To appear, International Joint Conference on Artificial Intelligence, 2001

Integrating Expectations from Different Sources to Help End Users Acquire Procedural Knowledge

Jim Blythe
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292, USA
blythe@isi.edu

Abstract

Role-limiting approaches using explicit theories of problem-solving have been successful for acquiring knowledge from domain experts[1]. However most systems using this approach do not support acquiring procedural knowledge, only instance and type information. Approaches using interdependencies among different pieces of knowledge have been successful for acquiring procedural knowledge, but these approaches usually do not provide all the support that domain experts require. We show how the two approaches can be combined in such a way that each benefits from information provided by the other. We extend the role-limiting approach with a knowledge acquisition tool that dynamically generates questions for the user based on the problem solving method. This allows a more flexible interaction pattern. When users add knowledge, this tool generates expectations for the procedural knowledge that is to be added. When these procedures are refined, new expectations are created from interdependency models that in turn refine the information used by the system. The implemented KA tool provides broader support than previously implemented systems. Preliminary evaluations in a travel planning domain show that users who are not programmers can, with little training, specify executable procedural knowledge to customize an intelligent system.

1 Introduction

In order to be successful, deployed intelligent systems must be able to cope with changes in their task specification. They should allow users to make modifications to the system to control how tasks are performed and to specify new tasks within the general capabilities of the system. For example, consider a travel planning assistant that can locate flights and hotel reservations to help a user create an itinerary for some trip. Many such systems allow users to search for hotels by cost, hotel chain and distance to some location. However, users often have individual requirements, such as "prefer a direct flight unless it is double the price of the cheapest connecting flight" or "if the flight arrives late in the evening, the hotel should be near the airport, otherwise it should be near the meeting." These requirements often go beyond the initial abilities of the travel assistant tool, leaving the user to check them by hand and severely limiting the tool's usefulness.

The ability to define requirements like these in a way that can be integrated with the tool is therefore essential for it to meet a wide range of users' needs. The requirements, which were suggested by an independent user, are typical in that they do not require adding new sources of information to the tool, which can already compute distances and knows flight arrival times. Instead, they require processing the information to produce new criteria: in the first case for instance, finding the minimum value of the price for the set of all connecting flights and multiplying this value by 2. These are instances of *procedural* knowledge, rather than purely *factual* knowledge like the distances or costs. A tool that can incorporate new procedural knowledge like this from users can have a range of applicability that goes well beyond that originally envisaged by the developer.

However, it is difficult for users who are not programmers to add procedural knowledge to systems. In the next section we discuss some of the challenges that users face in more detail. Some KA approaches use *expectations* of the entered knowledge to aid users [Kim & Gil, 1999]. Expectations are beliefs about the knowledge that the user is currently entering that can be used to constrain the possibilities for what can be entered next. For example, expectations can govern the return type of a function that the user is entering, or its general purpose within the system. They can be used both to interpret and check knowledge as it is entered, to provide feedback or to help a user enter correct knowledge.

Another common direction of work in knowledge acquisition (KA) aims to support users through explicit, domain-independent theories of the tool's problem-solving process, often called *problem-solving methods* (PSMs) [Breuker & de Velde, 1994; Eriksson *et al.*, 1995]. These 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. They can be

[1]I gratefully acknowledge support from DARPA grants F30602-00-2-0513, as part of the Active Templates program, and F30602-97-1-0195, as part of the High Performance Knowledge Bases program

used to structure the KA session for the user and provide context for the knowledge that is acquired. However, most KA approaches that use problem-solving methods focus on assisting knowledge engineers rather than domain experts [Fensel & Benjamins, 1998; Fensel & Motta, 1998].

Other KA tools such as SALT [Marcus & McDermott, 1989] take a *role-limiting* approach, allowing 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 [Musen, 1992] argues that although role-limiting provides strong guidance for KA, it lacks the flexibility needed for constructing knowledge-based systems (KBS). The problem-solving structure of an application cannot always be defined in domain-independent terms, and a single problem-solving strategy may be too general to address the particulars of an application. Puerta et al. advocate using finer-grained PSMs from which a KBS can be constructed [Puerta *et al.*, 1992].

