TellMe: Learning Procedures from Tutorial Instruction

Yolanda Gil, Varun Ratnakar, and Christian Fritz
USC Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292
gil@isi.edu, varunr@isi.edu, fritz@isi.edu

ABSTRACT
This paper describes an approach to allow end users to define new procedures through tutorial instruction. Our approach allows users to specify procedures in natural language in the same way that they would instruct another person, while the system handles incompleteness and ambiguity inherent in natural human instruction and formulates follow up questions. We describe the key features of our approach, which include exposing prior knowledge, deductive and heuristic reasoning, shared learning state, and selectively asking questions to the user. We also describe how those key features are realized in our implemented TellMe system, and present preliminary user studies where non-programmers were able to easily specify complex multi-step procedures.

Author Keywords
Natural instruction, tutorial instruction, procedure learning.

ACM Classification Keywords
H5.m. Information interfaces and presentation (e.g., HCI)

General Terms
Human Factors.

INTRODUCTION

Over the last few years, many successful approaches have been proposed to allow end users to teach procedures through demonstrations [Li et al 10; Castelli et al 10; Chen and Weld 08]. From a few demonstrations, a system induces a general procedure that generalizes from the particulars of the examples shown by the user. However, when procedures are complex it is hard to create demonstrations that cover the space of possible generalizations particularly if the user is to provide only a few examples.

A complementary approach would be to teach procedures through tutorial instruction, a method commonly used by people to teach procedures to other people. In tutorial instruction, the teacher provides a natural language description of procedures using general situations and abstract objects [Clark et al 01; Webber et al 95]. This is in contrast with situated instruction or demonstrations where a particular state is used to illustrate the procedure [Huffman and Laird 95; Thomaz and Breazeal 08]. Tutorial instruction is a concise way to communicate complex procedures, and can be supplemented with demonstrations or practice to improve the learning process [Fritz and Gil 11].

Alas, natural tutorial instruction is plagued with omissions, ambiguity, oversights, unintentional inconsistencies and errors [Gil 11]. In addition, teachers often make incorrect assumptions about the student's background knowledge and learning abilities and state lessons in a way that may be hard for the student to follow. Humans can learn despite such imperfections in natural instruction, and we would like to have systems that have that ability.

Our goal is to create an intelligent system that can learn from tutorial instruction of procedures expressed in natural language and in a way that a human would find *natural* to provide. We present a novel approach that combines several key features: 1) the use of a command line interface to guide the user to express instruction based on what the system already knows, 2) the use of paraphrase patterns to map the user's instruction into commands that the system understands, 3) sharing with the user what the system assumes of the instruction as well as alternative assumptions that could be possible, 4) deductive and heuristic reasoning to complete and correct the instruction, and 5) facilitating disambiguation through option presentation to the user. Based on this approach we have developed TellMe, an intelligent system that can learn procedures from natural tutorial instruction.

We begin describing in more detail the challenges of handling natural user instructions about procedures. We then discuss key design features that we incorporate in our approach to address those challenges, and illustrate how these features manifest themselves in the TellMe user interface. We then describe in detail the TellMe system which implements our approach, and show some preliminary user studies to assess whether our new approach is viable and to obtain initial feedback.

IUI'11, February 13-16, 2011, Palo Alto, CA, USA.

CHALLENGES OF LEARNING PROCEDURES FROM TUTORIAL INSTRUCTION

Our goal is to allow users to specify procedures using natural instruction, which raises some important challenges:

1. *Natural user instruction is typically in textual form and therefore hard to interpret*. The system must control user input while not being too constraining as to be unnatural.
2. *Natural user instruction is incomplete and ambiguous, so the system must fill the gaps*. The user may skip steps, leave out details of steps such as necessary inputs to a step, not include conditions of steps such as required object types, not mention argument assignments, or not state explicit links among the steps.
3. *Natural user instruction can be unintentionally incorrect*. The system would have to be able to handle such imperfections, and make assumptions about how to disambiguate or correct the instruction.
4. *Natural user instruction requires that the user formulate the lesson based on reasonable assumptions about what the system knows*. The user should not have to know details of the system's internal prior knowledge. Therefore, the user may refer to terms and concepts that the system does not know about and the system should be able to cope with that. In addition, the user should be able to make reasonable assumptions of what the system has learned so far during the lesson.
5. *Natural tutorial instruction requires that the system should participate in the interaction when needed*. It should acknowledge when it understands and ask when it cannot understand. At the same time it should only ask necessary questions, taking an active role in figuring out on its own what questions the instruction raises in the first place, and answering as many questions as possible itself. A system that demands from the user lots of additional input over what the user uttered would not be considered to learn in a natural way.

