

# Extending the Role-Limiting Approach: Supporting End Users to Acquire Problem-Solving Knowledge

Jim Blythe<sup>1</sup>

**Abstract.** Role-limiting approaches have been successful for acquiring knowledge from domain experts. However most systems using this approach ask the user a pre-determined sequence of questions and are therefore not flexible enough for a wide range of situations. They also typically do not support acquiring problem-solving knowledge, but only instance and type information. We extend the role-limiting approach with a knowledge acquisition tool that dynamically generates questions for the user based on a background theory of the domain and on the user's previous answers. The tool uses KA scripts to give an overall structure to a session. We show how to augment a background theory based on problem-solving methods to support this style of interaction. We present results from tool ablation experiments with domain experts in two different domains based on a problem solving method for plan evaluation. When the tool was used, tasks were completed in substantially less time than when the ablated version was used. In addition participants were able to complete substantially more tasks on average, twice as many in one of the studies.

## 1 INTRODUCTION

In order to be successful, deployed intelligent systems need to be robust to changes in their task specification. They should allow users to make modifications to the system to control how tasks are performed and even to specify new tasks within the general capabilities of the system. A common direction of work in knowledge acquisition (KA) aims to support this ability through explicit domain-independent theories of the problem-solving process, often called problem solving methods (PSMs) [3, 5]. Such theories can encourage re-use across different applications and the structured development of intelligent systems as well as providing a guide for knowledge acquisition from experts.

Our interest is in using background theories to guide automatic knowledge acquisition from domain experts who are not necessarily programmers. Background theories of the problem-solving process are appealing for this purpose because their model of the overall structure of the process can in turn be used to structure the knowledge acquisition session for the user and provide context for the knowledge that is acquired. However, most KA approaches that use problem solving methods typically focus on assisting knowledge engineers rather than domain experts [6, 7].

Other KA tools such as SALT [10] follow a role-limiting approach that allows domain experts to provide domain-specific knowledge that fills certain roles within a PSM. However, most of these have

been used to acquire instance-type information only. Musen [11] argues that although role-limiting strategies provide strong guidance for KA, they lack the flexibility needed for KBS construction. The problem-solving structure of an application cannot always be defined in domain-independent terms, and one problem-solving strategy may not address all the particulars of an application because it was designed to be general. Musen and others advocate using finer-grained PSMs from which a KBS can be constructed [13]. Gil and Melz [9] address this issue by encoding the PSM in a language that allows any part of the problem-solving knowledge to be inspected and changed by a user. In their approach, a partially completed KBS can be analysed to find missing problem-solving knowledge that forms the roles to be filled. This work extended the role-limiting approach to acquire problem-solving knowledge and to determine the roles dynamically.

However, Gil and Melz's approach is still not adequate to allow an end user to directly add problem-solving knowledge. There are at least two reasons for this. First, there is no support for a structured interaction with the user provided by tools like SALT. The knowledge roles, once generated, form an unstructured list of items to be added, and it can be difficult for the user to see where each missing piece of knowledge should fit into the new KBS. Second, the user must work directly with the syntax of their language to add problem-solving knowledge, which is not appropriate for end users.

In this paper we present an approach that builds on Gil and Melz's work to allow an end user to add problem-solving knowledge to a KBS with a dynamic role-limiting approach. In our approach, the tool structures an overall KA session using KA scripts [16]. While following the scripts, background knowledge is used to dynamically generate questions for the user in response to the user's answers to previous questions. As we describe in the next section, this background knowledge includes ontologies that describe the problem-solving knowledge to be added in terms that make sense for the user. Other roles are generated, as before, from an analysis of a partially completed KBS. The background knowledge is also augmented to support interaction with the user through constrained English sentences. We describe an implemented tool, called PSMTTool, that uses this approach and show empirically that end users without programming experience are able to add problem solving knowledge to a KBS with this system. The main contributions of this paper are (1) showing how to use background knowledge to generate a structured dialog with an end user to acquire problem-solving knowledge and (2) demonstrating an implemented KA tool that successfully uses the approach.

We report on two sets of experiments that tested PSMTTool. Experts in two different domains were asked to add new evaluation criteria to plan evaluation systems. The KA tasks required providing a combination of problem-solving and ontological knowledge that

---

<sup>1</sup> University of Southern California / Information Sciences Institute, Marina del Rey, CA 90292, USA, email: blythe@isi.edu

was integrated with the existing system. In both domains, experts were able to complete substantially more tasks using the KA tool and performed the tasks more quickly.