Gil and Melz [Gil & Melz, 1996] address this problem 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 analyzed to find missing problem-solving knowledge that forms the roles to be filled. This is done as part of the *interdependency analysis* performed by EXPECT [Swartout & Gil, 1995; Kim & Gil, 1999], which looks at how both problem-solving knowledge and factual knowledge is used in the intelligent system. This work extended the role-limiting approach to acquire problem-solving knowledge and to determine the roles dynamically. However, Gil and Melz's tools were not adequate for end users. There are at least two reasons for this. First, there is no support for a structured interaction with the user as there is in 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 their procedure syntax to add problem-solving knowledge, which is not appropriate for end users.

One solution is to exploit knowledge from a variety of sources to guide the user through all the stages of adding procedural knowledge. We view all the KA tools mentioned above as providing different kinds of expectations on the knowledge to be entered, either from background theories in the form of the PSMs, or from interdependency analysis. This framework allows the tool to exploit the background theory from the PSM to help a user begin the process of adding knowledge, and also to exploit interdependencies to help a user refine an initial definition of a procedure into an executable one. In our implemented system, which is built on EXPECT, modules that use expectations from the two sources share information in the form of input-output characterizations of expected procedural knowledge. This sharing is mutually beneficial to both the background theory-based and interdependency-based approaches.

In the next section I discuss some of the challenges that users who are not programmers face in defining procedural knowledge. Next I describe the use of expectations in more detail and show how they are integrated in an implemented tool, called *Constable*. I then report on initial user experiments with Constable that demonstrate the value of the approach.

2 Why do users find it difficult to enter procedural knowledge?

There are several challenges that users face in defining procedural knowledge. Here I sketch how some of them can be addressed, and highlight the role played by expectations.

Users do not know where to start. Adding a new capability to an intelligent system may require adding several related pieces of knowledge, in a form recognizable by the system. Simply beginning this process can be difficult even for an experienced programmer who does not know the system well. Expectations based on the **problem-solving method** can help identify the purpose and the initial structure of new procedural knowledge [Eriksson *et al.*, 1995].

Users do not know formal languages. A structured English editor allows users to modify English paraphrases of a procedure's formal representation [Blythe & Ramachandran, 1999]. Users can select a fragment of the paraphrase and choose from a list of suggested replacements for the fragment, which are automatically generated based on **expectations from method analysis**. This approach hides the internal procedure syntax while avoiding the challenge of full natural language processing.

Users may not know whether the added knowledge is free of syntax errors. The new knowledge may have errors in formal syntax (*e.g.* an "if" statement with no condition) or it may have type errors (*e.g.* trying to multiply the result of a subprocedure that returns a hotel). Since users create procedures by choosing replacements from a list, the editor can effectively eliminate some syntax errors. Others can be detected to generate a warning. If a procedure fragment is selected that contributes to a syntax error, some of the suggested replacements are formulated to fix the error, further helping the user.

Users may not know whether the added knowledge is correct. The procedure may be correct from a formal standpoint but not achieve the desired result. Constable tests new knowledge against examples as soon as it is entered to help find these problems.

It takes several steps to add new knowledge, so users can easily be lost. Users often do not realize and/or forget the side effects of the changes that require following up. Expectations from the PSM can be used to guide users through the initial steps in adding new knowledge. This is done through a script, as the next section shows. The approach is related to knowledge acquisition scripts [Tallis & Gil, 1999], though not as general.

This discussion of problems that users face indicates that providing help based on a combination of several different kinds of expectations may be key for non-programmers to add procedural knowledge. In the next two sections we show the help that can be given based on different kinds of expectations and describe how they are integrated in Constable. We begin by describing expectations from PSM task theories and then describe how the system makes use of expectations

derived from analyzing procedure interdependencies. In the following section we describe how the expectations are integrated by expressing expectations from background theories in terms of input-output type expectations.

3 Expectations from background theories

Expectations derived from the background theory in a problem-solving method are used to help clarify the purpose of the new procedural knowledge to be added by identifying its place in the PSM framework. Once this identification is made, initial templates are created for the new knowledge, which the user can refine until they perform the desired task. In this way, expectations from background theories help a user begin the process of defining new procedural knowledge to perform some task. In our approach, the background theory has two main components: (1) an ontology of concepts related to the task, and (2) generic procedural knowledge for performing each subtask.

