Helping End Users Modify Procedures by Instruction

Jim Blythe and Varun Ratnakar

USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
{blythe,varun}@isi.edu

Abstract

Many useful planning applications are handled by plan execution tools, such as PRS, that keep track of several interacting goals and tasks, and different ways to expand them, using procedure definitions. I describe Tailor, an implemented tool to help end users modify the procedure definitions used in these tools by interpreting instructions given as short sentences. This approach allows a natural and flexible interaction style in which users can refer to tasks by name or by their arguments, may skip some of the details of a conditions and use synonyms. Tailor uses search to find task modifications that match the user's input, warns about potential problems that the modifications may introduce and suggests fixes. We conducted preliminary tests using Tailor to modify domains drawn from the eHow web site, applying modifications posted by readers as 'tips', with promising results.

Introduction

We consider the problem of allowing end users, who are not experts in planning or knowledge engineering, to modify an existing domain for a plan execution system such as PRS [Georgeff & Lansky 89] or SPARK [Morley & Myers 04]. This task fits within the "knowledge refinement and maintenance" category of the competition. In many ways, this is a simpler task than creating an entirely new domain, yet end users still face a number of challenges. They probably will not know the precise terms used for tasks, concepts and relations in the domain representation, even if they are experts in the domain. They may not know how a change to one procedure definition may affect the system's overall task performance when many procedures are chained together. Even the syntax of the procedures and domain relations can be daunting.

Tailor helps end users modify procedure definitions by giving instructions as short sentences, for example "you don't need authorization if it costs less than $2000", or "email my manager before placing the order". Tailor maps the input sentence to a set of plausible modifications by first identifying it with one of a set of modification templates, for example "*make substep ?s be conditional on expression ?e*", and then filling out values for the parameters of the template by search. The process is interactive: the system shows the result of the modification that was most likely intended, allowing the user to fine-tune the interpretation or choose an alternative. Tailor also analyses the effect of the modification using a symbolic evaluation of a given top-level goal. If the procedure modification may remove a step that asserts a needed fact in the database, for example, Tailor gives a warning and makes suggestions for different ways to fix the problem. However, the user can always dismiss the warning, since the problem may never arise in ground applications.

Tailor's instruction-based approach is very flexible. It allows Tailor to be embedded in applications that communicate with the user in different ways, for example, using dialog or speech understanding as well as in more traditional interfaces. It also allows users to refer to tasks and conditions even if the precise terms are not known, allowing Tailor to hypothesize the modification by matching synonyms and using search. The research contributions of our work include defining a sufficient and concise set of instruction templates, efficient search techniques to generate plausible modifications and analysis techniques for PRS-style plan execution systems. Tailor is implemented in java and communicates with a separate plan execution system through the Open Agent Architecture [Martin et al. 99]. It is designed to be used with SPARK, but is applicable to any PRS-style system. In the next section we describe a scenario where Tailor is used and step through the techniques involved. The following section briefly describes an empirical evaluation. Finally we discuss the approach and planned future work.

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. When this is received, the order is placed with the purchasing department for completion. 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 $crit $seln)]
    [do: (complete req_form $form $seln)]
    [do: (obtain_authorizations $form $seln)]
    [do: (place_order $seln)]
    [context: (= (list_index $seln 0) $pseln)
        do: (print "Purchase of %s completed"
                   [$pseln])]]}

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

Figure 1. Procedure definition for laptop purchase

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 guesses that the word 'authorization' probably refers to the step 'obtain authorization from managers' and not its substep 'get authorization from the first manager'. It guesses 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 user can override Tailor's guesses and select alternatives if desired. 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.