A more detailed account of the nature of these and other challenges in learning procedures from natural instruction can be found in [Gil 11].

A simplifying assumption that we make in this paper is that the procedures that we target can be described based on the inputs they use, the outputs they produce, and that the links between the steps are producer-consumer links only. A procedure can be modeled as a graph with two types of nodes: executable *components* and *data objects*. The edges in the graph connect components with their data object inputs and outputs (see Figure 2, bottom right for an example). This accommodates a dataflow model among steps akin to data-centric workflows [Gil et al 10]. TellMe uses an underlying workflow system that provides a presentation for workflows as well as reasoning capabilities. As we will see, this basic model is already very rich and sufficient for representing interesting classes of procedures. The framework currently accommodates some forms of iteration, but would need to be extended for procedures that require conditionals, controlled iterations, and other types of links between steps.

TELLME: ALLOWING NON-PROGRAMMERS TO TEACH PROCEDURES THROUGH NATURAL TUTORIAL INSTRUCTION

We describe the main features of our approach using an example of the interaction experienced by one of our users who is a non-programmer and used our TellMe system to create a procedure.

Throughout the paper we use scenarios where the system learns procedures executed by airplane pilots to patrol an area looking for oil pollution from ships. The user is teaching the kinds of reconnaissance tasks that pilots do in the Belgian Navy, which are scenarios that we chose for our evaluation as we will describe later on. The system starts off having a number of primitive actions for recording the situation with a variety of instruments, including infrared and ultraviolet cameras, SLAR cameras, and digital picture and video cameras. There are also primitive actions to send alerts and reports back to the base, and to generate initial estimates of the volume of the spill.

Figure 1 shows an example of the utterances and the interaction of a user with the system to teach a procedure where a plane is to descend closer once a spill is found, then take videos and send them to the headquarters, and to record the GPS readings and send them along as well. The utterances are verbatim what the user would type. We will show later a screenshot of the current user interface, but for now we want to abstract from the interface details and focus on what the user and the system are communicating.

Key Feature: Exposing Prior Knowledge

A challenge that users face when teaching a system is to figure out what the system already knows or what capabilities it has. Lessons always build on prior knowledge, using it as building blocks to the instruction.

Our approach is to constrain the user's input with a command line interface that completes the user's utterance based on the objects and actions that are already known to the system [Groth and Gil 2009]. In this way the system exposes known actions and object types, which serve as building blocks to the user's expression of each instruction command. Here is an example of how the system exposes what it knows about properties of positions: when the user types "descend to a position with" then the system shows a pull-down menu that includes "descend to a position with height", "descend to a position with longitude", and "descend to a position with latitude".

User: "**find oil spill, descend to a position of height 200**"

<The system shows the user that it assumes that meant to descend after finding the spill. It also shows the user an alternative interpretation where the descent was meant to happen before finding the spill. It asks the user to either accept the assumed interpretation and if not to choose the alternative.>

User: "**film the spill**"

<The system indicates it did not understand that>

User: "**record videos and send them**"

<The system shows the user that it assumes that meant to iterate over each of the videos and send each in turn, since the send action is defined for sending one document at a time.>

<The system shows the user that the result of sending the videos is a series of message receipts.>

<The system shows that it assumes the position to be after the descent, it also shows the alternative interpretation that it is the position before descending. >

User: "**record GPS reading**"

<The system shows the user that it assumes that the instruction meant to record the GPS reading over the position after descending. It also shows the user an alternative interpretation which is to record the GPS reading at the position when the spill was first found. It asks the user to either accept the assumed interpretation and if not to choose the alternative.>

User: <selects the latter option>

User: "**record image**"

<The system shows three interpretations, one for the action to record IR image, another for record UV image, and another to record SLR image>

User: "**send thickness image**"

<The system shows the user that it is not familiar with the term high level alert, but that it assumes it is a kind of alert given the context in which the term is used. The user does not have to interrupt the instruction and define it now.>

<The system shows the user that the send action requires some evidence as input, and that it assumes that to be the output of the record GPS reading action.>

Figure 1. Overview of the interaction between a user and TellMe to teach a procedure to find and report oil spills in the water from a plane. The utterances are verbatim what the user would type. The TellMe user interface for this interaction is shown in Figure 2.

Key Feature: User Input as Controlled Natural Language

One important challenge that we need to address is that while natural language is a very natural way to provide tutorial instruction, interpreting unconstrained natural language is far beyond the state of the art.