This approach is general and can be applied to a wide range of generic tasks. In this paper we use an implemented problem-solving method for plan evaluation to illustrate the approach. 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 [Breuker & de Velde, 1994], it is a special case of assessment.

Each criterion for judging a plan is represented explicitly in this framework. Through experience with several intelligent systems for plan evaluation [Valente *et al.*, 1999], we have identified several patterns in the ways that the criteria are evaluated [Blythe & Gil, 1999]. These patterns are regularities that can be re-used across planning domains and provide guidance for knowledge acquisition. They are represented through an ontology of plan judgment criteria, called *critiques*, that is partially shown in figure 1. For example, `upper-bound` represents the class of critiques that can be evaluated by checking that some property of the plan has a value that is below a maximum value. Each class is identified with a pattern for evaluating a plan, implemented through generic procedural knowledge attached to the class. The PSM also includes concepts related to plans and the use of resources.

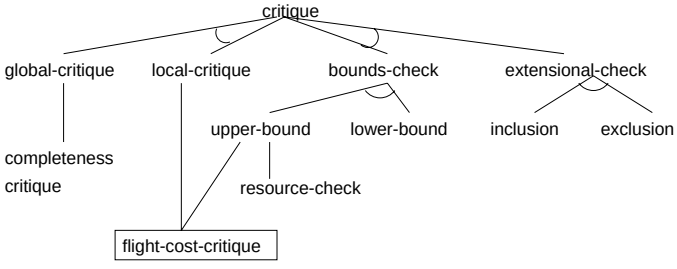

Figure 1: Different types of criteria for judging plans are part of the background theory of plan evaluation.

The second component of the task theory consists of generic procedural knowledge attached to some of the subtasks within the domain. In the plan evaluation domain, these subtasks are the generic critique types. For example, the following method says that a step satisfies an upper bound critique if and only if the actual value of the associated property is less than or equal to its maximum value. The tasks of estimating the actual and maximum values for the property are two methods that can be defined by the user with our tool.

capability: (determine-whether (obj (?thing is (inst-of thing)))
(satisfies (?bc is (inst-of upper-bound))))
result-type: (inst-of boolean))
method:
(check-that (obj (estimate (obj actual-value) (of ?bc) (for ?thing)))
(is-less-than-or-equal-to (estimate (obj maximum-allowed-value)
(of ?bc) (for ?thing))))

3.1 Using background theories in Constable

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 procedures needed complete each critique's definition. In the travel planning domain, for example, plans are itineraries for travel and the steps in plans represent reservations of flights, hotels and rental cars. One possible critique checks that no flight costs more than $500. This is implemented by defining the critique `flight-cost` as a subclass of both `local-critique` and `upper-bound`. The procedures 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 user completes the definition by defining methods to compute the actual amount (by retrieving the cost of the flight) and the maximum allowed amount ($500).

Figure 2 shows the main window through which a user defines the `flight-cost` critique in Constable. The tool presents questions of two kinds: those aimed at classifying the critique in the ontology, *e.g.* questions 2, 5 and 7, and those allowing the user to refine default procedural knowledge, which begin with the phrase "show me how to...". The questions are attached to the critique classes in the ontology and the tool asks them as it tries to classify the new critique. For instance, question 5, "Warn if flight-cost is too large?" is used to classify the critique as an upper bound. Once a positive classification is made, the tool gives the user an option to refine the procedural knowledge attached to the class. The generic procedural knowledge for this class include a default for "estimate the maximum allowed value of ..", so question 6 allows the user to refine this default for the flight cost.

The use of background theories to classify new knowledge and define default procedural knowledge helps to solve the first problem that users face in creating procedural knowledge: how to get started. The tool begins by asking questions about the nature of the knowledge to be added, and the default procedures are guaranteed to be applicable within the system. The task theory is also used to break the new knowledge into manageable pieces, an important step that is hard for users who are not programmers. However, refining the methods to compute actual values and maximum allowed values for flight costs, for example, can still be a daunting task requiring the tool to offer more assistance. Expectations based on

