characteristic aggregations of surfaces. We define the term feature as a description element based on geometrical and technological data of a product which an expert in a domain associates with certain informations (see also [KBL91]).
The features can be described by means of formal languages via attributed node-label-controlled graph grammars (ANLCGG's) [KBL91].
Each feature can be associated with knowledge about how the feature should be manufactured; this information can be used to generate a process plan. Given the geometrical/technological description of a workpiece, all relevant features can be identified by feature recognition.
To show the usability of our high-level-representation language FEAT-REP the FEAT-PATR-System was implemented as a prototypical part of PIM, by adopting a chart parser for our application in mechanical engineering. Input of our FEAT-PATR-System is a workpiece description in TEC-REP. Input is also the expert's feature knowledge about the workpiece, represented in a grammar. Output of the system is a feature-structure of the workpiece

3 Skeletal Plans

To combine the expert's knowledge about the manufacturing process with feature structures as shown above we use skeletal plans [FI85]. The skeletal plan representation formalism SKEP-REP allows the expert to write down his knowledge about the process necessary for the manufacturing of his workpieces and for special parts of this workpieces (features).

A skeletal plan described in the formalism SKEP-REP contains the *feature* or feature structure it is associated to. It then contains some *context information* which relates to other skeletal plans which form preconditions for the application of this particular plan. It may also contain some *applicability constraints* which are not expressed by the features or the context of skeletal plans. Then it contains a sequence of *operations*, which may result in the subroutine-like call of other skeletal plans or in the generation of concrete planning steps. In the domain of manufacturing by turning, concrete planning steps are chucking commands, cut instructions and tool selections.
Every operation may access the concrete technological and geometrical informations, especially measurements, which are represented in the TEC-REP of the surfaces which form the features associated with the skeletal plan or with the plans of the context (see above). Beside this, no information about the workpiece can be accessed. This realizes the concepts of modularity and information hiding for the skeletal plans and makes it possible to create skeletal plans for for a large bandwith of workpieces.
Some of the operations can result in the subroutine-call of special programs for particular tasks. In our prototype, the tool selection operation uses a constraint system to find the suitable group of tools for the intended operation.
To perform the selection and merging of the skeletal plans, a prototypical skeletal planning system was implemented as a part of our PIM system [Bec91], which uses the following algorithm:
When a given workpiece description has been transformed in a feature structure by the feature recognition process, the skeletal plans associated with these features are found and selected according to the constraints embedded in the plans. The resulting set of skeletal plans is then merged to one final plan, and abstract variables are replaced by the concrete data of the workpiece in question. The merging of the skeletal plans is oriented on several topics: Operations using the same tool should be performed consecutively (minimalization of tool change operations). Operations in one chucking context must be performed together, minimizing the changes of chucking. Different tools belonging to a common group may be exchanged against a more general tool of this group, such that several operations using slightly different tools can be merged to one operation using only one tool. Different surfaces of a workpiece which are treated with similar operations should be grouped together. These merging operations are supported by a hierarchical ordering of the available tools and a hierarchical grouping of the possible operations. Some heuristical approaches to skeletal plan merging are under investigation. The expert may influence the merging operations by explicit merging commands.

4 Connections to the real world

The PIM system fits into the CIM-chain by interfaces to existing CAD systems and to existing NC programming systems.
The interface to the CAD world transforms the necessary geometrical/technological

information about the workpiece from the CAD data into our TEC-REP. A translator for the forthcoming ISO-standard STEP [GASS89,BKL90] is currently under implementation.
Another interface connects the CAD system "Konstruktionssystem Fertigungsgerecht" of Prof. Meerkamm, University of Erlangen [MW91], with TEC-REP. This system is a augmented CAD system based on SIGRAPH. It uses an internal data format which can directly be transformed into TEC-REP. Thanks to the cooperation of Prof. Meerkamm and his team, this connection works satisfying.
To get connected to the NC-machines we rely on components of NC programming systems available on the market today. The primitive operations used in the skeletal plans can easily be compiled into the command language of the NC programming system, which subsequently can generate CLDATA Code as well as machine specific NC-Code without any further human interaction.