In the next section we describe a background theory for plan evaluation that we have used in this work. Then we describe how the KA tool uses the theory to generate questions for the user, with examples from a system that acquires new knowledge to evaluate travel plans. We then present results from our experiments and survey related work. Finally we discuss how the results might extend to other background theories of problem solving and mention future work.

## 2 A BACKGROUND THEORY OF PLAN EVALUATION

In this section we briefly describe the background theory for plan evaluation that was used in our experiments and is used in this paper to illustrate PSMTool. The tool itself is designed to work independently of a particular theory, and we begin by describing necessary features of a background theory in order for this approach to be successful.

### 2.1 How a background theory supports PSMTool

We use a collection of fine-grained PSMs as the basis for the background knowledge in PSMTool, and add to it the knowledge required for the tool to take over some of the tasks of the knowledge engineer. When using a collection of PSMs to build a knowledge-based system, the knowledge engineer typically selects a component from the collection that is best suited to the task and adapts it to address the task at hand [3]. We support this activity with two additions to the background theory of the task. First, we provide an ontology for the elements of the collection that distinguishes them based on features of the tasks they perform rather than their procedural structure. The ontology has a hierarchical structure with a number of orthogonal partitions that can be used to generate a dialog with the user dynamically as we describe below. Second, we provide guidance for adapting each element in the collection. Each element is a generic PSM for a given task, and is implemented as a collection of EXPECT methods [8]. Within each element, we identify the methods that are designed to be changed when the element is specialised. This information is used by PSMTool to further guide the KA process once a PSM element has been chosen. We illustrate these points in the context of a background theory for plan evaluation implemented in the EXPECT system.

EXPECT [8] is an architecture for knowledge acquisition that makes use of an explicit representation for both factual and problem-solving knowledge. It analyses an intelligent system defined in its language to provide KA support to the user, for example by identifying missing problem-solving knowledge or highlighting the information that is needed about objects in the knowledge base. EXPECT's explicit representation makes it ideally suited for representing background theories of problem solving that provide direct support for knowledge acquisition. Previous work has shown that background theories such as propose-and-revise can be represented in EXPECT to take advantage of its analytical capabilities and provide the benefits of both role-limiting KA strategies and completeness analysis [9].

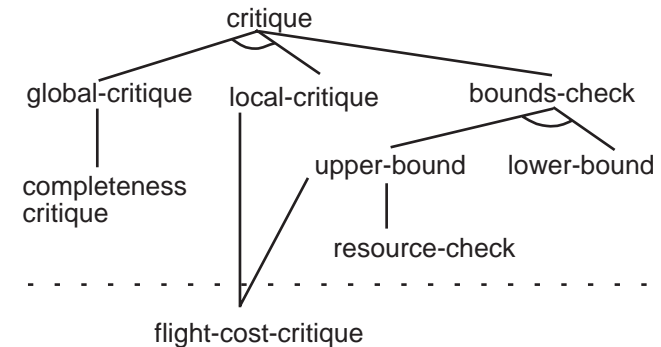
### 2.2 Plan evaluation

Plan evaluation problems belong to a domain-independent problem class in which an agent, typically a human expert, judges alternative plans according to a number of criteria. The aim is usually to see

which of the alternative plans is most suited for some task. In terms of a standard problem solving method library such as commonKADS [3], it is a special case of assessment.

Each different criterion along which the plan is judged is represented explicitly as a critique in our framework. Through experience with several intelligent systems for plan evaluation [18], Blythe and Gil have identified several patterns in the ways that critiques are evaluated [1]. These patterns of critiques indicate regularities that can be re-used across planning domains and provide guidance for knowledge acquisition.

In our background theory for plan evaluation this knowledge is represented through an ontology of critique types, partially shown in figure 1. Some groups of subtypes (shown with arcs connecting their links) partition their parents, while some just specialise the class. For example, the subtypes `local-critique` and `global-critique` form a partition, so any critique must belong to exactly one of these classes. `local-critique` defines critiques that are checked by performing some analysis on some or all of the steps in the plan on an individual basis, while `global-critique` defines critiques that have a global definition on the plan, for example the plan's total fuel consumption.