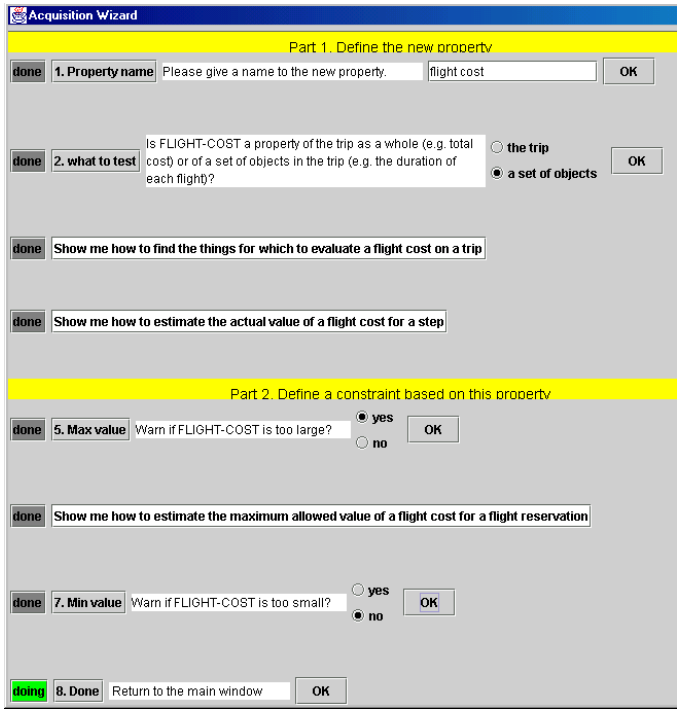

Figure 2: Constable's main window for defining a critique includes questions that classify the new critique within the ontology and questions that refine the generic procedural knowledge associated with the classification.

interdependency analysis are one source of this assistance.

4 Expectations from interdependencies

The expectations described in the last section come from background theories of tasks in problem-solving methods. Here we consider expectations that come from comparing the new procedural knowledge with the procedure syntax and with existing procedural or factual knowledge, which we refer to as *interdependency expectations*. Some examples of syntax-based expectations are that variables that are used in the procedure body are defined and that the rules of syntactic constructs such as "*if ... then ... else ...*" are observed. Some examples of expectations generated by comparing the new knowledge with existing procedural and factual knowledge are that relations and objects used are defined in the knowledge base, that other procedural knowledge that is used is defined, and that the relations and procedural knowledge used will return information of an appropriate type for the way it is being used (*e.g.* the procedure does not try to multiply a hotel by a number). These expectations are derived from interdependency models [Kim & Gil, 1999].

Figure 3 shows Constable's editor for procedural knowledge being used to define how to compute the maximum allowed value of the cost of a flight reservation. The purpose and body of the procedure are automatically paraphrased in English [Blythe & Ramachandran, 1999]. When the user selects part of the procedure to change, in this case "*Pittsburgh*", the editor suggests replacements in the lower panel, also automatically paraphrased. The suggestions are generated by analyzing the current knowledge base for possible terms and grouping them. Examples of groups are procedures or relations that could be applied to the current term or that have the same type. This approach hides the formal syntax while avoiding the challenge of full natural language processing.

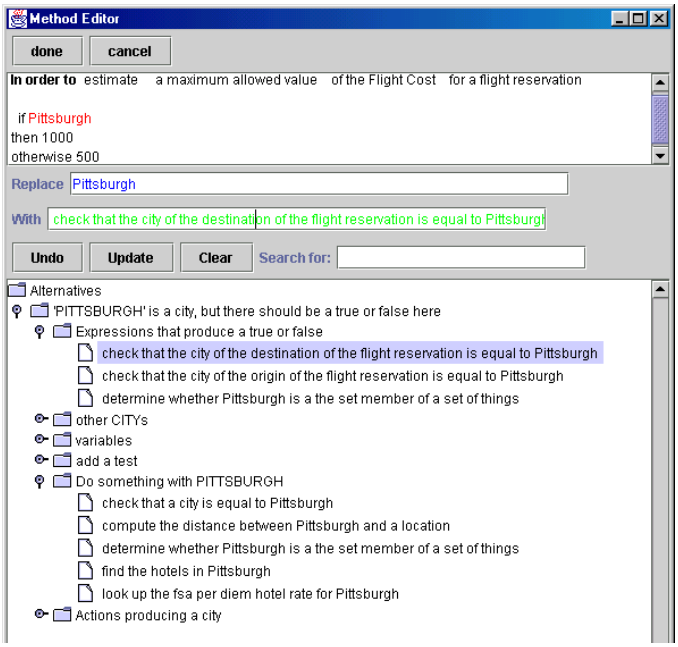

Figure 3: Constable's editor being used to define how to compute the maximum allowed value of the cost of a flight reservation.

