Task Learning by Instruction in Tailor

Jim Blythe
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
+1 310 448 8251
blythe@isi.edu

ABSTRACT
In order for intelligent systems to be applicable in a wide range of situations, end users must be able to modify their task descriptions. We introduce Tailor, a system that allows users to modify task information through instruction. In this approach, the user enters a short sentence to describe the desired change. The system maps the sentence into valid, plausible modifications and checks for unexpected side-effects they may have, working interactively with the user throughout the process. We conducted preliminary tests in which subjects used Tailor to make modifications to domains drawn from the eHow website, applying modifications posted by readers as 'tips'. In this way the subjects acted as interpreters between Tailor and the human-generated descriptions of modifications. Almost all the subjects were able to make all modifications to the process descriptions with Tailor, indicating that the interpreter role is quite natural for users.

Categories and Subject Descriptors
I.2.6 [**Artificial Intelligence**]: Learning – *knowledge acquisition.* H.5.2 [**Information Interfaces and Presentation**]: User Interfaces - *Interaction styles*.

General Terms
Human Factors, Design, Experimentation.

Keywords
Knowledge acquisition, task learning by instruction, reasoning about actions.

1. INTRODUCTION
In order for intelligent systems to be applicable in a wide range of situations, end users must be able to modify the way intelligent systems perform their tasks. However, users who are not programmers face a number of difficulties in making these changes. The precise domain representation used within the system may be unfamiliar even to a domain expert, and so may the formal language used for procedure knowledge.

This problem has been addressed in previous work through Programming by Demonstration (PBD), in which the user selects examples which the system generalizes in order to induce or modify an action description [8,9]. This approach can be inefficient if examples are hard to formulate, for instance in complex domains, or if many examples are required to learn the correct procedure. Other approaches have used problem-solving methods, interdependency analysis and smart editors to help users work with the procedure representation [3,6]. In these approaches, however, users are often required to make implementation-level decisions about changes to procedures, during which it is easy to lose track of changes or make mistakes.

In some cases, however, it would be more efficient to tell the system what change should be made in a relatively informal language, and have the system modify the procedure appropriately. This is the approach taken in task learning by instruction [5]. For example, to ensure that a step is only performed in the appropriate situation, the user might provide an instruction such as 'Don't seek authorization when the cost is below $2000', and interact with the process reasoning tool to help it modify the process knowledge appropriately. The instruction can directly specify the point where the process knowledge should be changed and indicate the relevant domain features, but the user does not need a detailed knowledge of the domain representation used to specify the process and does not need to make low-level decisions about how to modify the process definition.

Such a system must be able to (1) recognize the user's instruction as a proposed modification to a knowledge base of procedures, (2) map the modification into a candidate change with the correct syntax, and (3) detect whether the change may affect the system's overall problem-solving behavior in unanticipated ways, based on its procedure knowledge base, and take steps to rectify this. All of these steps require a partnership between the human user and the program. The program can suggest probable steps where changes are to be applied, and valid expressions that may match the user's sentence. However, the user must make the final decision on whether to apply a modification proposed by the program, making use of knowledge about the task that the program does not possess.

We introduce Tailor, an implemented system that follows the steps described above to help users modify a knowledge base of procedure information, used by a PRS-like task execution system called SPARK [10]. We report on preliminary tests in which five subjects made modifications to two previously unseen domains

IUI'05, January 9--12, 2005, San Diego, California, USA.

using Tailor, based on paraphrases of user modifications posted on the eHow website. All the subjects were able to make some modifications to the processes, providing evidence that this role of interpreter between the system and human-generated inputs is a natural one. The system is also under test in an office domain, where users modify the process by which equipment is purchased.

The next section gives a scenario in which the tool is used to modify a procedure for making office purchases and briefly describes the approach taken in Tailor for the three steps described above. The following section covers the approach in more detail. We then describe initial user experiments with process descriptions drawn from the eHow website. We follow with a discussion of related work, lessons learned and our future plans for Tailor.