**Figure 1.** Different types of critiques in the background theory of plan evaluation. Each subclass is identified with a recurring pattern for evaluating a critique, implemented through EXPECT methods attached to the class.

In addition to the ontology of critiques, the background theory uses an ontology of planning terms and a specialised ontology of resources based on Ozone [15]. More details about these can be found in [1].

The critiques are defined operationally through EXPECT methods. For example, the method in Figure 2 says that a step satisfies an upper bound (a kind of critique) if and only if the actual value of the property to be measured by the critique is less than or equal to some maximum value. The tasks of estimating the actual and maximum values for the property respectively form two subgoals that are addressed by other methods in the theory.

The background theory can be used to create a working plan evaluation system for a particular domain by defining domain-specific critique classes within the ontology and adding the necessary EXPECT methods to complete each critique's definition. We illustrate this in a travel planning domain. In this domain, plans are itineraries for travel and the steps in plans represent reservations of flights, hotels and rental cars. The plan critiquer for the travel planning domain is implemented as an extension of the background theory for plan critiquing. For example, one possible critique in the domain checks that no flight costs more than a set amount, say \$500. This is implemented by defining the class `flight-cost-critique` as

```

(defmethod evaluate-local-upper-bound
  "To determine whether a step satisfies an upper
  bound, check that the actual value of the property
  to measure for the upper-bound is less than or equal
  to the maximum allowed value of the property."
  :capability
  (determine-whether
   (obj (?task is (inst-of step)))
   (satisfies (?bc is (inst-of upper-bound))))
  :result-type (inst-of boolean))
:method
  (is-less-or-equal
   (obj (estimate (obj actual-value)
                  (of (property-to-measure ?bc))
                  (for ?task)))
   (than (estimate (obj maximum-allowed-value)
                   (of (property-to-measure ?bc))
                   (for ?task))))

```

**Figure 2.** An EXPECT method.

a subclass of both `local-critique` and `upper-bound`. The value of the relation `property-to-check` is required for any subclass of `upper-bound`, and here it takes the value `flight-cost`. The EXPECT methods for those classes are then used to evaluate the new critique, resulting in a check that the actual amount of `flight-cost` for each step is less than or equal to the maximum amount. The developer completes the definition by defining methods to compute the actual amount (the cost of the flight) and the maximum amount (\$500).

### 3 INTERACTING WITH PSMTTOOL

We have implemented a tool, called PSMTTool, that makes use of a background theory for problem-solving to guide a user in extending the capabilities of an intelligent system. PSMTTool follows a script to organise its interaction with the user that is hand-written for the background theory. However, the questions that the tool asks the user while it executes the script are automatically derived from the theory, as we show below. Before describing this we show an example of the tool's interaction with a user.

The KA tool begins with a window split in two screens, showing a plan on the left and on the right a list of evaluations of the plan made with the critiques currently in the knowledge base. The user can choose to add a new critique, in which case a second window appears that will keep track of the interactions with the user made in defining the critique, shown in Figure 3. The interaction follows a script that has three steps: first, it adds a new critique class and classifies it in the ontology, then it generates a list of EXPECT methods for the user to edit that will be used in evaluating the critique. Once these steps are complete the new critique has been defined. The third step then applies the critique to the plan in the first window and the result is shown for the user to verify. We now show how the questions used in the first two steps of the script are dynamically created.

### 4 SUPPORTING KNOWLEDGE ACQUISITION WITH BACKGROUND THEORIES

We are interested in using a background theory to provide support for users who are not necessarily programmers to extend and tailor systems for their domains. We added the following kinds of information to the background theory described above.

The critique ontology, which records functional differences in types of critique based on their patterns of usage, can be used to generate questions to get information from the user about a new critique without requiring programming knowledge. To do this, each class in the ontology is augmented with a text description that characterises the class and, in the case of partitions, distinguishes the class from other elements in the partition.

In many cases, once the user has provided information so the new critique is classified, much of the problem-solving knowledge needed to implement the critique is available through the critique's parents in the background theory. This problem-solving knowledge can handle overall organization of the critique and its integration with the system, which is sometimes the most complex part of the critique's procedural definition.

The background theory also distinguishes some EXPECT methods as default methods that may be specialised when a new critique is defined. This serves two purposes. While all the methods in the background theory can in principle be specialised, the distinguished subset can give the KA tool a road-map for tailoring the theory for a new critique. In addition, the body of the general method can provide a starting-point for the specialised method and the general method can be included for this reason, even if it is not intended to be used without specialisation.