Our approach is to use a paraphrase-based interpretation system that matches the user's utterance against a set of pre-defined paraphrase patterns, following the approach in [Gil and Ratnakar 2008]. Each paraphrase pattern is associated with a set of primitive commands that the user would have to use in order to have the intended effect that is described with the paraphrase pattern. The paraphrase patterns are exposed to the user through the command line interface described above.

For example, the utterance "descend to a position with altitude 200" is mapped to a paraphrase pattern component-as-verb +output-object +output-property + output-property-value. This paraphrase pattern is tied to a command that finds a component whose name matches the verb and adds it to the procedure. It further determines which of its defined outputs matches the uttered output object and asserts the output property value for the output property of the object corresponding to that output.

When an utterance cannot be mapped to any paraphrase pattern, the system indicates so to the user and then the user has to reformulate that instruction. This is the case with the utterance "film the spill" in Figure 1.

Several studies have found that users bring up new terms in any domain following a Zipf's law and there are always new terms that come up (e.g., [Bugmann et al 01]). When a new term appears in an utterance, TellMe will make assumptions about what it might mean. For example, when the user utters "send thickness image" and the system is not familiar with that term, it will assume that the term refers to an object (as opposed to an action or a property), and that it is a way to refer to an IR image since that is a type of image output by a step that is already in the procedure.

The combination of the command line interface and the paraphrase-based interpretation system gives the user the illusion of entering free text while the system actually is controlling what the user can input in ways that are amenable to understanding and interpretation.

Key Feature: Shared Learning State to Establish Trust

An important principle in user interface design is establishing user trust. A user needs to understand what the system is doing about the input she provided, and trust that the system is taking appropriate action. In our case, the system should give feedback to the user about what it is learning from the instruction. It must do so unintrusively, more as a nod than a detailed report, so that the user can focus on continuing with the lesson. Users need to know what the system has understood and learned so far as the lesson progresses.

Our approach is that the system always shares its internal learning state. For example, in many cases the user's instruction is ambiguous and the system creates alternative interpretations, each resulting in a different procedure hypotheses. To show that it is considering these hypotheses, it shows them to the user. She is always asked to select one of them.

For example, in the third utterance the user specifies "record GPS reading". The user did not say from what position to take the reading. In this case, the system generates three interpretations and shows them as options in the history window. The first interpretation is that the image

should be taken at the position after the descent. But it is possible that the user meant the position before the descent, and that is presented as a second option. The third interpretation considers taking the reading from yet another position that the user may want to describe later. The user has to select one before continuing, and the top option is selected by default. As we will see next, TellMe uses heuristics to rank these options, and because it considers the first interpretation of the three to be more likely it will rank it first.

Key Feature: Deductive and Heuristic Reasoning

One important challenge is that natural instruction is often incomplete. Therefore, the system has to address those shortcomings if it is to learn the complete procedure. Our approach is to use deductive and heuristic reasoning.

Deductive reasoning is used to make assumptions about the objects and steps in the procedure in order to create constraints on objects that are underspecified. The system is effectively performing deductive reasoning to infer what is not mentioned in the instruction. All the constraints shown in the top right panel were deduced by the system, and most refer to objects that were not mentioned in the instruction. For example, the user does not mention that the input to the procedure is an area to survey to find the spill, but the system deduces that from what the instruction says. Also, the fact that taking a picture results in a new image being created is not mentioned in the instruction, but the system adds that to its procedure hypothesis.

Deductive reasoning is also used to interpret new terms that the system has never seen before. Recall that based on their role in the paraphrase patterns the system assigns a syntactic category. Through deduction, the system infers what is the type of new terms and possibly other constraints based on their role in the procedure. In our example, "thickness image" will be classified as a type of image.

The second kind of reasoning used in TellMe is heuristic. Heuristic reasoning is used to figure out what information about the procedure is still missing given the instruction so far, and what are possible ways to complete it. These heuristics essentially create possible completions or corrections of the procedure hypothesis that the system created from the user instruction. TellMe shows the user options that are ranked heuristically.

Heuristic reasoning makes the instruction more natural in that the system not only has identified what issues to resolve, which would result in questions to the user. The system has gone further in taking the initiative to formulate possible answers to those questions. This makes the instruction more natural because this is something that teachers expect from human students.

Key Feature: Selective Questions

An important principle in designing effective user interfaces is to take into account the cost of requesting user interventions. Because instruction is incomplete, the system may have many possible interpretations and therefore it could ask many questions to the user to determine which is the one that the user intended. Yet, later instruction may address those questions and so the user intervention was unnecessary and would not be considered natural. Although a user in teaching mode can be expected to be more willing to cooperate than in other circumstances, the system should not insist on asking questions constantly just to satisfy its learning goals to disambiguate and to complete the instruction.