2. SCENARIO

Consider a process description for purchasing equipment such as a laptop within a company, including the following steps: the user identifies the laptop to be purchased, then an initial purchase request form is generated and submitted to two different line managers for authorization. Assuming this is received, the order is placed with the purchasing department who will complete the purchase. The tasks are defined hierarchically in Spark, allowing the tool to track the progress of each task, for example tracking the form as it is emailed to the managers and then the purchasing department. Figure 1 shows the top level procedure definition; the lower level procedures are defined in the same way. Figure 2 shows the overall process as Tailor presents it to the user. This is automatically generated from the Spark definitions, along with action and predicate templates that show how to generate the text, and how to refer to variables once they are bound. An example template is shown in Figure 1.

```
{defprocedure "Buy Laptop"
 cue: [do: (purchase $item $criteria)]
 precondition: (= laptop $item)
 body:
 [context: (and (User $user) (Called $user $name))
  seq: [do: (find_laptop $item $criteria $seln)]
    [do: (complete requisition_form $form $seln)]
    [do: (obtain_authorizations $form $seln)]
    [do: (place_order $seln)]
    [context: (= (list_index $seln 0) $pseln)
          do: (print
    " %s:  *** Purchase of laptop %s completed ***"
               [$name $pseln])]]}


{defActionTextTemplate
    (get_authorization $manager $form)
    "Get authorization on $form from $manager"}
```

Figure 1. An action definition for purchasing a laptop

Suppose that the process has been used successfully until, for the first time, the purchase cost is below $2000. At this level, authorization is not required, but the planning tool is unaware of this and makes a request for authorization as part of its plan. The user notices the problem and modifies the planning knowledge by typing the sentence *'You don't need authorization when the cost is below $2000*'. Tailor takes this sentence and relates it to its process description. It reasons that the word 'authorization' probably refers to the step 'obtain authorization from managers' and not its substep 'get authorization from the first manager'. It reasons that 'cost' probably maps to '(computer-Total-Price $laptop)' based on the predicates in its knowledge base and the variables that are bound at this step in the plan.

Tailor therefore proposes to make the step conditional on this value being less than $2000. In Figure 3, the proposed change is presented to the user. A summary of the change is provided in the 'Summary' panel, and the new definition is shown in the 'Procedure Description' panel, with the new condition highlighted. The effects of this change go beyond the current plan. Since the procedure definition will be changed, all future plans in situations that match this condition will also be changed.

Tailor also reasons about the consequences of making this change. The step to send the request to the purchasing department has a precondition that authorization has been received, so it may fail when the 'obtain authorizations' step is skipped. Tailor can't know for certain that it will fail, because authorization might already be in its database, but it warns the user that this is a potential problem and suggests three ways to handle it, from which the user chooses to ignore the authorization precondition when the cost is below $2000 and send the request to purchasing anyway. At this point, the modified process description can be used in Spark to produce the desired result.

3. APPROACH

Tailor follows the three steps described above.

3.1 Recognizing the Intent as a User Instruction

Tailor attempts to recognize each sentence from the user as a proposed modification to the process knowledge base. We consider three categories of modification:

- adding a new step in the body of a procedure definition, e.g. "Always get authorization before submitting a purchase request", when no authorization step is currently in the plan,
- modifying the parameters used in an existing step, e.g. "Use galvanized nails to fix planks in an external deck", when a step to fix the planks is already in the plan, and
- modifying the conditions under which an action is performed, e.g. "Only sand the planks if they are visibly gouged", when the sand step is already in the plan.

Although there are other types of modification, for example changing the timeline or relative order of existing tasks, these three have been found in practice to cover a wide range of user modifications, as we discuss in the section on experimental results. We are working to support all three, but currently full support is only given for the third category, modifying the conditions for an action.