#### 4.1 CLASSIFYING THE CRITIQUE FROM THE ONTOLOGY

The questions asked in the first part of the critique window are automatically derived from the ontology of critiques in the theory and classify the new critique according to the the ontology. Once the classification process is complete, a new critique class is defined for the new critique as a subclass of the identified classes. This class is able to inherit generic critiquing knowledge from its ancestors as described in the previous section.

The algorithm for determining questions in the critique script adds each question for one of three reasons: 1) to determine which element of a partition the critique belongs to, 2) to determine whether the critique is a subclass of some class that is not part of a partition and 3) to find the value of a required field of a class. The algorithm traverses the critiques in depth-first order to help maintain a focus of attention for the user in answering the critiques. Figure 4 shows the algorithm that generates the questions in the critique window.

Figure 3 shows the questions that are generated using this algorithm for the class `flight-cost-critique`, defined as shown in Figure 1. The text for each question is stored with each class or required field in the ontology. Although the algorithm requires the developer to store canned text with the classes and fields in the ontology, this is considerably easier than generating scripts of questions for the different possible classifications of a critique. The answers to the five questions in steps one through five have allowed the KA tool to classify the new critique class as a subclass of both `upper-bound-check` and `local-constraint`.

#### 4.2 CHOOSING AND CREATING NEW METHODS TO EDIT

Once the new critique has been classified in the ontology, the system builds a problem-solving tree in EXPECT for the generic goal to determine whether a plan satisfies the critique. EXPECT is able to detect subgoals for which no method can be found and the information that

**Critique Wizard**

**Quit**

**Part 1. Answer some questions about the critique**

Done	1) Critique name	Please give a name to this critique.	flight-cost-critique
Done	2) Number based	Does the critique reason about numbers? (e.g amount of fuel or distance, rather than whether some item exists)	yes
Done	3) Checks a resource	Does the critique check for a sufficient amount of something to be present? (e.g checking the amount of fuel is sufficient, rather than checking that a distance is too great)	no
Done	4) Quantity	Please give a name to the property to be checked.	flight-cost
Done	5) Once or per task	Is the check to be made once for the whole plan or individually on certain steps?	Individually on certain steps

**Part 2. Define some methods for the critique**

I will evaluate the critique by evaluating each relevant task in turn.  
 I will evaluate each task by estimating the actual amount of the property for the task and the maximum allowed amount for the task, and by checking that the actual amount is not greater than the maximum amount.  
 Please edit the following methods to choose the relevant steps, estimate the actual amount and the maximum allowed amount.

Done	6) edit a method to find the steps in a trip for which to test a flight cost critique
Done	7) edit a method to estimate the maximum allowed value of a flight cost for a step
Doing	8) edit a method to estimate the actual value of a flight cost for a step

**Part 3. Check the critique**

To do 9) Run the critiquer

**Figure 3.** The critique window asks the user questions in order to classify the new critique within the background ontology and then prompts the user to modify several new EXPECT methods to tailor the behaviour of the new critique.

```

Initialise C to the partitions and subclasses of critique
1  if C is empty, stop
2  set c to the first element in C, and remove it from C
3  if c is a partition of subclasses,
4    ask which subclass in c the new critique belongs to.
5    set n to the answer
6  else if c is a subclass,
7    ask whether the critique belongs to the subclass
8    if YES, set n to c
9    if NO, goto step 1.
10 for each required field r of n:
11  ask for the value of r.
12 push the partitions and subclasses of n onto C
13 goto step 2.

```

**Figure 4.** The algorithm that dynamically generates classification questions in the critique window.

is required of objects in the domain, such as plans and critiques. Requesting information that is missing in these categories is similar to the behaviour of role-limiting knowledge acquisition systems such as SALT [10], but more flexible since it is generated from an analysis

of an explicit model of the background theory. This connection is discussed in more detail in [9]. We augment this algorithm so that it also finds subgoals addressed by methods that are marked in the background theory as specialisable. The KA tool then prompts the user to enter methods for these subgoals one by one. If a specialisable method exists, this forms the initial version of the new method.