We distinguish two ways that a KA tool can use interdependency expectations to help a user define procedural knowledge, called hard and soft expectations. For a *hard expectation*, the tool does not allow procedural information that violates the expectation to be expressed. For example, the only way for a user to introduce an "if" construct into a procedure is to select one from the suggested replacements, so the user can never express a procedure that violates the basic syntax of the construct. All syntax expectations are hard expectations in this editor.

For a *soft expectation*, the tool will allow a user to define procedures that violate the expectation, but will provide warnings, and possibly remedies, for the violation. For example the method in Figure 3 currently violates a type expectation. To compute the maximum allowed cost for a flight, the user specifies "if Pittsburgh then 1000, otherwise 500", but in the formal syntax, the keyword "if" must be followed by code that returns a boolean value. The editor shows an error message in the lower panel and, when "Pittsburgh" is selected to be replaced, suggests expressions that include "Pittsburgh" but will produce a boolean value, including "check that the destination city is equal to Pittsburgh" and "check that the origin city is equal to Pittsburgh". These expressions are produced by an anytime search algorithm. The suggestions

also include expressions that will not fix the violation, such as "compute the distance between Pittsburgh and a location".

Choosing to make an expectation hard or soft can impact how easily a user can express a procedure through successive replacements and how easily a user can be confused by an intermediate form of the procedure. Empirically, users find it useful to build intermediate expressions that violate interdependency expectations, such as in the above example or by using undefined procedures. This allows users to concentrate on other issues before fixing the violations. These are soft expectations in Constable. However it is not so useful for users to violate syntactic expectations; these are hard in Constable. Different KA tools may choose different expectations to be hard or soft, and the optimal choice may also depend on the skill level of the user.

Almost all compilers find and report errors based on input-output expectations. Constable goes beyond this in suggesting modifications to the existing code that would address the error. The suggestions are found by a breadth-first search through the space of possible sentences, taking the user-selected fragment as a starting point. A node in the search space is a term in the formal syntax, and it is expanded by applying all known applicable relations or procedures. The search terminates when terms are reached that resolve the expectation violation, or match a user-typed search string. Since the search space can be infinite, the suggestion module times out after a short time.

Users who are not programmers can be daunted by the need for executable procedural knowledge to have precise syntax with subgoals returning correctly typed information. The use of the expectations described above, in both hard and soft form, can provide significant help.

5 Integrating expectations from different sources

The previous two sections described how both expectations from background theories and method-based expectations provide valuable help for users to define procedural knowledge. In combination, the two kinds of expectations can provide more assistance than the sum of their parts. This is because useful information about the emerging procedural knowledge can be exchanged between the two sources.

There are several examples of this information flow when Constable is used to create the flight cost critique, summarized in the window in Figure 2. As we described earlier, the questions in Figure 2 are generated through expectations in the background task theory. Some of the questions are to classify the new critique in the ontology and some are to refine the attached procedural knowledge.

For each of the procedures to be defined in questions 3, 4 and 6, type expectations are sent from the task theory to the method analysis module, which uses them to help guide the user. Constable uses the task theory to assign types to each input variable and a desired output type, and calls the procedure editor. For example, the method "estimate the maximum allowed value of ..." has a desired type *(inst-of number)*, so the method editor will warn the user and suggest remedies if the method defined does not produce a number.

Some of the type expectations for the default methods are refined in the method analyzer by considering their interdependencies. For example, the user defines a method "estimate the actual value of flight cost for a flight reservation" in Figure 2. The input variable for this method has type *(inst-of flight-reservation)*, but this cannot be inferred from the task theory alone: it comes from the result type of the method "find the things for which to evaluate a flight cost on a trip". This is defined by the user to return a set of flight reservations. When this definition is complete, EXPECT's interdependency model is re-generated. It shows that the subsequent methods should take a flight reservation as input, and this information is passed back to the task theory from the method analyzer.