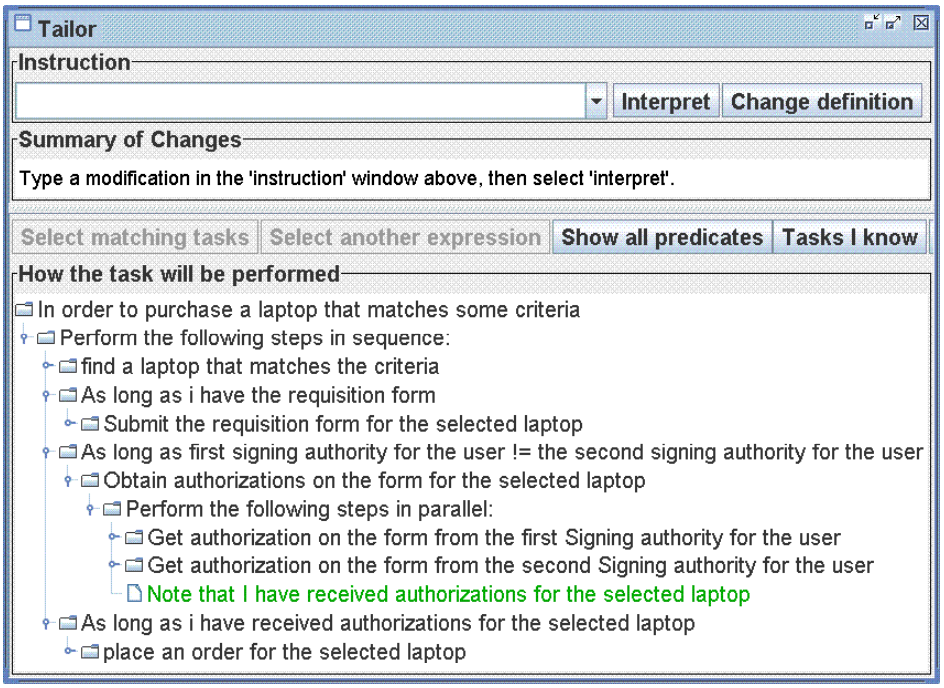

Figure 2. Tailor's display of the initial process description

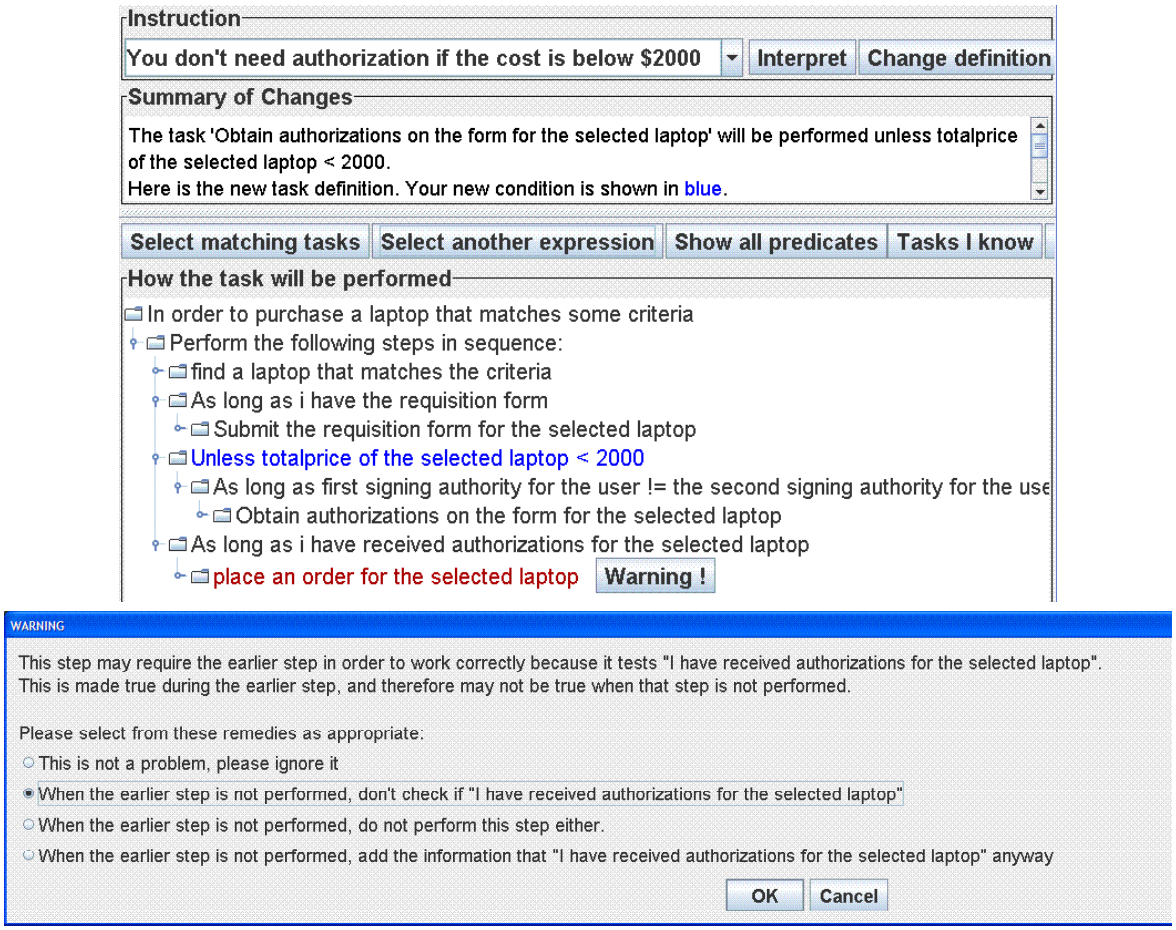

Figure 3. Tailor's summary of the modification based on the user's instruction includes a warning and suggested remedies.