The user edits problem-solving methods via the English-based editor, which uses constrained English sentences to present and edit the methods rather than the formal language in which the methods are defined [2]. The method capability and body are presented as structured English, and when the user selects a word or group of words in them, the lower window of the editor shows alternatives that could be substituted for the selection and maintain the syntactic correctness of the method. The alternatives can include Loom or EXPECT expressions. This editor is important to the success of PSMTTool, since the intended users are not programmers and would not be comfortable with formal languages for ontologies or problem-solving knowledge.

In the case of the class `flight-cost-critique`, the problem solving tree makes use of the method in Figure 2, and the critique tool finds three specialisable methods, which are shown in the lower part of the critique window in Figure 3. Figure 5 shows the English editor being used to define one of the methods, to compute the cost of a flight. Once these three methods are defined, the new critique is

ready to be applied. In this case, they can be defined using just Loom expressions and constants, without calling other EXPECT methods, although the editor allows this.

## 5 EXPERIMENTAL RESULTS

Our aim is to design a knowledge acquisition tool that can be used by domain experts. To test the tool, we ran two sets of experiments that test the same hypotheses, but in different plan evaluation domains and with different sets of domain experts. The hypotheses we tested are 1) that with the knowledge acquisition scripting tool, domain experts are able to add more complex critiques than without the tool and 2) that domain experts are able to add the same critique more quickly using the tool. In each domain, we ran a tool ablation experiment, in which participants used two versions of a knowledge acquisition environment that were identical except that only one gave access to the KA tool. Both versions allowed users to add a critique class through a class browser and to start the English method editor to add a method.

Participants were asked to define new critiques for a plan evaluation system. In order to control the conditions of the experiment as far as possible, the critiques to be added using the tool were structurally isomorphic to the critiques to be added with the ablated version. In other words, corresponding critiques from the two groups critiques required the same pattern of EXPECT methods and loom retrievals in their definitions, although the names of the methods and loom relations could be different. In order to control for learning effects during the course of each experiment, half the participants worked first with the tool and then with the ablated version, and half worked the other way around.

The participants in the first experiment were four Army officers from the Army Battle Command Battle Labs in Fort Leavenworth. The problem domain for this experiment evaluated Army battle plans. The participants in the second experiment were three project assistants from our division and one business school professor, and the problem domain was the travel plan evaluation system that we have used to illustrate the paper. None of the participants from either experiment had written a computer program. It is unfortunate that the number of participants was too low to draw statistically valid conclusions from these experiments, but the experiments are costly to run since they require one experimenter per participant to make observations.

The critiques in each set were graded in difficulty, so that we could also investigate the effect that this had on the impact of the KA tool. The simpler critiques can be implemented with one EXPECT method, while the more complex critiques require iterating over a set of steps in the plan, or require two or three EXPECT methods to be specialised if they are defined using the background ontology.

As an example of a simple critique, participants in the second study were asked to define a check that the plan included a hotel reservation. As an example of a more complex critique, participants in the first study were asked to define a check that the force ratio, defined as the ratio of friendly fire-power to enemy fire-power, is adequate for each task in the plan. The level that is considered adequate must be computed based on the task.

Figure 6 shows the number of KA tasks that were completed by the subjects with the KA tool and with the ablated version in each of the domains. In this figure, each critique has been split into a number of component tasks, corresponding to adding a new critique class or completing either the capability or body of a method. We have aggregated the tasks from the two different sets given to the

participants. As the graph shows, use of the tool does not affect the number of tasks that are completed for the easiest critique, but there is a marked difference for more complex critiques. The same is true if we consider the number of whole critiques added rather than the component tasks. Participants in the battle planning experiment were able to complete more than twice the tasks with the tool than were completed with the ablated version. Participants in the travel planning experiment showed smaller increases but still significant ones.

Figure 7 shows the average times taken to define a critique in each domain. To ensure that the times are comparable between the standard and ablated versions of the system, only participants who completed both tasks were used to compute the averages. The point representing the most complex critique attempted without the KA tool is missing from both experiments because no participant successfully added the critique.

The results from these experiments support our hypotheses. Participants complete significantly more KA tasks using the tool, and take considerably less time to do it.