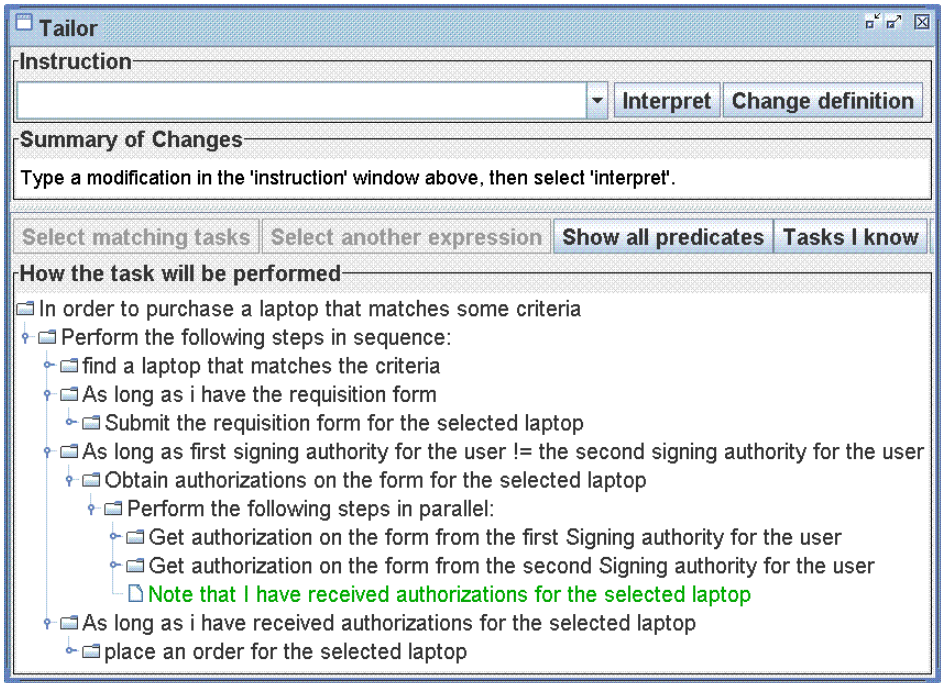
Approach

Tailor interprets the instruction in three steps: mapping the instruction to a template, filling in the roles of the template and reasoning about the effect of the changes.

Mapping the instruction to a template

Tailor currently considers three types of modification: (1) adding a new substep into a procedure body, (2) modifying the parameters of an existing step, e.g. "use screws instead of nail to fix planks to the deck" and (3) modifying the conditions under which a substep is performed, e.g. "only sand the planks if they are visibly gouged". Each category is modeled by a template in Tailor that holds 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.

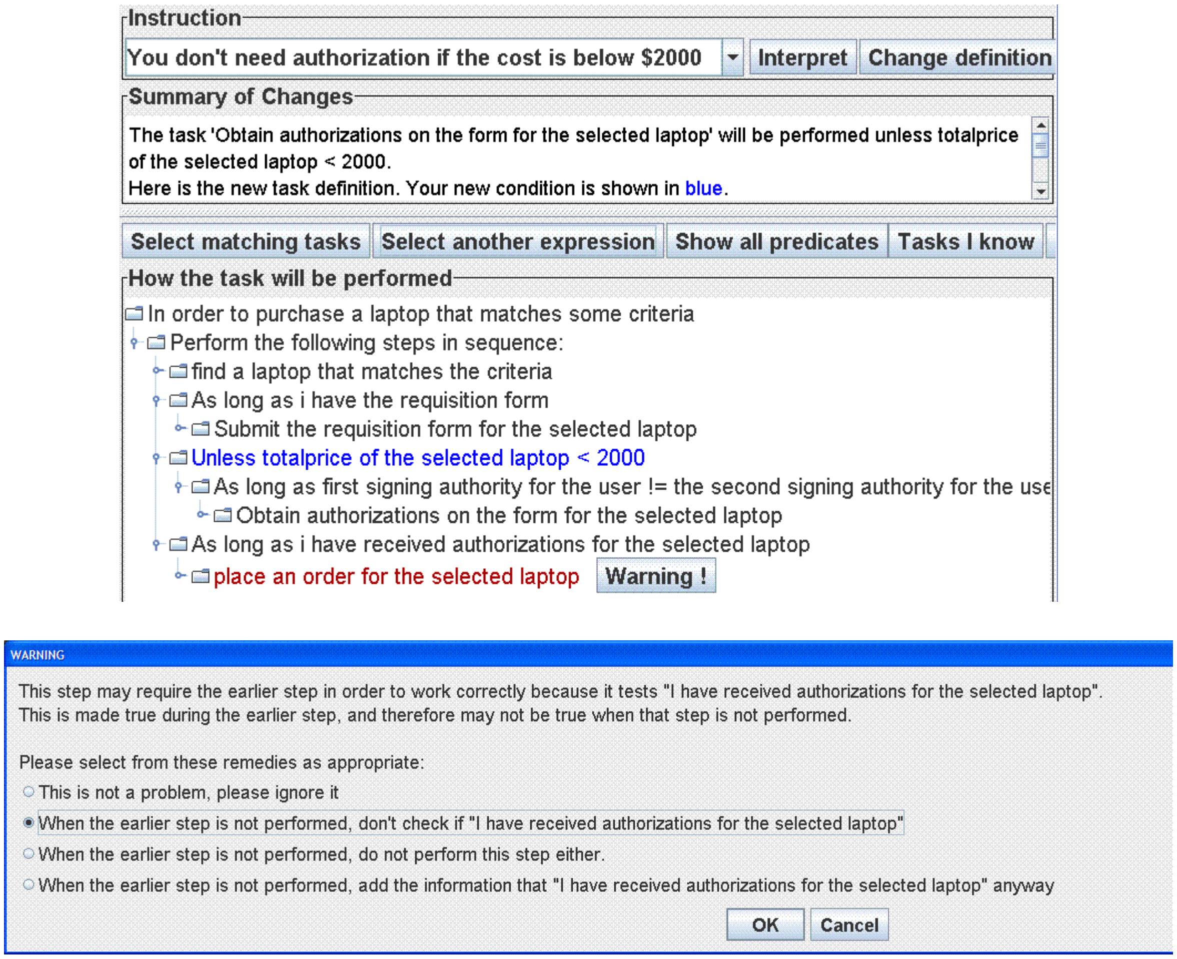
To link a user's sentence to the appropriate template, we first use an off-the-shelf parser, JavaNLP [Klein & Manning 02] and then use declarative rules to match keywords and elements in the parse tree. The rules also determine how words in the sentence are assigned to different fields in the template. The example sentence is linked with the condition modification template, and the word "authorization" is assigned to the 'substep' field while the words "the cost is below $2000" are assigned to the 'condition' field. More details for this and the subsequent steps can be found in [Blythe 05].