Each category is modeled by a template in Tailor that stores information fields that must be filled using the words in the sentence. For example, the template for adding a new step has two information fields: the step to be added and optionally another step used as a temporal reference point. The sentence in the scenario, "You don't need authorization when the cost is below $2000", matches the template for modifying conditions, which has two fields: the step and the condition. Templates also contain information on how to provide feedback to the user about the proposed modification.

In order to match a template and decide which words in the sentence contribute to which information field, we combine declarative matching rules with a parser, JavaNLP [7]. After running the parser, the nodes of the tree are traversed in a depth-first order such that the words, which form the leaves, are traversed in their sentence ordering. The declarative rules implement a finite state grammar and can mention the word or a substring, or a node's label in the parse tree in order to make a decision or set internal state. The parser allows the rules to be more general than a keyword matcher. For example, Figure 4 shows the parse tree for the sentence in the initial scenario, and two rules that fire on an internal node in the parse tree. By the first rule, the 'field' variable is set to 'condition' when the internal node with label 'SBAR' is encountered, which signifies a clause introduced by a subordinating conjunction. The second rule states that, when the field is 'condition' and a word with label 'NN' is seen (signifying a noun), it is added to the words in the condition field of the template. In this way nouns that appear grouped with words like 'when', 'if', 'since' or 'because' are added to the condition field of the template.

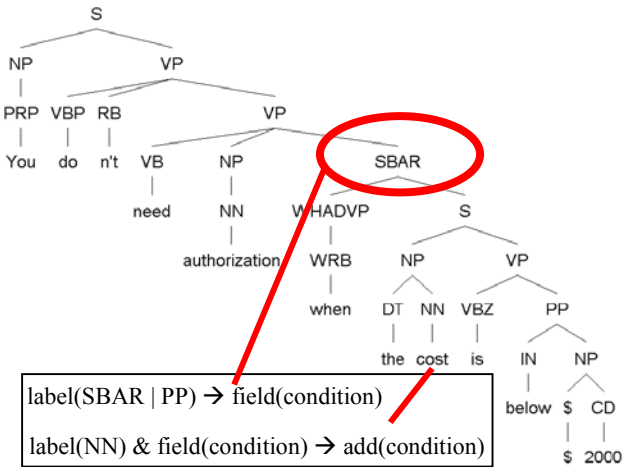

Figure 4. Declarative rules act on a parse of the sentence to assign words to information fields in the template.

3.2 Mapping Sentences to Hypothesized Changes

Once the words of the sentence are assigned to each field, they are used to map the sentence into a well-defined, plausible modification of the original process description. Template fields that refer to existing steps are matched to the steps in the current process description and the closest match is usually returned. For fields that refer to conditions, Tailor searches for compound expressions built from terms in its domain knowledge base that both match the words in the field and use variables that will be bound when the relevant step is reached. In both cases, Tailor uses WordNet synonyms [4] to increase the breadth of matches that it finds. For example, the sentence fragment 'the cost is below $2000' is mapped to

```
(and (Computer_Total_Price $seln $x)  (< $x 2000))
```

A phrase such as 'the cost' in the example sentence fragment may refer to an object (as the syntax implies), or a variable, or a slot of another unmentioned object (as is the case here). The user may not know which is the case, since it depends on decisions made during the implementation of the domain model. When unstated objects are involved, Tailor must identify them in order to provide a well-defined modification.

The set of objects that a conditional expression may refer to is not fixed, but depends on the point during the process at which the condition is tested. For example, the expression may refer to a selected laptop after the `find_laptop` step has completed, but not before. In addition, several objects may have a cost and a choice must be made between them. Framing the set of choices requires a detailed knowledge of the domain, while making the choice may require task knowledge only available to the user. Therefore, the user and the tool must work together to find the best matching expression.

We use a dynamic programming approach over a graph of data types to build a ranked list of plausible conditions [1,2]. To simplify the interface, Tailor initially picks the first element of the ranked list and presents this modification to the user, but also allows the user to view the list and select a different modification. Figure 5 illustrates the approach, used to match the expression "when the vendor is in the US". The nodes in the graph correspond to data types in the domain, and the arrows represent relational queries that can result in new objects, pointing towards the type of the new object. This graph can be built before search begins. When search begins, the relevant step in the process is first identified, and used to build the list of variables that the expression can refer to. Each variable is added to the node corresponding to its type, for example, the variable '$selection' is shown under the node 'Laptop'. Constants mentioned in the expression, e.g. 'US' are also added to the appropriate node.

