Par-KAP: a Knowledge Acquisition Tool for Building Practical Planning Systems

Leliane Nunes de Barros and James Hendler*
University of Maryland
Department of Computer Science
College Park, MD 20742, USA
leliane@cs.umd.edu, hendler@cs.umd.edu

V. Richard Benjamins+
Artificial Intelligence Research Institute
CSIC, Campus UAB, 08013 Bellaterra
Spain, richard@iiia.csic.es, and
University of Amsterdam, SWI
Roetersstraat 15, 1018 WB Amsterdam
The Netherlands

Abstract

Recently, attention has been focused on providing Knowledge Acquisition (KA) support for building practical planning systems. Such support is needed to guide a knowledge engineer in selecting planning methods, as well as for building and validating the planning knowledge-base for a given practical domain. Following current practice in knowledge acquisition, developing KA tools for planning requires that a number of planning knowledge components are made explicit. This includes explicating (i) a planning domain ontology, (ii) a library of problem-solving methods (PSMs) used in planning, and (iii) a set of domain requirements that are used to select a suitable PSM. In this paper, we summarize the planning knowledge components which we have identified in previous work, and, based on these, present an implementation (Par-KAP) that can exploit these models to aid knowledge engineers in constructing practical planning systems.

1 Introduction

Constructing a planner for a particular application is a difficult job, for which few knowledge acquisition tools currently exist. Due to an increasing need for the building of planning systems that can handle real world applications, knowledge engineering efforts need to focus on the following questions:

1. Is a particular practical planning system from the literature suitable for a given domain?
2. What type of domain knowledge is required for the application and how can it be represented?
3. What are the most suitable planning methods for the domain?
4. What control strategy can best satisfy a desired system performance?

Although much is known about planning systems, and the literature is extensive (cf. [Tate *et a*/., 1990; Tate, 1996], the focus of past work has primarily been on the development of planning systems, as opposed to answering questions such as those above. At the same time, while knowledge acquisition research has focused on such questions for other AI tasks [Schreiber *et al.,* 1994], little attention has been given to planning. In is only recently that research has started to focus on this sort of KA for planning [Valente, 1995; Benjamins e*t ai,* 1996; de Barros *et al,* 1996; Cottam and Shadbolt, 1996; Chien, 1996; Tu and Musen, 1996]. This work primarily builds on modern approaches to Knowledge Acquisition which stress the importance of libraries with reusable modeling components to support the knowledge engineer in constructing the required system model [Breuker and van de Velde, 1994]. Examples of library constituents include domain models, domain ontologies, generic tasks, problem-solving methods, inference structures, control knowledge, etc.

This paper builds on the work referred to above, and explores the use of a library of problem-solving methods for planning, which consists of three main building blocks.

1. A set of typical *knowledge roles* used in planning methods. These roles characterize the main types of domain knowledge used in planning, e.g. the domain ontology for planning. They also help in understanding the way knowledge is structured by providing an index to the *domain models* used to play these roles.
2. A set of *basic methods* used in composing a planning strategy. A *task-method decomposition structure* indexes these basic methods by defining the different ways in which a planning task can be (recursively) decomposed into subtasks
3. A set of *suitability criteria* for problem-solving methods which is used to specify the connection between the knowledge roles and the basic methods by defining what domain knowledge a method needs in order to be applicable in a particular application.

*This research was supported in part bv grants from ONR (N00014-J-91-1451), ARPA (N00Ol4-94-IO9O, DAST-95-C003, F30602-93-C-0039), and the ARL (DAAH049610297). Dr. Handler is also affiliated with the TIM Institute for Systems Research (NSF Grant NSF EEC 94-02384).Leliane is also supported by CNPq,a Brazilian Research Grant.

*Author is supported by the Netherlands Computer Science Research Foundation with financial support from the Netherlands Organisation for Scientific Research (NWO), and by the European Commission through a Marie Curie Research Grant (TMR).