Type a modification in the 'instruction' window above, then select 'interpret'.

In order to purchase a laptop that matches some criteria
- Perform the following steps in sequence:
  - find a laptop that matches the criteria
  - As long as i have the requisition form
    - Submit the requisition form for the selected laptop
  - As long as first signing authority for the user != the second signing authority for the user
    - Obtain authorizations on the form for the selected laptop
      - Perform the following steps in parallel:
        - Get authorization on the form from the first Signing authority for the user
        - Get authorization on the form from the second Signing authority for the user
        - Note that I have received authorizations for the selected laptop
  - As long as i have received authorizations for the selected laptop
    - place an order for the selected laptop

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



You don't need authorization if the cost is below $2000

The task 'Obtain authorizations on the form for the selected laptop' will be performed unless totalprice of the selected laptop < 2000.
Here is the new task definition. Your new condition is shown in blue.

In order to purchase a laptop that matches some criteria
- Perform the following steps in sequence:
  - find a laptop that matches the criteria
  - As long as i have the requisition form
    - Submit the requisition form for the selected laptop
  - Unless totalprice of the selected laptop < 2000
    - As long as first signing authority for the user != the second signing authority for the use
      - Obtain authorizations on the form for the selected laptop
  - As long as i have received authorizations for the selected laptop
    - place an order for the selected laptop

This step may require the earlier step in order to work correctly because it tests "I have received authorizations for the selected laptop". This is made true during the earlier step, and therefore may not be true when that step is not performed.

Please select from these remedies as appropriate:
- This is not a problem, please ignore it
- [✓ When the earlier step is not performed, don't check if "I have received authorizations for the selected laptop"]
- When the earlier step is not performed, do not perform this step either.
- When the earlier step is not performed, add the information that "I have received authorizations for the selected laptop" anyway

Figure 3. Tailor's summary of the proposed modification includes a warning and suggested remedies

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 found and will match words from either the formal expression definition or the text-generation template.

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.

We use a dynamic programming approach over a graph of data types to build a ranked list of plausible conditions [Blythe & Gil 04]. Tailor picks the highest-ranked element of the list and presents this modification to the user, but also allows the user to select a different modification. The graph is built before the interactive session. 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 ‘$seln’ is shown under the node ‘Laptop’. Constants mentioned in the expression are also added to the appropriate node.