On each iteration, each arrow in the graph is inspected. If new terms were created at the base of the arrow in the previous iteration, then a new compound term is created at the tip of the arrow in the current iteration. For example, in the first iteration, a new object of type 'Vendor' is created by following the arrow from node 'Laptop' to 'Vendor'. This object corresponds to the result of the query '`(vendor $selection $v)`'.

As each new object is added, it is tested as a match to the user's expression. An expression is considered to match if the text that would be used to present it contains the words in the expression, allowing for synonyms from WordNet. If several expressions match, higher rank is given to those using fewer synonyms, and with more words appearing in the correct order.

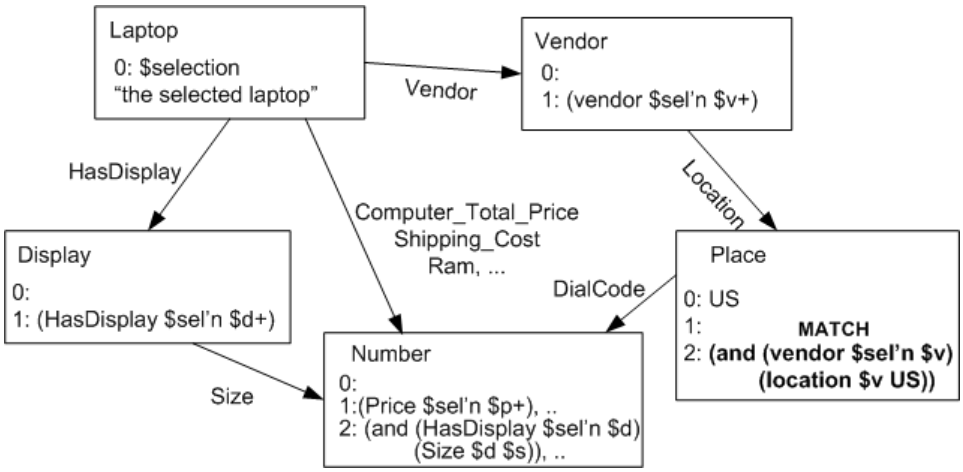

Figure 5. The type graph is used to build up a list of expressions that contribute to the matched conditions

3.3 Reasoning About the Effect of Changes

A plan execution system typically comprises a highly interconnected structure of tasks and methods in which constraints and information are passed between tasks. Changes suggested by the user are likely to have consequences on the system's overall behavior that require changes to other procedure definitions, some foreseen and some unforeseen by the user. Tailor reasons about the overall problem-solving behavior through a symbolic evaluation for one or more top-level goals in order to warn the user of potential problems and suggest fixes.

For example, Figure 2 shows a warning and a choice of remedies that are generated after the modification from the initial scenario is identified and applied. Step 4 is highlighted in red, and a warning button is placed next to it. When the user clicks on this, it shows that this step, which places the final order for the laptop, has a precondition that authorization is received. This condition may not be true if the `obtain_authorizations` step is skipped, so the plan may fail. Tailor offers three possible fixes – (1) mark authorization as achieved in the current situation, (2) relax the precondition, so that authorization is not tested in situations where `obtain_authorizations` is skipped, or (3) don't execute step 4, placing the order, under these conditions. The second choice matches the semantics of 'authorization' in this case, but sometimes the other choices are preferable. Tailor's method for finding problems is general, but the remedies are hand-chosen for each type of problem.

Since Tailor is working with a symbolic evaluation of the plan, it cannot be certain that an issue it detects will be a real problem at runtime. For example, a database might be available that provides the needed information. We use the following rule of thumb to provide warnings that are likely to be relevant: rather than analyze potential problems in a single evaluation, we compare the evaluation traces before and after the modification and only make warnings about issues that are introduced by the modification. In the scenario example, the precondition interaction is introduced when the step to obtain authorizations is made conditional, and so the warning is generated.