5 Conclusion

The observation of human expert's problem-solving behavior resulted in a model of process planning which supports a knowlegde-based approach to CAPP. Based on technological/geometrical information about the workpiece, higher-level features are defined and associated with skeletal plans. Special languages for a adequate representation of the necessary knowledge on the different abstraction levels were presented. The transformation and interpretation steps between the different languages have been implemented and form the planning system PIM. The resulting system is especially tailored to a concrete manufacturing environment and uses the expert's knowledge optimally. This should lead to good quality of the produced plans and to high acceptance of the system. Positive comments of many domain experts support this claim.

References

[Bec91] Andreas Becker. Analyse der planungsverfahren der ki im hinblick auf ihre verwendung in der arbeitsplanung. Dokument D-91-17, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Postfach 20 80D-6750 Kaiserslautern, september 1991.

[BKL90] Ansgar Bernardi, Christoph Klauck, and Ralf Legleitner. Step: Überblick über eine zukünftige schnittstelle zum produktdatenaustausch. Dokument D-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Postfach 20 80, D-6750 Kaiserslautern, september 1990.

[BKL91] Ansgar Bernardi, Christoph Klauck, and Ralf Legleitner. Tec-rep: Repräsentation von geometrie- und technologieinformationen. Dokument D-91-07, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Postfach 20 80, D-6750 Kaiserslautern, june 1991.

Cha90] Tien-Chien Chang. *Expert Process Planning for Manufacturing*. Addison-Wesley, 1990.

FI85] Peter E. Friedland and Yumi Iwasaki. The concept and implementation of skeletal plans. *Journal of Automated Reasoning*, 1:161–208, october 1985.

GASS89] H. Grabowski, R. Anderl, B. Schilli, and M. Schmitt. Step - entwicklung einer schnittstelle zum produktdatenaustausch. *VDI-Z*, 131(9):68-76, september 1989.

KBL91] Christoph Klauck, Ansgar Bernardi, and Ralf Legleitner. Feat-rep: Representing features in cad/cam. In *IV International Symposium on Artificial Intelligence:Applications in Informatics and International Conference on Manufacturing Automation*, 1991.

LBK91] Ralf Legleitner, Ansgar Bernardi, and Christoph Klauck. Pim: Skeletal plan based capp. In *International Conference on Manufacturing Automation*, 1991.

MW91] H. Meerkamm and A. Weber. Step - entwicklung einer schnittstelle zum produktdatenaustausch. In *Erfolgreiche Anwendung wissensbasierter Systeme in Entwicklung und Konstruktion. VDI-Berichte 903*, pages 231–249, 1991.

Multi-agent Planning

Norbert Kuhn Jürgen Müller
AKA-MOD
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3
6600 Saarbrücken 11
Germany
e-mail: kuhn@dfki.uni-sb.de mueller@dfki.uni-sb.de

Abstract

In contrast to the planning of actions for a single agent, planning in a multi-agent environment is concerned with the construction of plans for several agents that act in the same world. This implies that there may be either conflicting or common goals in the different agents' plans that need to be coordinated. Thus, an important aspect in a multi-agent society are the ways of interaction that can occur between the agents.

Our main interest is in multi-agent systems where the agents try to cooperate, i.e. they are consciously acting together to reach a common goal and they coordinate their activities by negotiations and by making arrangements. Therefore, the agents must have capabilities to reason about their own activities and other agents' actions in order to resolve conflicting situations in a cooperative manner.

We present a first approach to deal with the quantifiable aspects of cooperation which can also be used to solve a second major problem of multi-agent planning, namely the problem of task decomposition and task allocation. The solution of this problem is crucial for exploiting the synergetic effects from an agent-society. Unfortunately, a synergetic effect is system-immanent most of the times and it is hardly to be captured by some numerical function. Thus, we also deal with some sociological aspects of cooperation, such as the effect of the structure of the groups that are working together and of the communication features on the global functionality of a community of agents.