In previous work, we have presented different parts of the library: knowledge roles and domain models [Valente, 1995], basic methods and the task-method decomposition structure [de Barros *et* a/., 1996], and suitability criteria [Benjamins *et ai,* 1996]. In this paper, we focus on the extension of these methods to control knowledge, and on the implementation of a KA system for supporting the development of planners. In Section 2, we briefly summarize our past work on a KA framework for planning and on how it is extended for control knowledge. This is followed in Section 3 with a discussion of Par-KAP, an implemented KA prototype that uses our framework. In Section 4 we give examples of how Par-KAP supports the construction of planning systems. It should be noted that the aim of this work is not to cover all existing planning methods in the AI literature[1], but to show how an extendible library can help to construct planners for particular application domains by providing a high-level, abstract synthesis of the available planning methods.

2 A library of PSMs for planning

In this section we present an overview of our previous work and discuss the extension to control knowledge for planning.

2.1 A domain ontology for planning

One of the critical elements in the analysis of a planning method is specifying the different *roles* that domain knowledge plays during the planning process. In the planning literature, domain knowledge is defined as static knowledge about the world which is only consulted during planning, but not manipulated. However, from a KA perspective, we must consider how this same knowledge is used by the planner itself in defining its dynamically changing "model" of the world. Thus, we identify two roles for domain knowledge: static and dynamic [Schreiber et a/., 1994].

Static knowledge roles in planning

Figure 1 shows the hierarchical organization of static knowledge roles for planning [Valente, 1995]. The leaves of the hierarchy of static roles are associated with the types of domain knowledge (domain models)[2] that can play these roles (through the "plays" relation).

The plan model role defines what a plan is and what it is made of. It consists of two parts: a world description and a plan description.

The world description role describes the world in which planning occurs. It is comprised of two sub-roles. (1) The state description role, which contains the knowledge necessary to represent or describe the state of the world (for example, a set of first order predicates as in STRIPS or a set of fluents from the Situation Calculus [McCarthy and Hayes, 1969]). (2) The state changes role explicates the information connected to the specification

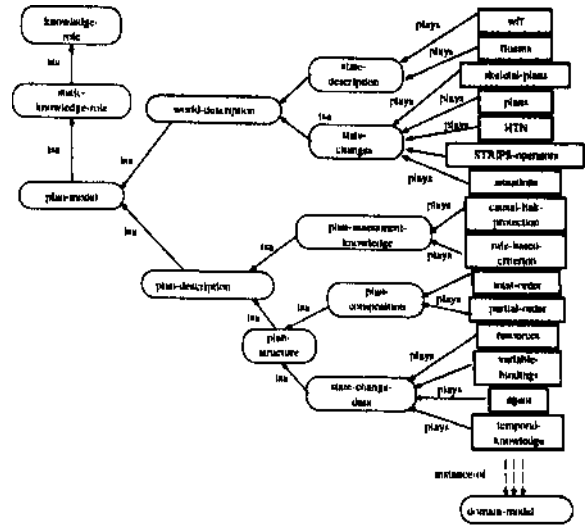

Figure 1: Hierarchy of static roles in planning and the corresponding domain models that can play these roles.

of changes in the state of the world (e.g., STRIPS-like operators or hierarchical task networks (HTNS)).

The plan description role describes the structure and features of the plan being generated, and comprises two sub-roles: plan structure and the (optional) plan assessment knowledge. (1) The plan structure role specifies how the parts of a plan (actions, sub-plans) are assembled together. It has two sub-roles: (a) the plan composition role, which describes whether the plan composition is total or partial order, whether it includes iteration and/or conditional operators, and whether the composition is hierarchical (i.e. whether plans can be recursively decomposed into *sub-plans),* (b) The state change data role contains additional information about the plan such as interval constraints for binding the variables involved in the state changes. It is also possible to assign different *resources* to each state change or sub-plan. Two particularly important resources are agents and time. (2) The plan assessment knowledge role determines whether a certain plan (or sub-plan) is valid (hard assessment knowledge), or whether a plan is better than another (soft). Based on this knowledge, a plan can be modified or criticized. An example of hard plan assessment knowledge is a *rule-based-criterion* which could be used to find out if a condition is true at some point in the plan (as in the modal truth criterion in TWEAK [Chapman, 1987]). Another example is the *causal-link* knowledge used in SNLP [McAllester and Rosenblitt, 1991]. An example of soft assessment knowledge would be the user preferences which can guide planning in the SIPE planner [Tate *et al,* 1990].