4. EXPERIMENTAL VALIDATION

We aim to verify that users who are not familiar with Tailor can modify realistic process descriptions, where neither the initial process nor the modifications are chosen by us. To do this we used data from eHow[1], a popular website with thousands of descriptions of how to perform useful tasks, in categories such as home repair and health care. Many of the advice pages on eHow can be viewed as a process descriptions in text form. These pages often have links to tips from users, which can take the form of modifications to the processes. We asked subjects to translate some of these modifications for Tailor in process models that were hand-built to capture the advice pages.

For example, one page in home maintenance describes how to remove texturing called 'popcorn' from a ceiling using a scraper. One user tip advocates squirting the popcorn with water first to soften it, while another points out that, if the popcorn is scraped dry, it does not need to be sanded and so this is the easier and quicker option. The second tip is as follows: "*The surface left behind dry-scraped, unpainted popcorn gives a knockdown texture appearance that is suitable for painting. Cannot be achieved if popcorn is removed with water-spray technique. Surface is ready for paint after scraping -no sanding required."* In order to reduce problems matching a domain ontology, we included the step to squirt water in Tailor's model, then showed subjects this paragraph and had them tell Tailor to "*skip the sand steps if the ceiling is dry*".

We made an estimate of how many of the user tips on eHow could be captured by Tailor. Many of the advice pages do not have user contributions and many user contributions are critiques of the process, or suggestions to use completely different approaches, rather than modifications to the existing process. We made a survey of examples in the 'home repairs and maintenance' category of eHow, selected because it is likely to yield a good combination of process-like advice and user-added tips. There are 692 advice pages in this category. Within a subset of 200

[1] eHow "eHow.com – Clear Instructions on How to Do (just about) Everything"

focussing on structural repairs, 45 were found with user tips that modified the process. Of these, 28 specified extra steps to be added to the process, 17 discussed changing equipment, e.g. using screws rather than nails, 6 suggested adding or modifying conditions under which steps in the process were applied and 6 discussed other aspects, such as refining the way substeps are performed. We intend to cover the first three modification types in Tailor. Currently, Tailor can be used to add or modify conditions on steps and we only picked examples of this type.

We built process models in Spark by hand for two of the advice pages in the 'home repairs and maintenance' section of eHow, hanging drywall from the ceiling, and removing textured 'popcorn' from the ceiling, for which user tips modified the conditions under which substeps are performed. We also built a domain from the 'health and safety' section with the same characteristic and used it for training. In each case we ensured that the underlying domain model was rich enough to encode the user's advice as well as the initial process description.

Five subjects were trained on modifications to Tailor's domain model for the health page and then asked to make 3 modifications based on the two drywall advice pages. The modifications varied in difficulty and in the required interaction with Tailor. For example, the modification to skip the sanding steps required applying the same condition to two steps. A method for doing this had been demonstrated during training.

Four of the five subjects were able to complete all the modifications successfully, although one subject needed two general hints: to use shorter sentences if Tailor is confused and frame sentences to avoid a certain parsing error. The fifth subject completed two of the three assignments successfully. The subjects tried different sentences until Tailor understood correctly. We define each time that a user entered an alternative instruction for a modification, rather than fine-tuning the instruction through provided interactions, as a separate 'attempt' at the modification. Figure 4 shows the number of tasks that are completed successfully within a certain number of attempts, shown on the x axis. Almost half of the modifications are successfully completed within 3 attempts, and all except two within 12 attempts. As can be seen, two subjects took 20 and 21 attempts. However, along with most of the subjects, they did not find the task onerous.

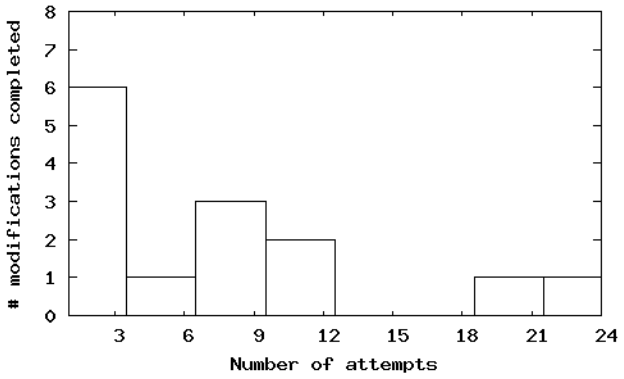

Figure 6. The number of instructions attempted before a modification is understood. The y axis shows the number of problems solved with that number of instructions.