Addressing this challenge is difficult, because if the system postpones all its questions then there may be a large space of possible candidate interpretations that would make learning very unmanageable.

We use eager questioning to ensure that a single procedure hypothesis is chosen by the user. The system asks the user to select among procedure hypotheses when several are possible. We use lazy questioning for other matters. For example when an unknown term is used in the instructions, the system makes assumptions about it and proceeds without interrupting the user with questions. This is the case in Figure 1 when the user refers to a "thickness image" which is an unknown term.

LEARNING FROM TUTORIAL INSTRUCTION IN TELLME

TellMe reasons about 1) the current user's utterance, 2) the results of the interpretation of prior utterances, and 3) its prior knowledge about the objects and actions in the domain of discourse. During this process, the system will detect omissions in the instruction and generate possible alternative interpretations to present to the user as options. Any interpretations that are found to be inconsistent with the system's prior knowledge are ruled out. As the system processes the instruction, it may detect incorrect aspects and give it back to the user for reconsideration. The options are ranked heuristically and presented to the user for selection. At any given point, the user and the system are pursuing a single option as the active procedure hypothesis.

We first describe the user interface, and then describe four major modules in the system to process user instruction.

TellMe User Interface

Figure 2 shows a screenshot of the user interface that corresponds to the interaction shown in Figure 1. The user enters her utterances (e.g., "Find oil spill") in the command line on the top left. TellMe tries to provide the user with a menu of options to complete the current utterance.

When the utterance is completed, TellMe will show on the right hand side the leading hypothesis for the procedure being taught. At the bottom a dataflow diagram is shown. At the top, a set of constraints is shown, most of them inferred by the system. Green ovals show the inputs to the procedure, in this case the area the user wants to survey.