Dynamic roles in planning

Dynamic knowledge roles characterize planning in terms of the relevant variables whose values are constantly updated during the planning process. The dynamic roles include: (1) The current state role, which is initially filled

[1]In particular, we concentrate on what are known as "classical" planners as defined in [Hendler and McDermott, forthcoming].

[2]Our use of the term *domain models* for this knowledge is based on the use of the term "model" in KA, as opposed to in planning.

by a description of the world at the beginning of the plan, but is subsequently modified to represent intermediary states in the plan. (2) The goal role, which describes the active goal or subgoal being worked on by the planner. The content of goal can be a set of conditions or a set of actions to be accomplished. Initially, this role points to the original goal, and during planning it may be updated with subgoals or decompositions of the original goal. (3) The conflict role contains the result of checking the plan for inconsistencies with respect to its conditions. (4) The plan role is a composite role consisting of (a) plan-steps, (b) ordering constraints over the plan-steps, (c) variable binding constraints, and (d) auxiliary constraints that represent temporal and truth constraints between plan-steps and conditions.

2.2 The tasks and methods for planning

Based on an analysis of many classical planning systems, we have identified relevant tasks and problem-solving methods. We organize these into a task-method decomposition structure [Orsvarn, 1996] and show these in Figure 2 (where ellipses represent tasks and rectangles methods). A method *executes* (solid lines) a number of subtasks and a (sub)task can be *performed by* alternative (dashed lines) methods. The leaves of the task-method tree are called *primitive-methods* and the tasks they perform, *primitive-tasks*. Methods have two additional types of knowledge associated with them (not shown in the figure: control knowledge (Section 2.3) and suitability criteria (Section 2.4).

The class of planners we are dealing with share a general, high-level problem-solving method called propose-critique-modify (PCM) [Chandrasekaran, 1990][3]. That is, the planners all contain in one way or another these three basic tasks: (i) *propose expansion,* (ii) *critique plan* and (iii) *modify plan.* Planners differ in the problem solving methods (PSMs) they use to perform these three tasks. These differences also reflect how planning knowledge is represented. For example, in Figure 2, the propose-] method consists of the three subtasks: select goal, propose expansion, and test for unachieved goals. The propose expansion task can, in its turn, be realized by three different methods: smart propose, goal achievement propose, and decomposition propose. For a detailed description of all tasks and methods involved, see [de Barros *et* al., 1996].

2.3 Control knowledge

The task-method decomposition discussed above, defines each method in terms of (sub)tasks. During planning, these subtasks can be executed in various ways by the application of different control regimes. In fact, many planning systems from the literature differ from each other only with respect to their control knowledge. Therefore, we associate control knowledge with every PSM that can be decomposed into sub-tasks. Control knowledge includes the steps of a strategy, the order between them, conditions, loops, backtracking points, exit points, etc.

[3]In the literature, planning algorithms are usually not described in a common terminology. We have tried to choose fairly generic names that capture the whole array of these approaches.

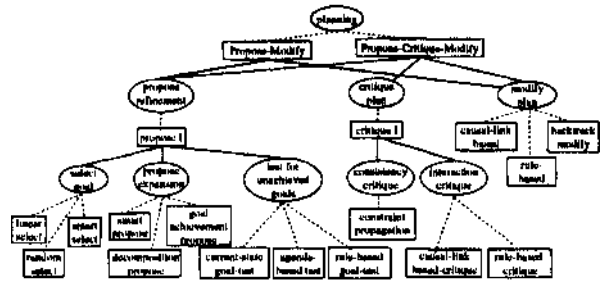

Figure 2: The task-method decomposition structure. Solid lines stand for *executes* (a method executes its subtasks), dashed lines denote *performed-by* (a task can be performed by alternative methods).