Although in two cases the subjects made over 20 attempts, no modification took longer than around 13 minutes to complete. Different styles are discernible in the subjects' approaches: for the same assignment, one subject took three attempts, at a rate of around 1 attempt per minute, while two other subjects completed the task more quickly but with 9 and 12 attempts respectively, with a rate slightly below 5 attempts per minute. Figure 7 shows the cumulative number of modifications made by all subjects within a given time period on the x axis. The steep initial curve shows that most modifications were made swiftly, with 11 of the 14 taking less than 6 minutes each.

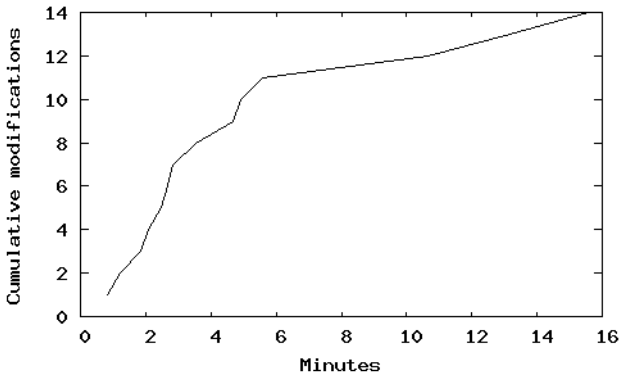

Figure 7. The cumulative number of modifications that were completed by a given time, shown on the x axis.

The subjects were able to use Tailor to choose from alternative modifications or steps to change as well as communicate the desired modification. In one case, Tailor did not provide as its first choice the condition that three of the subjects wanted to use, but included it in its list of possible choices. Two of the subjects used the menu of choices to change the condition, and one rewrote the instruction to direct Tailor to choose the desired condition. In the case where two steps were to be modified in the same way, three of the four subjects who successfully matched the condition were able to use the step menu to create the desired modification. The fourth attempted to convey this through the text instruction, but this is currently beyond Tailor's scope.

This initial experiment is an encouraging indication that end users can modify real process descriptions using Tailor when we did not design the process or its modification. Based on our sample of modifications on eHow, we expect that within the next year Tailor will be able to recognize most of the modifications that are posted on the website as tips. We are planning a more comprehensive experiment to verify this.

5. RELATED WORK

One of the most closely related pieces of work is that of Huffman and Laird on Instructo-Soar and related agents [5]. These agents receive instructions from users about how to achieve a task that is currently being executed, and use situated explanation to reason about the advice in the context of this task. The most important differences are that Tailor reasons about an abstract process description, rather than one that must be situated in a particular task, and Tailor does not assume enough domain knowledge to support explanation of the advice. Work in Tailor on recognizing user intent and mapping to well-defined modifications is also novel.

Programming by demonstration (PBD) [8,9] allows users to specify procedure knowledge by giving examples to a system that

learns the procedure by induction. This has the advantage that in many cases the user does not have to reason about the abstract procedure, just its behavior in certain cases. However, the user must provide scenarios that contain most of the information that is relevant for learning the procedure. In applications programming environments, for example word processing and operating system macros, this information is readily available, but it is less so in less constrained tasks, for example fixing a broken washing machine. Demonstration is sometimes an inefficient way to convey information, requiring the user to provide numerous cases in order to rule out undesired generalizations when it would be simpler to articulate the conditions under which some action should or should not be performed. In general, we view these techniques as complementary rather than as alternatives.

The Expect system provides an explicit language for procedures and several tools to help users create or modify procedural information using english-based editors and reasoning about interdependencies [3,6]. Tailor differs from Expect in taking short sentences of instruction as the principal source for procedure modifications, and using a standard procedure language, a variant of PRS, rather than a proprietary one. In using short sentences it provides assistance at a higher level, since the user says more about the purpose of the change and less about the details of implementation than in a system like Expect.

Myers [11] describes the Advisable Planning system that has a language for high level advice. It was initially developed for an HTN planner, although it is currently also being used with Spark. The constructs in the language constitute advice about choosing particular fillers for roles in procedures, e.g. the agent or instrument, or choosing the procedure to use to perform a task. These constructs specify values for possible choices in the procedure knowledge base rather than modifying which substeps are performed and when. They may provide a useful additional target language for interpreting user instructions in future versions of Tailor.

Etzioni and colleagues [12, 13] describe a natural language interface for communicating with appliances, based on NL interfaces to databases. This work is concerned with setting goals based on task models rather than modifying the models. In addition, it places stronger constraints on the query and target languages in order to provide performance guarantees. Tailor is intended for applications where these requirements are not met, and relies on interaction to resolve ambiguity in the instruction if possible. However, a way to provide performance guarantees that take into account the use of interaction and synonyms is an interesting avenue of future work.