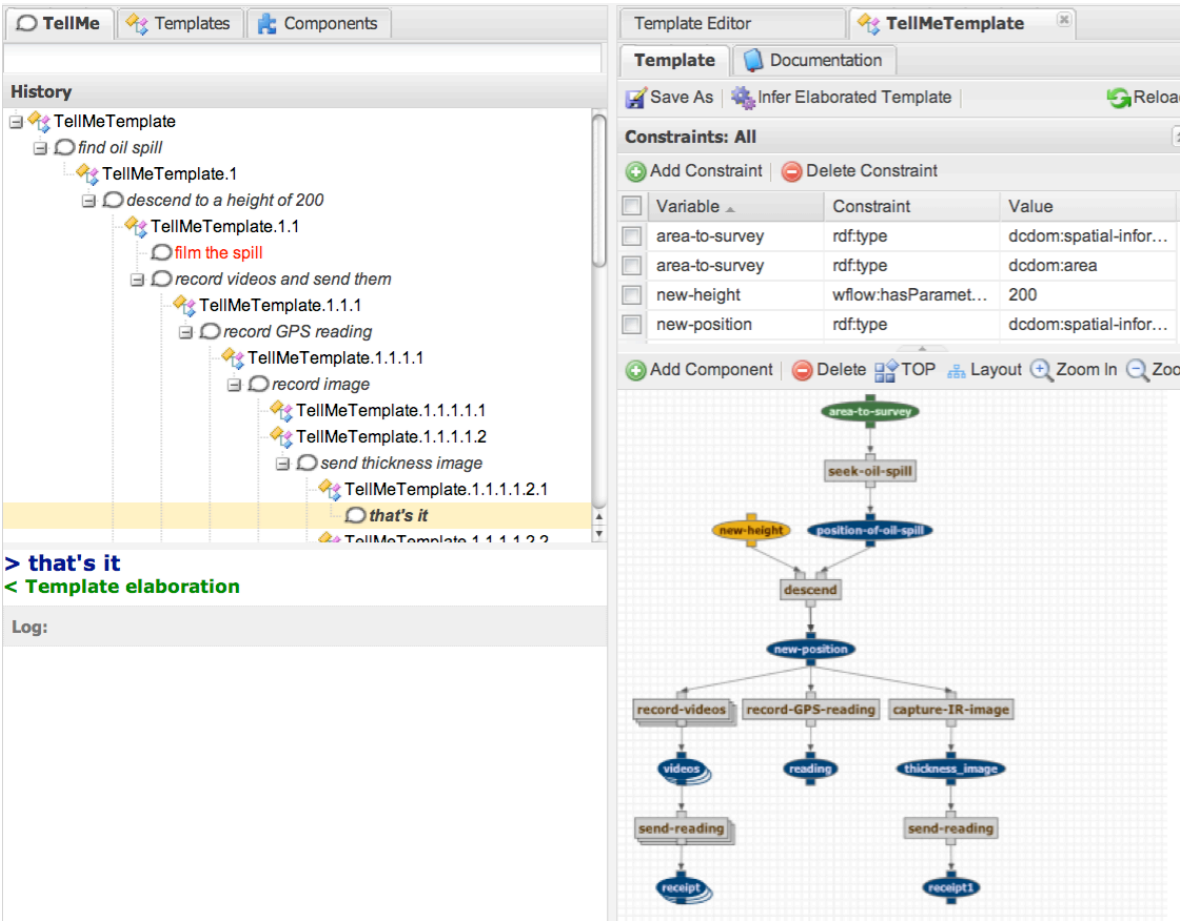


Figure 2. Screenshot that corresponds to the instructions and interaction in Figure 1.

The previous utterances of the user, registered by the system are shown in the History area on the top left.

When TellMe is unable to parse or understand an utterance, it is shown in red in the History. This gives the user the option to rephrase her input. This kind of interaction allows users to quickly learn how to communicate effectively with the system – as we found in our formative user studies (see below).

When TellMe does not understand a particular word in the user's instruction, it will create a new object for it in the procedure but will mark it in red to point out that it did not understand it. It will further infer as much as possible by context or from background knowledge. Entering "Find oil spill in the bay", e.g., when the system does not know what "bay " is, TellMe will mark it read, but also infer that "bay" is an area, because it knows how to find spills in an area. Users are not required to immediately answer questions about the new object.

When TellMe finds several possible ways of understanding a user's utterance, it will create several options, listed in the History window. Before continuing with the instructions, the user can see the respective graph of each of these by clicking on them. This way she can choose the one she wants to continue working on. For example, the utterance "record image" has three interpretations, and are shown as three possible branches of the option tree shown in the History.

A significant benefit of the History window is that the user can always go back if she later changes her mind about a choice she made or if she types something she would like to undo. She can then continue from that previous version from the History.

The user finishes a teaching session by typing "the end" or "that's it". TellMe will then further elaborate the learned procedure, as we describe in more detail below.

TellMe Matcher: From User Utterances to Commands

The TellMe Matcher builds on our prior work on mapping to-do lists into declarative agent capabilities through paraphrase pattern matching [Gil and Ratnakar 06]. In that approach, each formal action corresponds to a set of alternative paraphrase patterns that a user could use to state it. In TellMe, each command that would modify a procedure is associated with a set of paraphrases. For example, the utterances "descend to a new position" (a verb), or "descending to a new position" (a gerund, as in "after descending to a new position, record gps readings") both map to a single command `AddNodeWithOutput +c +o` to add a component +c="descend" and an output +o="new position" to the procedure being created. In

addition to paraphrase patterns for commands, TellMe also has a collection of paraphrase patterns to express step orderings (e.g., "first find spill then descend"), steps, data, and data properties. For example, "descend to a position with altitude 200" and "descend to a position with 200 altitude" are alternative ways to express properties of the position after descending. Step ordering are enforced via dataflow links in the procedure graphs. All the paraphrase patterns are domain independent. TellMe also has a collection of domain-specific paraphrase patterns that represent equivalent terms to refer to each component, data, and property. For example, the "capture-IR-image" component correspond to the terms "capture IR image", "take IR picture", "capture infrared image", "record IR image", and so on. There are paraphrases that represent morphological variations, for example plurals of nouns and the gerunds above. We have two dozen paraphrase patterns for the oil spill domain.

The TellMe Matcher uses all these paraphrase patterns to generate the possible utterance completion options to the user, as described in [Groth and Gil 09]. The result is a specific set of formal commands to modify the procedure whose rendering is the user utterance. For example, for "after descending to a new position, record gps reading", TellMe generates a set of commands:

```
AddNodeWithOutput n1 +c=descend +o=new-position
AddNode n2 +c=record-gps-reading
AddLink n1 n2 +v1=+o
```

All these paraphrase patterns used by the TellMe Matcher are created manually beforehand by analyzing a corpus of typical expressions of procedures in the domain. This is typical practice for speech recognition and natural language input systems. This component could in principle be extended with WordNet synonyms (synsets) and with more extensive grammars than our current paraphrase-based approach. However, our current simple approach fulfills the important function of guiding user to express their utterances in a way that has the structure needed by the system to turn them into formal commands.