One interesting difference between the participants in the two experiments was the amount of training that they received for the task. The battle planning experiments were part of a number of experiments made at Fort Leavenworth with different KA systems and the participants received eight hours of training before using the tool — four hours of training with EXPECT and four hours with a related KA tool. In the travel planning experiments, participants received only one hour of training with the tool and none with EXPECT. However the number of tasks completed and the average times are very similar between the two groups. This indicates that domain experts might require very little training to be able to use the tool. We have not yet studied this systematically, but can offer one more anecdotal piece of evidence: at Fort Leavenworth, a fifth participant was asked to add the same critiques using the KA tool after less than one hour's training. The participant succeeded, and took time comparable with the participants who had received eight hours of training.

## 6 RELATED WORK

Other knowledge acquisition tools help users add domain-specific information to a knowledge base [10, 5]. Some, such as ASTREE [14] use explicit ontologies as we do to organise problem-solving methods. Like our tool, these tools are built for domain experts and not knowledge engineers, but they do not allow users to provide domain-specific problem-solving knowledge as PSMTool does.

Other researchers have developed tools to support the acquisition of planning knowledge [4, 12], but these systems are built for knowledge engineers. Both of them address some aspects of the acquisition of plan evaluation knowledge, but their central focus is plan generation. Valente and his colleagues have also studied problem-solving methods related to planning [17]. Our model has similarities to theirs but is more aimed at plan evaluation, a separate activity from plan creation that has been less studied from a computational viewpoint. Another difference in our approach is the automatic support for KA via EXPECT and script generation.

## 7 DISCUSSION

We described an extension to the role-limiting approach for knowledge acquisition from end users that dynamically generates knowledge roles to be filled and can be used to acquire problem-solving knowledge. Our approach combines a collection of problem-solving methods with supporting ontologies that can be used to guide the

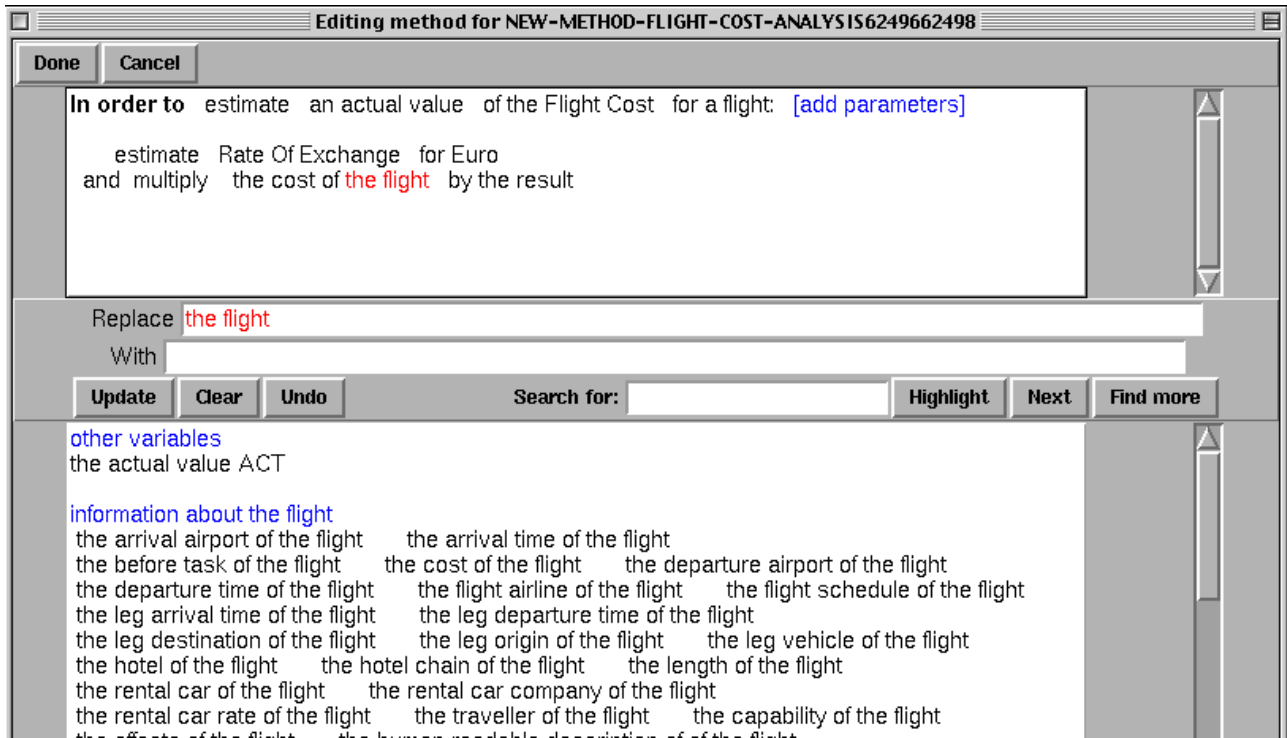


Figure 5. The English-based method editor in use to change the method to compute the actual cost of a flight.