1 Introduction

The task in multi-agent planning is to construct sequences of actions for several different units that act in the same world. This can be done by a distinguished central planner as well as by a set of differerent decentralized planning units. In either case the existence of a society of agents has some impacts on the planning process itself. ¿From an agent's local point of view this means that he should take care of other agents' actions already at the planning stage. From a more technical point of view the system should at least provide facilities to resolve possible conflicts, e.g. synchronization mechanisms for resource allocation.
Another problem that has to be addressed is the finding of an appropriate task decomposition, i.e. breaking down a given task into subtasks and assigning them to the available agents. Thus, in multi-agent planning systems we will have in general at least the two phases of task decomposition and the individual planning of actions. Assuming that every agent has only incomplete information about the other agents and they cannot predict their actions leads to the necessity of a third phase in the planning process, the coordination of the individual plans.
In the following we first present some possible approaches to plan coordination to motivate our model of decentralized multi-agent planning. This model should enable us to handle task decomposition and the coordination of the individual plans in the same manner.

2 Coordination of individual plans

To fulfill given tasks the agents will develop sequences of actions. By performing specific actions the agents will obstruct each other, e.g. if two agents compete for the same resource, or parts of the plans may include the possibilty of mutual support. Thus, there are relations between the sequences of actions which can be either possible obstructions or that may include the possibility that the agents can 'help' each other. In the coordination phase all the obstructions should get eliminated and the possibilities of supporting each other should be exploited.
The handling of the obstructing relationships between sequences of actions is also of some relevance in other kinds of distributed systems, like distributed operation systems or distributed databases, and a lot of mechanisms to deal with this kind of relation have been developed. These mechanisms may be applied in our setting, too. But, in general these mechanisms lack of exploiting the intelligent features of the agents.
Some of the approaches for coordinating agent activities apply the communication of partial plans between the agents. If those partial plans include for example a competetive request for some resource the corresponding agents have to resolve this conflict via negotiation. In the solution of v.Martial [vM92] the agents try to modify (shift or shrink) the time intervals for which they require a common resource to achieve an agreement.
Another possibility for controlling resource allocation of the agents is the use of an

artificial market mechanism. We endow the agents with an amount of money to pay for the use of the resources within a specific time period. Every single resource is associated with a price which may vary over time. These so called transfer prices [Mey80] can be used to control the access to the resources.
The general idea is the following: To fulfill a certain task the agents may have to choose between the use of different resources. This choice is done by evaluating a private function that yields a priority on the resources considered. One variable in those functions should be the price associated with each resource. By modifying the prices we try to influence these functions to achieve a good rate of resource utilization. The following simple example illustrates this idea:
Let a_1, a_2 be agents, and r_1, r_2 be two equal resources with the associated prices p_1 and p_2. If agent a_i wants to use resource r_j he will have the costs $c_{ij} = p_j + w_{ij}$, where w_{ij} is a_i's extra cost for getting r_j.

a_1	1	4
a_2	1	2

If the w_{ij} are as in the table to the left then for each agent the cost of using r_1 is less than the one for using r_2. Thus, both agents will run for r_1, and we are in a situation that we have a conflict for the reservation of one resource while an identical one stays unused. But, if we increase the value of p_1 by 2, i.e. $p_1 := p_1 + 2$, we see that for agent a_2 the cost now for r_2 is less than that for r_1. So he will revise his decision made up so far and the conflict will be solved.
The modification of the prices can be done in various ways - e.g. by a central unit, by communication between the resources or by putting up the resource for auction. The use of a market model for controlling resource allocation in a multi-agent system provides a mechanism to quickly resolve conflicts for reservation. In addition, the change of the reserve of money yields varying priorities of the agents. Of course, the model is refinable to take care of optimality criteria or to model cooperation between agents.
In the modelling of the transportation companies the market model can be used to control the task allocation in the same way as the resource allocation. At every time point t there is a set of open tasks and a set of companies with open capacities. Every task has associated a profit and the agents try to estimate their costs when they would accept this task. Then they bid for a contract for the task they rated best. Conflicts can then be solved by modification of the profit associated with some task.