6. CONCLUSIONS AND FUTURE WORK

We have described Tailor, a tool for task modification by instruction that follows three basic steps: (1) recognize the kind of modification the user intends, (2) map the sentence into one or more plausible, syntactically valid modifications, and (3) detect potentially unintended consequences of the modification, warn the user and suggest remedies. Our preliminary evaluation of Tailor used process descriptions and user-supplied modifications from the eHow web site, to demonstrate that users can translate modifications for Tailor to apply in several new domains.

Task learning by instruction is a promising way to help users make these modifications that has been studied relatively little recently. It allows a more direct statement of steps to change and relevant conditions than pure example-based approaches, but frees the user from some implementation decisions by putting some of the burden onto the task learning system. However, several PBD approaches are also able to take interactive guidance from the user, and it is more productive to view these approaches as complementary rather than as alternatives. We plan to investigate the advantages of hybrid strategies with colleagues working on example-based learning techniques.

We are currently working to improve the scope of the system to recognize and reason about modifications that add new steps, when the actions referred to are already known to the system, and modifications that change the parameter bindings in steps. With these modifications we expect to be able to handle the majority of process modifications seen on eHow.com and similar sites.

We currently model Tailor's overall process as a set of cascading steps, first recognizing the intent of the sentence, then finding a mapping, and finally finding and resolving issues in the new process definition. However, these choices are not independent of one another. For instance, the intent of the expression 'Disconnect the main power before you unscrew the plate' depends on whether a step to disconnect the main power is already in the process description, perhaps after the unscrewing step. Thus, the mapping step should influence the intent recognition. Similarly, it is reasonable to prefer mappings that introduce fewer or simpler unintended consequences in the process description. We plan to move to an iterative model of the process that will support this reasoning.

The experiments we described are in domains where the plan execution system cannot apply primitive actions to change the world. However, we have also used Tailor in domains where SPARK is executing instances of the operators during the learning process. This provides a richer environment for user feedback that we plan to explore and will be further studied.

7. ACKNOWLEDGMENTS

I am grateful to the following people whose comments improved this paper and the work reported, including Yolanda Gil, Jihie Kim, Tim Chklovski and Karen Myers. Varun Ratnakar implemented much of the system. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No.NBCHD030010.

8. REFERENCES

[1] Blythe, J. Integrating Expectations from Different Sources to Help End Users to Acquire Procedural Knowledge, *in Proceedings of IJCAI '01* (2001)

[2] Blythe, J., Gil, Y., Incremental Formalization of Document Annotations through Ontology-based Paraphrasing, *in Proceedings of WWW '04* (New York, NY, June 2004)

[3] Blythe, J., Kim, J., Ramachandran, S., Gil, Y., An Integrated Environment for Knowledge Acquisition, *in Proceedings of IUI '01* (Santa Fe, NM, Jan 2001)

[4] Fellbaum, C. (ed.) Wordnet: an Electronic Lexical Database, MIT Press, May 1998

[5] Huffman, S. and Laird, J. Flexibly Instructable Agents, *Journal of AI Research*, 3, 1995

[6] Kim, J. and Gil, Y. Deriving Expectations to Guide Knowledge-base Creation, *in Proceedings of AAAI '99* (1999)

[7] Klein, D. and Manning, C. Fast Exact Inference with a Factored Model for Natural Language Parsing, *in Proceedings of NIPS '02* (2002).

[8] Lau, T., Bergman, L., Castelli, V., Oblinger, D. Sheepdog: Learning Procedures for Technical Support, *in Proceedings of IUI '04* (Feb 2004)

[9] Lieberman, H. (ed.) Your Wish is My Command, Morgan Kaufmann, San Francisco, 2001

[10] Morley, D. and Myers, K. The SPARK Agent Framework, *in Proceedings of* AAMAS '04 (New York, NY, July 2004)

[11] Myers, K. Planning with Conflicting Advice, *in Proceedings of* AIPS '00 (2000)

[12] Popescu, A., Etzioni, O. and Kautz, H. Towards a theory of natural language interfaces to databases, *in Proceedings of IUI'03*, Miami Beach, FL, January 2003

[13] Yates, A., Etzioni, O. and Weld, D., A reliable natural language interface to household appliances, *in Proceedings of IUI'03*, Miami Beach, FL, January 2003