The specification of control knowledge is important because it lets us relate the individual tasks and methods to specific planners. We do this by recognizing certain patterns of control as corresponding with well-known planners when possible. In some cases, we can recognize a well-defined pattern corresponding to a planner from the literature, and recommend to a user the use of that algorithm (i.e., STRIPS, NONLIN, SIPE, SNLP, UCPOP, etc). In other cases, we can recommend certain control regimes that may be of use in implementing a planner for the specific application. We refer to the first of these cases as a match with a "fully-specified" control regime, and the second as matching a "partially-specified" control method.

As an example of a fully-specified control structure, the following would be the definition in our system of the SNLP planner [McAllester and Rosenblitt, 1991].

1. test-for-unachieved-goals (method: agenda-based-test)
2. if goal = empty then exit
3. else select-goal (method: random-select)
4. propose-expansion (method: goal-achievement-propose); *backtracking-point*
5. interaction-critique (method: causal-link-based-critique)
6. consistency-critique (method: constraint-propagation)
7. if conflict ≠ empty then
8. modify-plan (method: causal-link-based-modify); *backtracking-point*
9. else ; *recursive-invocation*

Besides these fully defined algorithms, our analysis of classical planning systems shows that there are some other control features that are common between multiple algorithms. For example, in many planners a *test-for-unachieved-goals* task is used as a termination-point, the *propose-expansion* task is a backtracking point, a *select-goal task* is always executed before the *propose-expansion* task, etc. We can exploit this by including these pieces of control knowledge in the KA system for making suggestions when one of the known control regimes is not a perfect fit. An example of the specification of one of these partially-specified control structures is the method *propose-critique-modify:*

Tasks:
STEP-1. propose; *isa backtracking-point and exit-point*

STEP-2. critique; *isa fail-point*
STEP-3. modify; *isa backtracking-point* and *has-a cond:* (conflict ≠ empty)
Control ordering: STEP-2 *is-before* STEP-3

2.4 The suitability criteria

The use of suitability criteria for establishing the applicability of methods based on the domain specification is an important part of knowledge acquisition [Benjamins, 1995]. In our work, each planning method is associated with such criteria to specify constraints on the domain features used to fill a knowledge role. For example, the decomposition propose method requires that the static role world description is fulfilled by the domain model HTN. The smart propose method requires the two role fillings plan composition = total order, and world description = STRIPS-like operators. Similarly, suitability criteria are used to establish the connection between the control structure of a method and the domain knowledge. A complete description of the suitability criteria we use is beyond the scope of this paper, but see [Benjamins *et al.*, 1996] for a complete discussion.

3 Par-KAP: an implemented KA tool

As discussed in the introduction, the goal of defining a KA library such as the above one is to allow the implementation of a KA tool for developing planning systems. We have implemented a system called Par-KAP which uses the knowledge framework above for this purpose. In this section we briefly present implementation details and examples of the use of the system.

3.1 Implementation

Par-KAP (for Parka for Knowledge Acquisition in Planning) is implemented using the Parka knowledge representation system developed at the University of Maryland [Andersen *et al.*, 1994]. Parka is a frame-based KR language which can be used to represent an ontology consisting of classes, subclasses, and individuals and properties of these. Parka is implemented in C, using relational databases to provide scaling to large knowledge bases. Par-KAP uses Parka's Application Programming Interface (API) but is itself implemented in Lisp, running on a Sparc workstation. In the remainder of this section, we describe how the frame-language is used to represent the planning KA library.

Task method decomposition structure This structure is represented as a tree of methods and tasks, linked by two types of relations, *executes* and *is-performed-by.* A task *is-performed-by* a method (possibly by more than one), and a method *executes* a task (as shown in Figure 3).