The TellMe Matcher also handles new terms in the instruction. For example, an utterance such as "descend to a safe position" mentions a "safe position" which is not defined anywhere in the system. This is a very common occurrence in instruction, as studies have shown that the number of terms used in subsequent commands increase and continue to grow following Zipf's law [Bugmann et al 01]. The TellMe Matcher hypothesizes the grammatical category of the new term based on its function in the paraphrase. In the example above, it will assume that "safe position" is a type of object since it is mentioned where an object type would be mentioned. The TellMe Matcher also assigns it a type based on the other known terms in the utterance. In the example, since "descend" outputs a position then "safe position" is assumed to be of type position. The new term is nevertheless marked as never seen before, so that the user can either corroborate these assumptions or retract the instruction and retry expressing what she intended. Additional information about the new term can be learned later on, but sometimes just going along with the assumptions is enough information to be able to learn a procedure that will execute just fine even with limited knowledge about the new type of object.

TellMe Creator: Extending the Procedure

The TellMe Creator adds to the existing procedure sketch according to the commands that result from the TellMe Matcher. The underlying workflow framework has a graphical editor for procedures, where users drag and drop components and connect them through links. The commands that are input to the TellMe Creator correspond to editing commands in that editor.

The instruction, however, may result in underspecified editing commands. In the graphical editor, one has to select the argument (or port) where an input is linked. In contrast, instruction tends to be incomplete because a user would rarely mention in an instruction what argument identifier the input corresponds to. Users would expect the system to figure this out based on the definition of each action. The TellMe Creator does this by checking the types of the inputs and assigning the link to the input that has a compatible type. For example, if the instruction stated "find spill and descend at that position", a component to descend is added and an object of type position is added. In this case the input to descend is of type position, so the link is compatible and the argument identifier is clear. If more than one link is possible, then the TellMe Creator generates an alternative procedure hypothesis for each link. The set of hypotheses is carried to the next module.

The TellMe Creator handles some forms of incorrect instruction. An interesting case that arises is when the types assigned to objects are not compatible. An example is the instruction "estimate volume of oil spill with a UV image", which results in a component to do the estimate being added and an input to it also being added with type UV-image. In this case, the definition of the estimate component specifies that its input must be an IR image, not a UV image. The instruction is considered incorrect, and TellMe will present the utterance marked in red back to the user so she can reconsider what she stated.

In extending the procedure, TellMe handles incompleteness in the specification in the instruction of an individual step. For example, for the utterance "descend", TellMe will assume that the user meant to specify "descend from a position to another position with a specified altitude." Even though those objects are not mentioned as part of the step, the system will add them as entities relevant to the procedure. These objects will be used in the module that we describe next.

The TellMe Creator also handles collections of objects by introducing iterations when an action input is limited to one object but the instruction refers to a set. This is how the utterance "record videos and send them" is handled.

TellMe Unifier: Tightening the Procedure

Instructions also tend to be incomplete in that they do not express that the same object is relevant to several steps. Consider the following utterance: "Find oil spill, descend, and take a picture". This utterance does not say that the position where the oil spill is found is the same position where the descent starts. It also does not say that the position where the descent ends is the same position where the picture should be taken. So this instruction may be interpreted as each of those steps referring to different positions. TellMe uses a tightening heuristic that suggests that objects of the same type tend to be the same across steps unless otherwise indicated in the instruction. As any heuristic, this is not always the correct assumption and we need to give the user a way to indicate what the correct correspondence between objects is. Therefore, the TellMe Unifier makes as many mappings between objects as are compatible given the current procedure hypothesis, and will make them all possible interpretation options to the user. It will use the tightening heuristic to rank these options, presenting to the user the option where all objects are the same across steps as the most likely one. In cases when the heuristic is not correct, the user simply selects another option among those presented. This way, the system exploits an effective heuristic to make assumptions about how to complete the instruction while making it easy for the user to override those assumptions.

TellMe Elaborator: Elaborating the Procedure

The TellMe Elaborator is applied for efficiency considerations only when the user concludes the instruction, indicated with utterances such as "that's it" or "the end". It uses additional knowledge about the steps in order to check the validity of a procedure hypothesis. Recall that the TellMe Creator already does some validation by using the types of the inputs and outputs of steps to validate the links. The TellMe Elaborator applies other constraints in the steps and propagates them through the procedure as it is defined. For example, suppose the instruction states "descend to a position of altitude 200." The previous modules will create a step where the output is a position whose has-altitude property has the value 200. A rule about the descend step will be used by the TellMe Elaborator to set the input parameter to 200 in order to obtain that desired output or effect. The TellMe Elaborator applies the constraint propagation algorithms described in [Gil et al 10]. Those additional findings were not specified in the original instruction, but are important to make the procedure executable. The TellMe Elaborator will add them as constraints to the procedure.

CAN NON-PROGRAMMERS USE TELLME?