Information from the method analyzer is also used by the task theory in classifying the critique. For instance, once the user completes the definition of a procedure to "estimate the actual value of flight cost..", the method analyzer notes that it produces a number. This information is used in the task theory to classify the critique as a *bounds-check* according to the ontology of Figure 1. Question 5, "Warn if flight cost is too large?", attempts to classify the critique as a *lower-bound*. If the method to compute the actual value produced a different type, for example an airline, this question will not be asked. Instead, the task theory will classify the critique as an extensional critique and ask whether there are any preferred values, to test whether the critique is a *positive-extensional-critique*.

Information is passed between the PSM task theory and the method analyzer in both directions as the user defines a critique. The method analyzer makes use of type expectations from the task theory to help the user. It also passes refined type information to the task theory, based on the new procedural information and on interdependency analysis. The task analyzer uses this information as it classifies the new task. This interplay between the expectations from different sources significantly improves the guidance that can be given the user as the new knowledge is defined.

6 Preliminary experiments

We performed a preliminary evaluation of the approach by evaluating the performance of six subjects, who were not programmers, at adding and modifying critiques in the travel planning domain. In training, users followed written instructions to modify three critiques and add one new critique using Constable. In testing, users were asked to add and modify as many critiques as possible from a list of six. Subjects took an average of one hour to complete the training phase and thirty minutes working on the test critiques.

The simplest training task was to change a constant representing the maximum value of the distance between the hotel used and a meeting location. In the most complex task, the maximum distance for the hotel was defined as twice the distance of the closest available hotel. The procedure is:

```
(multiply (obj (find (obj (spec-of minimum))
                     (of (compute (obj (spec-of distance))
                                  (between (find (obj (set-of (spec-of hotel)))
                                                 (in (r-city (r-hotel ?t)))))
                                  (and (r-meeting (r-trip ?t)))))))
          (by 2))
```

All subjects entered this definition correctly using the tool. Even though they were following instructions, it is unlikely that this success rate would have been achieved without using Constable. The test tasks had the same range of complexity as the training tasks.

Adding or modifying a critique may require a number of steps to be completed. To measure the partial performance in the test tasks we counted each correctly defined method as a step, and also counted classifying the new critique correctly in the ontology as a step. Table 1 shows the average number of steps completed in each the first four test tasks. Every subject was able to make modifications to constant maximum values. Four of the six subjects were able to define a hotel cost critique according to the definition "If the city of the hotel is Pittsburgh, then the maximum hotel cost is $70, otherwise it is $120". Defining this kind of critique is beyond the scope of tools that do not allow users to define procedural knowledge.

Task	steps completed / Total steps
Modify constant maximum value	1/1
Max value is conditional on city	0.66 / 1
Complex new upper bound	1.4 / 3
Complex new upper bound II	1 / 3

Table 1: Average number of steps completed by subjects for each of four test tasks.

In addition to the pre-defined critiques, subjects attempted to add their own critiques to the plan evaluation tool. These critiques were written down before subjects worked through the training or test cases, so that they would not affect their choices. No subject succeeded in completely adding these critiques. Of the 26 critiques that subjects described, 17 could in principle be added through Constable. The others could be identified in the critique ontology but used relations and concepts, such as "bed-and-breakfast", that the tool did not include.

One shortcoming of the current tool is that subjects were not able to easily see all the concepts and relations available in the domain. The editor shows only those that are related to the current procedure definition. While this is helpful when users are making local changes to a procedure, they also need a way to see all available information. Also, the short time-scale of the experiment limited the time that subjects could spend thinking about the critiques. We hope to build a tool that the subjects would use on a daily basis that includes Constable, and investigate its impact.

After completing the experiment, several subjects whose job includes travel planning volunteered more critiques that would be useful in their work. It is interesting to compare these with the critiques that were expressed before the experiment: they all entail more complex procedural reasoning than the earlier critiques, but can be expressed in the critique ontology. Examples include "if the flight arrives late in the evening, the hotel should be near the airport, otherwise it should be near the meeting", and "prefer a hotel that is close to each of two meeting locations".