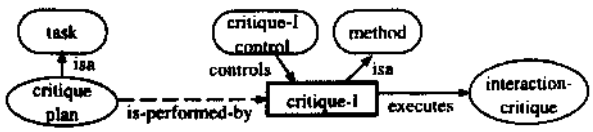

Figure 3: Representation of a task-method decomposition in Par-KAP.

In Par-KAP, we have also defined a number of methods corresponding to well known planners (like STRIPS, NONUN, SIPE, SNLP, UCPOP, etc.) which correspond to specific subtrees in the task-method decomposition described previously. These methods are directly linked to their primitive-tasks and respective primitive-methods. An example is shown in Figure 4. These specialized methods are used during the KA process as described in Section 4 below.

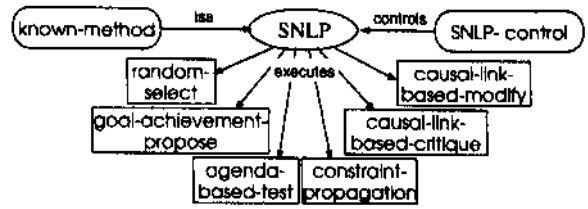

Figure 4: The SNLP method in Par-KAP.

Knowledge roles and domain requirements. In Par-KAP, knowledge roles are represented as an isa link (class/subclass) hierarchy. Individual domain models are represented as instances of the general class *domain model,* and these have plays links to the knowledge roles as was shown previously in Figure 1.

As discussed above, during KA the methods and knowledge roles are associated to each other by suitability criteria. Suitability criteria are also used to link control knowledge with methods. Figure 5 illustrates how Par-KAP represents suitability criteria, using the example of recognizing that the decomposition-propose method can only be used if the world description is in the form of Hierarchical Task Networks.

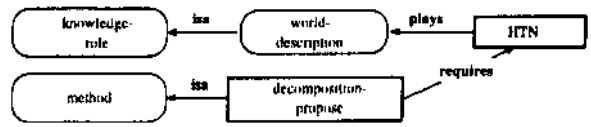

Figure 5: A suitability criterion for the *decomposition-propose* method.

Control-structure. Representing the control relationships among the tasks is done by linking methods to control methods via the property *controls.* Control methods are defined in terms of steps, which are related to the (sub)tasks which are executed by the method. The control method provides ordering and algorithmic relations between these steps, using pointers to various control-specific terms such as backtracking-point, exit-point, etc. Thus, the control information for the propose-critique-modify method shown in Section 2.3 can be represented as shown in Figure 6. Methods can be associated with more than one control regime, and as we have mentioned before, domain requirements must be used as a criterion to select between them.

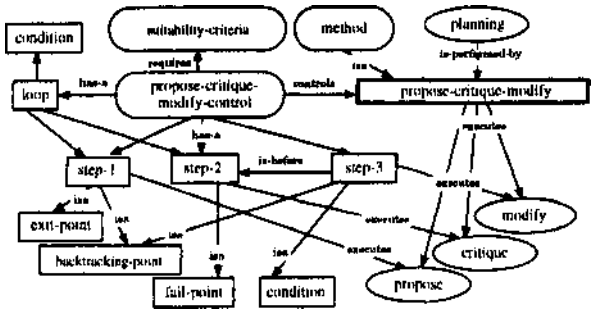

Figure 6: Control structure knowledge for the *propose-critique-modify* method.

```
(Task planning
   ((method propose-critique-modify
   ((task propose-refinement
      ((method propose-1
          ((task select-goal ((method random-select)))
           (task propose-expansion ((method decomposition-propose)))
           (task test-for-unachieved-goals
             ((method agenda-based-test))))))
   (task critique-plan
      ((method critique-1
          ((task consistency-critique ((method constraint-propagation)))
           (task interaction-critique ((method causal-link-based-critique)))))))
     (task modify-plan ((method causal-link-based-modify))))))))

CONTROL-STRUCTURE:
1. propose-refinement backtracking-point and exit-point
2.         select-goal backtracking-point
3.         propose-expansion backtracking-point
4.         test-for-unachieved-goals exit-point
5. critique-plan fail-point
6.         consistency-critique fail-point
7.         interaction-critique
8. modify-plan backtracking-point and cond: (conflict ≠ empty)
Control ordering. (5 is-before 8) and (2 is-before 3) (3 is-before 4)
```