3 Teams of agents

One of the main motivations of representing real-world problems by multi-agent scenarios is the hope to be able to catch the phenomenon of a synergetic effect, i.e. to be able to build up systems where the whole is more than the sum of its parts. Synergetic effects have been the matter of attention in many areas of research, in particular in psychology and sociology. For us, the studies concerned with working groups or teams are of special interest. A group structure that is appropriate for a special problem raises the probability that a synergetic effect can be achieved.

Forster [For82] distinguishes three areas where teamwork has advantages over single-handed work:

- advantage in the field of *Carrying and Lifting*
- advantage in the filed of *Searching and Finding*
- advantage in the field of *Classifying and Standardizing*

Implemented multi-agent systems where these effects are demonstrated are found through the last decade. One of the latest is ARCHON [Jen91], where different expert systems are linked together. These expert systems are able to communicate some of their current hypotheses. For instance, there is one expert system for high voltage diagnosis and one system weather forecast. If there will be a power failure and a thunderstorm, this will lead to the hypothesis of the global system that there may be some damage by lightning. Another famous example is the blackboard system HEARSAY-II [EHRLD80], where different agents work together for analyzing a given sentence in natural language.
All these sytems have in common that the synergetic effects observed are not measurable by some numerical function. And, one of the main variables effecting the functionality of the global system is the structure of the communication between the agents.
Thus, a multi-agent system we have in mind should be comprised of teams of agents. The teams themselves always have a common goal to achieve. They consist of a small number of agents, where every agent fulfills a specific task. Communication should be possible almost direct, i.e. agent-to-agent. The membership of an agent in a team is fixed, while teams may build up groups whose combination as well as the form and the degree of cooperation may vary over time. Each task in the system is assigned to a certain team which is responsible for the fulfillment of the task. If the team is not able to guarantee this it has to look for other teams for help.

4 Conclusion

We have presented some ideas to show how to incorporate coordination and cooperation into multi-agent planning systems. The agents' architecture could be based on the RATMAN model [BM91]. We motivated the division of cooperative aspects into quantifiable and non-quantifiable aspects. The quantifiable ones may be captured by the evaluation of some weighting function that controls the behavior of the agent in the near future. The non-quantifiable aspects heavily depend on the application domain and should be represented by an appropriate communication structure of the multi-agent system we design.

References

[BM91] H.-J. Bürckert and J. Müller. RATMAN: Rational Agents Testbed for Multi-agent Networks. In *Decentralized AI Vol. 2*. North Holland, 1991.

[EHRLD80] L.D. Ermann, F. Hayes-Roth, V.R. Lesser, and D.R.Reddy. The HEARSAY-II speech understanding system: Integrating Knowledge to resolve uncertainty. In *Computing Surveys 12(2)*, pages 213–253, 1980.

[For82] J. Forster. Teamarbeit. Sachliche, personelle und strukturelle Aspekte einer Kooperationsform. In W. Grunwald and H.-G. Lilge, editor, *Kooperation und Konkurrenz in Organisationen*, pages 153–168. UTB, 1982.

[Jen91] N.R. Jennings. Cooperation in Industrial Systems. In *ESPRIT '91 Conference Proceedings*, pages 253–263, 1991.

[Mey80] G. Meyer. *Dezentrale Planung - Betriebswirtschaftliche Analyse ausgewählter Dekompositionsmodelle.* Verlag Dr. Peter Mannhold, Düsseldorf, 1980.

[vM92] F. v. Martial. *Coordinating Plans of Autonomous Agents.* PhD thesis, University of Saarbrücken, 1992.

DFKI Workshop on Planning

Susanne Biundo, Franz Schmalhofer (Eds.)

D-92-07
Document