7 Discussion

Adding procedural knowledge to intelligent systems is a challenging task, but one that is necessary for the system to be useful to a wide range of users. We presented a novel approach in which expectations from background theories and from interdependency analysis are integrated to guide the user through the KA process. As well as providing assistance across a larger subset of the KA task than previous systems, the approach is able to combine information from the two sources to provide help that is not otherwise possible. In general, procedural knowledge should be added in concert with factual knowledge such as new concepts, relations and instances. Constable includes tools for adding factual knowledge which were not described here for space reasons. Information on more of the tools, but with less detail on the use of expectations, can be found in [Blythe *et al.*, 2001].

We are currently working on a number of applications of Constable. One is a travel planning tool that retrieves flight, hotel and other information from the web and uses constraints on partial travel plans to help users assemble an itinerary. Integrating Constable will help users specify individual preferences to assemble itineraries. We also intend to apply Constable to acquire procedural knowledge for reasoning about the biochemistry of DNA.

References

[Blythe *et al.*, 2001] Blythe, J., Kim, J., Ramachandran, S., and Gil, Y. 2001. An integrated environment for knowledge acquisition. Best Paper, *Proc. International Conference on Intelligent User Interfaces.*

[Blythe & Gil, 1999] Blythe, J., and Gil, Y. 1999. A problem-solving method for plan evaluation and critiquing. In *Proc. Twelfth Knowledge Acquisition for Knowledge-Based Systems Workshop*.

[Blythe & Ramachandran, 1999] Blythe, J., and Ramachandran, S. 1999. Knowledge acquisition using an english-based method editor. In *Proc. Twelfth Knowledge Acquisition for Knowledge-Based Systems Workshop.*

[Breuker & de Velde, 1994] Breuker, J., and de Velde, W. V. 1994. *CommonKADS Library for Expertise Modelling: Reusable Problem Solving Components.*

[Eriksson *et al.*, 1995] Eriksson, H.; Shahar, Y.; Tu, S. W.; Puerta, A. R.; and Musen, M. A. 1995. Task modeling with reusable problem-solving methods. *Artificial Intelligence* 79:293–326.

[Fensel & Benjamins, 1998] Fensel, D., and Benjamins, V. R. 1998. Key issues for automated problem-solving methods reuse. In *Proc. the European Conference on Artificial Intelligence.*

[Fensel & Motta, 1998] Fensel, D., and Motta, E. 1998. Structure development of problem solving methods. In *Proc. Eleventh Knowledge Acquisition for Knowledge-Based Systems Workshop*.

[Gil & Melz, 1996] Gil, Y., and Melz, E. 1996. Explicit representations of problem-solving strategies to support

knowledge acquisition. In *Proc. Thirteenth National Conference on Artificial Intelligence*.

[Kim & Gil, 1999] Kim, J., and Gil, Y. 1999. Deriving expectations to guide knowledge-base creation. In *Proc. Sixteenth National Conference on Artificial Intelligence*, 235–241. AAAI Press.

[Marcus & McDermott, 1989] Marcus, S., and McDermott, J. 1989. Salt: A knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence* 39:1–37.

[Musen, 1992] Musen, M. A. 1992. Overcoming the limitations of role-limiting methods. *Knowledge Acquisition* 4(2):165–170.

[Puerta *et al.*, 1992] Puerta, A. R.; Egar, J. W.; Tu, S.; and Musen, M. A. 1992. A multiple-method knowledge acquisition shell for the automatic generation of knowledge acquisition tools. *Knowledge Acquisition* 4(2):171–196.

[Swartout & Gil, 1995] Swartout, W. and Gil, Y. 1995. EXPECT: Explicit Representations for Flexible Acquisition. In *Proc. Ninth Knowledge Acquisition for Knowledge-Based Systems Workshop*.

[Tallis & Gil, 1999] Tallis, M., and Gil, Y. 1999. Designing scripts to guide users in modifying knowledge-based systems. In *Proc. Sixteenth National Conference on Artificial Intelligence*. AAAI Press.

[Valente *et al.*, 1999] Valente, A.; Blythe, J.; Gil, Y.; and Swartout, W. 1999. On the role of humans in enterprise control systems: the experience of inspect. In *Proc. of the JFACC Symposium on Advances in Enterprise Control.*