Figure 7: Par-KAP Output

4 Knowledge acquisition

In this section, we provide two examples of how Par-KAP supports knowledge acquisition in planning. The first example is to find a list of the domain requirements necessary for a given planning method to be applied. The second is to find a set of possible planning methods, given a domain specification.

Domain requirements. One use of Model-based Knowledge Acquisition tools is to find a set of domain requirements imposed by a particular problem solving methodology. For planning, Par-KAP can thus assist a knowledge engineer in identifying restrictions on the domain knowledge if a particular planning strategy is used. This could be used by a knowledge engineer to help build a planning knowledge base. Par-KAP assists the user by specifying what type of knowledge has to be acquired and how it should be represented.

The input to Par-KAP for this type of KA request is a planning strategy, which can be a well known strategy or a new one given by the user (defined in terms of a set of basic planning methods plus a control structure). Par-KAP returns a list of domain requirements associated with each method involved in the input strategy, and also the requirements associated with the specified control structure.

A simple example of this use of Par-KAP is to input a known planning method. Par-KAP produces the information about how the plan is represented, assessed and composed. Thus, if the input is the known planner SNLP, the output is simply

```
state-changes = STRIPS-operators
plan-assessment-knowledge = causal-link-protection
plan-composition = partial-order
```

In a more complex example, the system could also output information about state descriptions, how state changes are tracked, and specific information about control regimes.

Planning strategy. Another use of such a KA tool is to help identify a problem solving strategy given a set of known domain features (expressed as domain modeling features as discussed in Section 2). In order to construct a planning strategy, Par-KAP takes as input a specification from the knowledge engineer of what sorts of features can be elicited from domain experts, literature, etc. Par-KAP then outputs either a known algorithm (if it covers all the features) or a set of techniques that would need to be combined to process this sort of domain.

To do this, Par-KAP uses the task-method decomposition structure shown previously in Figure 2. The system finds those tasks which are achieved by a method that could be applicable given the domain specification. A method is considered applicable if it meets two conditions. First, all of its domain requirements (if any) must be satisfied within the given domain specification. Second, for each of its subtasks, there exists at least one method which is applicable to perform that (sub)task. This essentially defines a recursion which bottoms out when it finds primitive methods matching the domain specification.

After generating a list of tasks and the corresponding primitive-methods, Par-KAP has to select a suitable control regime for the application. As described in Section 2.3 in Par-KAP, control knowledge can be fully-specified or partially-specified.

A fully-specified control regime may be selected (i) when the list of selected primitive-methods match with the primitive-methods of a well known planner (as in Figure 4) and (ii) when all that planner's domain requirements are satisfied by the domain specification. When none of the fully-specified control regimes can be selected, Par-KAP suggests the partially-specified control regime associated to each of the applicable non-primitive methods. (A better result would be to allow the interleaving or other complex composition of the control techniques suggested by Par-KAP. This sort of algorithm composition for planning is a topic of future research.)

As an example, suppose we give Par-KAP the following domain specification:

```
state changes = HTN
plan-assessment-knowledge ■ causal-link-protection
plan-composition = partial-order
state-change-data = resources
state-description - logical-predicates
```

The system returns a list of how the high-level tasks can be decomposed into the primitive tasks shown in Figure 7. This figure also gives an example of one of the control structures that is returned for these inputs.

5 Conclusion

We have presented a knowledge-acquisition library with components, designed for constructing planning systems. The library is comprised of the essential ingredients needed for giving concrete support when building a planner: problem-solving methods including control knowledge and a characterization of the domain knowledge used in planning. Suitability criteria, that form the connection between planning methods and both domain knowledge and control structure, are also included. A prototype KA tool for planning systems, Par-KAP has been implemented using this library.