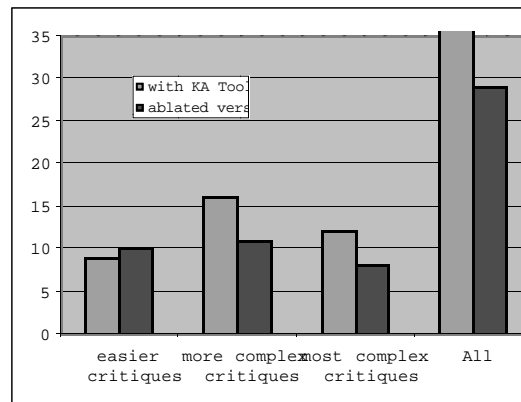
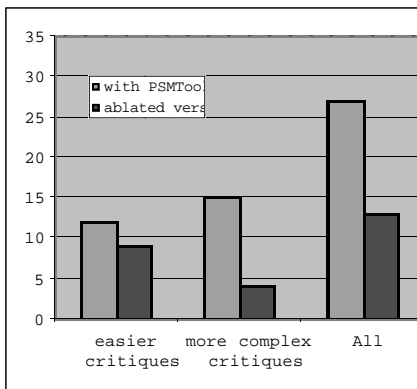


Figure 6. Tasks completed with the full tool and with the ablated version. Results in the battle planning domain are shown on the left and results in the travel planning domain are shown on the right.

activities of method selection and specialisation. As well as demonstrating the approach for a plan evaluation problem solving method, we have shown through controlled experiments that domain experts can provide more complex knowledge to an intelligent system using PSMTTool and that they take less time to provide the same knowledge.

Although we have discussed these techniques in the context of the plan evaluation theory they are of course not limited to this background theory. The ways in which we augment the theory to support knowledge acquisition would be useful for a range of theories, and our algorithms for generating questions to ask a user are domain-independent. The script that we demonstrated, however, was specific to the plan evaluation theory. Other background theories would certainly need their own scripts and will probably need a greater variety

of ways to support knowledge acquisition. Future work includes generalizing this result to other classes of task. We plan to investigate theories to help acquire process control knowledge and search control knowledge for AI planning systems, among others.

## Acknowledgments

I am very grateful to present and past members of the EXPECT project who have contributed many good ideas as well as constructive criticism to the work described here, especially Yolanda Gil, Andre Valente, Surya Ramachandran, Jihie Kim, Marcelo Tallis and Bill Swartout. I gratefully acknowledge the support from DARPA under contracts DABT63-95-C-0059 as part of the DARPA/Rome Lab-

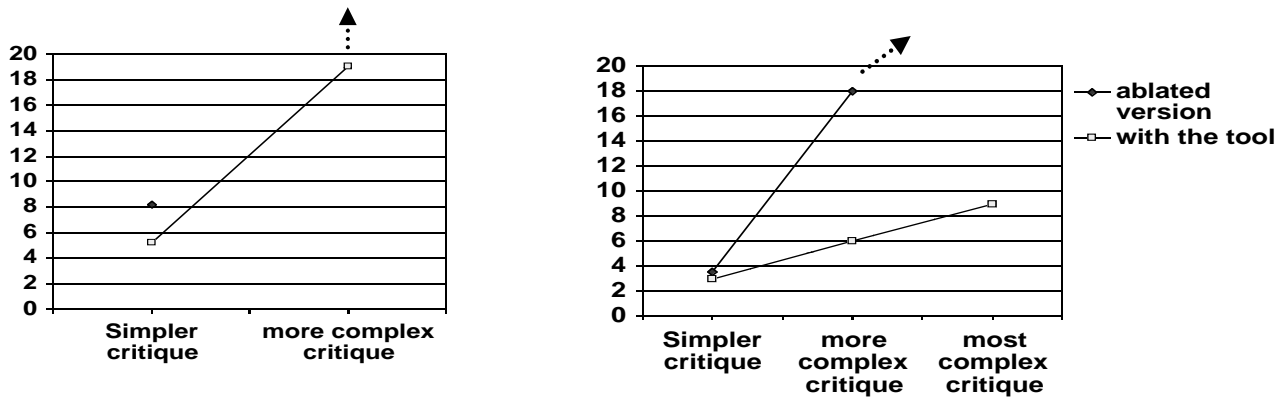


Figure 7. Average times (in minutes) taken to define each critique. Results from the battle planning domain are shown on the left and results from the travel planning domain are shown on the right.