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 when `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.

Empirical evaluation

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 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.

We estimated how many of the user tips on eHow could be captured by Tailor. Within a subset of 200 process descriptions focusing on structural home 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

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

are performed. We intend to cover the first three modification types in Tailor. At the time of testing, Tailor could be used to add or modify conditions on steps and we only picked examples of this type.

Five subjects were trained on modifications to Tailor's domain model for a health domain and then asked to make 3 modifications based on two user tips in a home repair domain. 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 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.

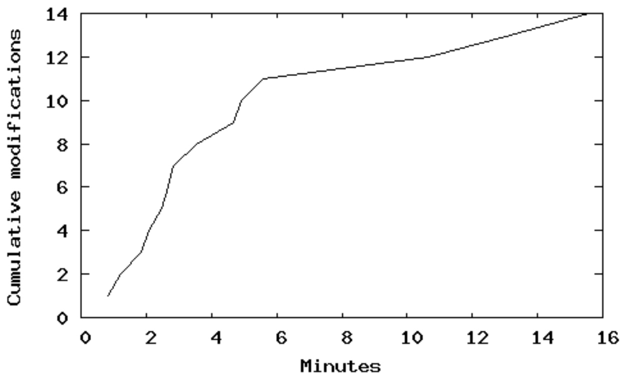


Figure 4. The cumulative number of modifications completed by a given time.

Discussion

Tailor is 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. The tool is also undergoing tests as part of an office assistant, managing equipment purchase, travel reimbursement and similar tasks.

Tailor’s has a modular implementation that allows templates for new kinds of instructions to be added relatively easily. Similarly, support for different action languages, including a subset of PDDL, could be provided without major alterations. We believe that the techniques used in Tailor can also be applied to HTN and subgoaling planning domains, but this would require a more extensive revision of the template set, the problems detected and their remedies.

Task learning by instruction is a promising way to help users make modifications to knowledge about actions that has been studied relatively little recently. It allows a more direct description of the steps to change and the 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. While interpreting a user’s instruction in a less constrained context would be beyond the state of the art, here we exploit the knowledge that the instruction is a modification to a plan execution domain that is understood by the tool. This constrains interpretation to a small number of templates, each of which has fields whose values are bounded by the current plan and the domain procedures, objects and relations. The interactive nature of the tool also allows the user to correct mistakes in interpretation.

One of the most closely related pieces of work is that of Huffman and Laird on Instructo-Soar and related agents [95]. 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. Tailor reasons about an abstract process description, rather than one that must be situated in a particular task, and does not assume enough domain knowledge to support explanation of the advice. Our work in Tailor on recognizing user intent and mapping to well-defined modifications is novel.

We are currently working to show the sufficiency of a given set of instruction templates for the modifications that are syntactically possible in a given planning system. This can be used to demonstrate theoretical completeness in a system like Tailor. We will also expand the experimental results as Tailor’s scope increases. A promising direction of research is to combine instruction-based approaches with programming by demonstration [Liebermann 01], either to improve the interpretation step based on examples, or to begin with an example but fine-tune the modification through instruction. We intend to explore this direction in the coming year.

Acknowledgments

We gratefully acknowledge discussions with Yolanda Gil, Jihie Kim, Tim Chklovski, Karen Myers and others. 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.

References

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

[Blythe 05] Blythe, J., Task Learning by Instruction in Tailor, *Proc. Intelligent User Interfaces 2005* (IUI 05)

[Georgeff & Lansky 87], Georgeff, M. and Lansky, A., Reactive Reasoning and Planning, *Proc. AAAI* 1997

[Huffman & Laird 95] Huffman, S. and Laird, J., Flexibly Instructable Agents, *Journal of AI Research*, 3, 1995

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

[Liebermann 01] Liebermann, H., *Your Wish is my Command*, Morgan Kaufmann, San Francisco, 2001

[Martin et al. 99], Martin, D., Cheyer, A. and Moran, D., A Framework for Building Distributed Software Systems, *Applied Artificial Intelligence* vol 13, number 1-2, pp 91-128, January-March 1999

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