Par-KAP shows that such a framework can provide concrete support to knowledge engineers building planning systems. In particular, Par-KAP gives two kinds of support: (1) given some planning strategy, support the (knowledge acquisition) process of building the knowledge-base to which to apply the problem-solving strategy; (2) given a characterization of a domain, generate a planning system strategy suitable for that domain.

One feature of Par-KAP is that the planning framework is represented using Parka, an efficient, frame-based AI language. This allows easy inspection and maintenance of the knowledge. In future versions of Par-KAP, we will introduce a user interface to the planning library, in order to allow the continuous update and refinement of the planning knowledge. We are also working to make the Par-KAP tool available over the Internet to allow it to be remotely accessed and used.

References

[Andersen *et al,* 1994] W. Andersen, J. Hendler, M. Evett, and B. Ketter. Massive parallel matching of knowledge structures. In H. Kitano and J. Hendler, editors, *Massively Parallel Artificial Intelligence.* AAAI/M1T Press, 1994.

[Benjamins *et al,* 1996] V. Richard Benjamins, Leliane Nunes de Barros, and Andre Valente. Constructing planners through problem-solving methods. In *Knowledge Acquisition Workshop - KAW'96 (Banff),* 1996.

[Benjamins, 1995] V. R. Benjamins. Problem-solving methods for diagnosis and their role in knowledge acquisition. *International Journal of Expert Systems: Research and Applications,* 8(2):93-120, 1995."

[Breuker and van de Velde, 1994] J. Breuker and W. van de Velde, editors. *CommonKADS Library for Expertise Modeling.* IOS Press, Amsterdam, The Netherlands, 1994.

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

[Chapman, 1987] D. Chapman. Planning for conjunctive goals. *AI,* 32:333-377, 1987.

[Chien, 1996] Steve A. Chien. Knowledge acquisition, validation and maintenance in a planning system for automated image processing . In *KAW'96. Banff,* 1996.

[Cottam and Shadbolt, 1996] Hugh Cottam and Nigel Shadbolt. Knowledge acquisition for search and rescue. In *KAW96. Banff,* 1996.

[de Barros *el al.,* 1996] L. Nunes de Barros, A. Valente, and V. R. Benjamins. Modeling planning tasks. In *AIPS-96,* pages 11-18, 1996.

[Hendler and McDermott, forthcoming] J. Hendler and D. JVIcDermott. *AI Planning Systems, chapters circulated for review.* Morgan Kaufmann, forthcoming.

[McAllester and Rosenblitt, 1991] D. McAllester and D. Rosenblitt,. Systematic nonlinear planning. In *Proc. of AAAI-91* pages 634-639, Anaheim, CA, 1991.

[McCarthy and Hayes, 1969] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4,* 1969.

[Orsvarn, 1996] K. Orsvarn. Principles for libraries of task decomposition methods - conclusions from a case-study. In N. Shadbolt, K. O'Hara, and G. Schreiber, editors, *Lecture Notes in Artificial Intelligence, 1076, EKAW-96,* pages 4865. Springer-Verlag, 1996.

[Schreiber *et al,* 1994] A. Th. Schreiber, B..J. Wielinga, R. de Hoog, J. M. Akkermans, and W. Van de Velde. CommonKADS: A comprehensive methodology for KBS development. *IEEE Expert,* 9(6):28 37, December 1994.

[Tate *et al,* 1990] A. Tate, J. Hendler, and M. Druminond. Readings in planning, 1990.

[Tate, 1996] A. Tate, editor. *Advanced planning technology - technological achievements of the ARPA/Rome Planning Initiative.* AAAI Press, 1996.

[Tu and Musen, 1996] Samson W. Tu and Mark A. Musen. Episodic refinment of episodic skeletal plan refinment, In *KAW'96. Banff,* 1996.

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