ratory Planning Initiative, F30602-97-1-0195 from the High Performance Knowledge Bases (HPKB) program, F30602-97-C-0118 from the Joint Forces Air Component Commander (JFACC) program, and F30602-97-C-0068 from the Control of Agent Based Systems (CoABS) program.

## REFERENCES

- [1] Jim Blythe and Yolanda Gil, 'A problem-solving method for plan evaluation and critiquing', in *Proc. Twelfth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta, (1999).
- [2] Jim Blythe and Surya Ramachandran, 'Knowledge acquisition using and english-based method editor', in *Proc. Twelfth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta, (1999).
- [3] Joost Breuker and Walter Van de Velde, *CommonKADS Library for Expertise Modelling: Reusable Problem Solving Components*, IOS Press, 1994.
- [4] Leliane de Barros, Jim Hendler, and V. Richard Benjamins, 'A knowledge acquisition tool for building practical planning systems', in *Proc. 15th International Joint Conference on Artificial Intelligence*, Nagoya, Japan, (August 1997). Morgan Kaufmann.
- [5] Henrik Eriksson, Yuval Shahar, Samson W. Tu, Angel R. Puerta, and Mark A. Musen, 'Task modeling with reusable problem-solving methods', *Artificial Intelligence*, **79**, 293–326, (1995).
- [6] Dieter Fensel and V. Richard Benjamins, 'Key issues for automated problem-solving methods reuse', in *Proc. the European Conference on Artificial Intelligence*, (1998).
- [7] Dieter Fensel and Enrico Motta, 'Structure development of problem solving methods', in *Proc. Eleventh Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta, (1998).
- [8] Yolanda Gil, 'Knowledge refinement in a reflective architecture', in *Proc. Twelfth National Conference on Artificial Intelligence*. AAAI Press, (1994).
- [9] Yolanda Gil and Eric Melz, 'Explicit representations of problem-solving strategies to support knowledge acquisition', in *Proc. Thirteenth National Conference on Artificial Intelligence*. AAAI Press, (1996).
- [10] Sandra Marcus and John McDermott, 'Salt: A knowledge acquisition language for propose-and-revise systems', *Artificial Intelligence*, **39**, 1–37, (1989).
- [11] Mark A. Musen, 'Overcoming the limitations of role-limiting methods', *Knowledge Acquisition*, **4**(2), 165–170, (1992).
- [12] Karen Myers, 'Advisable planning systems', in *Advanced Planning Technology*, ed., Austin Tate, Edinburgh, UK, (1996).
- [13] A. R. Puerta, J. W. Egar, S. Tu, and M. A. Musen, 'A multiple-method knowledge acquisition shell for the automatic generation of knowledge acquisition tools', *Knowledge Acquisition*, **4**(2), 171–196, (1992).
- [14] Chantal Reynaud and Francoise Tort, 'Using explicit ontologies to create problem solving methods', *International Journal of Human-Computer Studies*, **46**, 339–364, (1997).
- [15] Steven F. Smith and Marcel Becker, 'An ontology for constructing scheduling systems', in *AAAI Spring Symposium on Ontological Engineering*, Stanford University, (March 1997).
- [16] Marcello Tallis and Yolanda Gil, 'Designing scripts to guide users in modifying knowledge-based systems', in *Proc. Sixteenth National Conference on Artificial Intelligence*. AAAI Press, (1999).
- [17] Andre Valente, Richard Benjamins, and L. Nunes de Barros, 'A library of system-derived problem-solving methods for planning', *International Journal of Human-Computer Studies*, **48**, 417–447, (1998).
- [18] Andre Valente, Jim Blythe, Yolanda Gil, and William Swartout, 'On the role of humans in enterprise control systems: the experience of inspect', in *Proc. of the JFACC Symposium on Advances in Enterprise Control*, San Diego, CA, (November 1999).