To evaluate our approach, we conducted initial formative user studies. The goal was to collect feedback on the overall approach and to find out whether there were any major barriers for users to communicate procedural knowledge with our interface.

We tested six subjects with ages ranging from 11 to 55. None of the subjects had programming experience, except one of the younger ones who had used Scratch [Resnick et al 09]. Each subject spent between 15 minutes and 1 hour using the system in total.

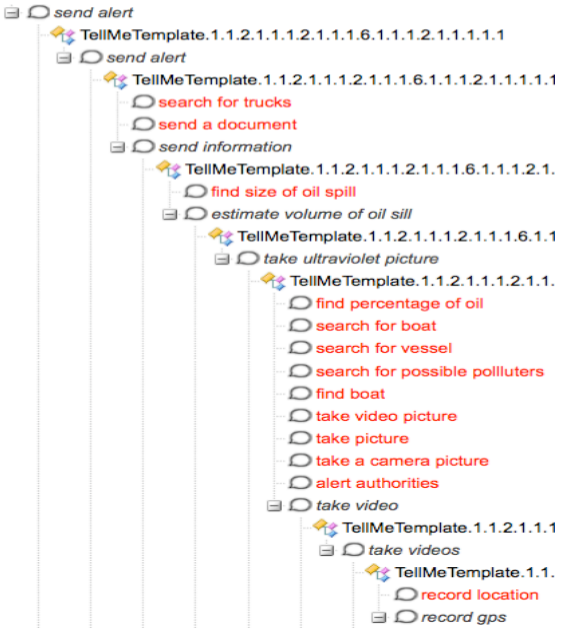
We wanted to avoid giving the subjects extensive training on our system. If our goal is to show that it is natural to give this kind of instruction, then any substantial training would defeat that purpose. The instructions to the subjects were limited to one page that described the different features and buttons of the TellMe interface, and are available on our web site[1].

We also wanted to avoid giving them descriptions of what procedures to teach because then they would likely just utter to the system the procedure in the way we would describe it to them. Therefore, we choose a Web site not developed by us that describes activities that are carried out by pilots that detect oil spills that pollute the sea off the coast of Belgium, without describing explicitly the procedures to be followed[2]. We coded basic actions and object types, as well as the paraphrases based on the text that appears on that site. We asked the subjects to begin by looking at the actions that were already defined in the system. We then asked them to think of different kinds of procedures that they would think are reasonable for a pilot to carry out, and to teach them to the system. As a result, the procedures that different subjects created were unique and not comparable. There was wide variation on the size and nature of the procedures that were designed by the users themselves. For example, some procedures record images at different heights, while others focus on sending alerts and as early as possible. We noted that some subjects redesigned the procedure as they went along rather than beforehand, so in those cases they had to adjust portions of the procedure that they had previously created, which they were able to do.

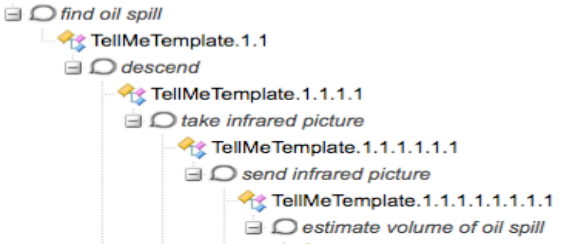
Our first goal was to check whether users would be able to use TellMe to specify procedures using natural language. For most subjects, the first few utterances were hard to get right, but after a few tries they learned to express what they wanted. This kind of interaction was typical across subjects. We also noted that the interaction with the command line helps the user construct interactively utterances that are already conformant to paraphrase patterns that the system understands. Here is an example of an initial interaction of one of our subjects, with the utterances that were rejected by the system are marked in red:

[1] http://www.isi.edu/ikcap/tellme/instructions.html

[2] http://www.mumm.ac.be/EN/Monitoring/Aircraft/methods.php

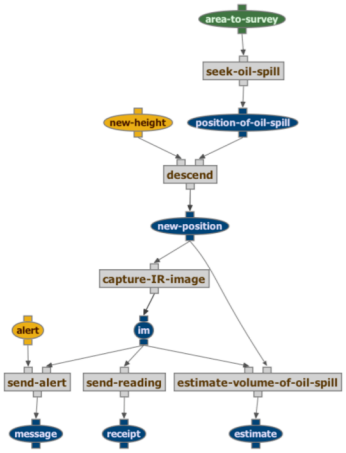


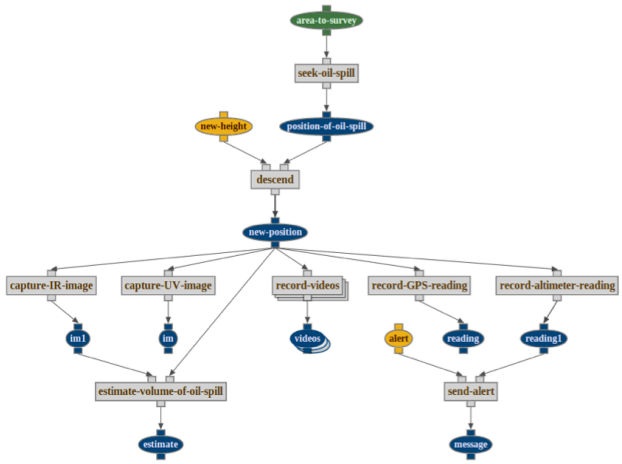
The subject was learning about what is possible to say, and what commands were understood by the system. After a few minutes, the same subject articulated the following instruction:



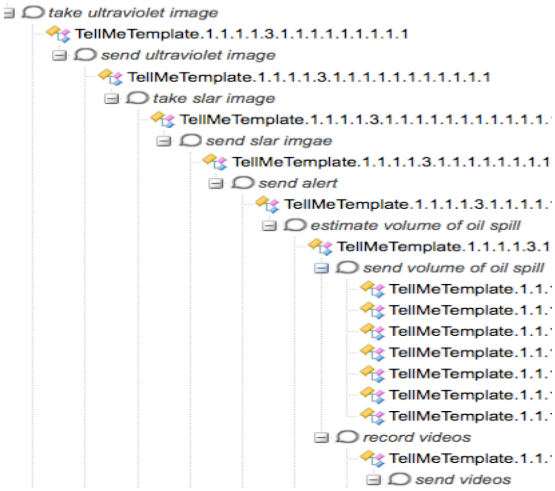
where all the utterances were understood by the system.

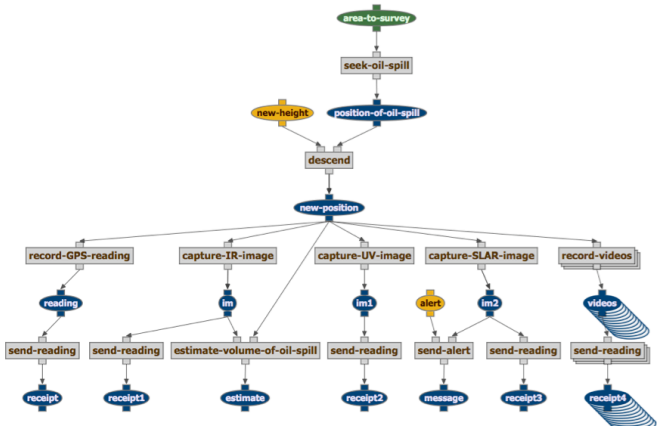
None of the procedures created was a linear sequence, rather all procedures had an interesting structure with some parallel steps and with multiple step dependencies. The shortest procedure created by any subject had 4 steps. Below are two examples of the procedures that were created with TellMe by two different subjects:

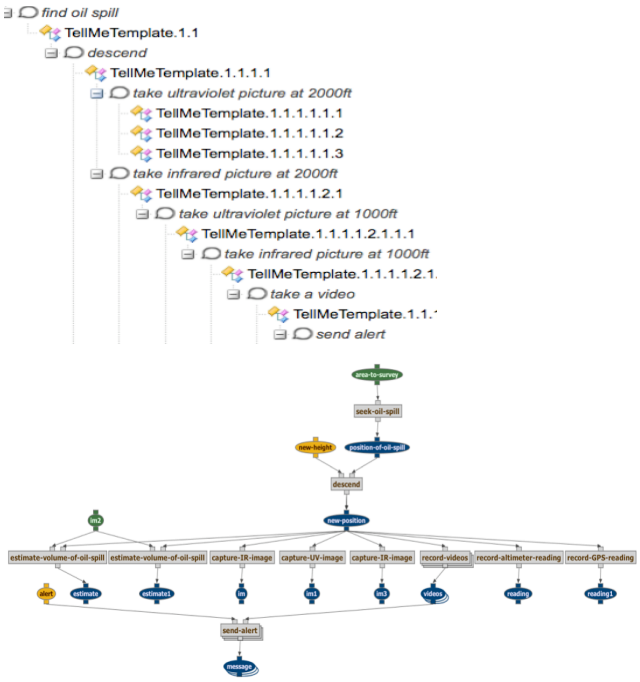




Particularly interesting was the ease with which users created iterations that were implicit in the instruction but were not directly expressed. Below are two examples of instructions followed by the procedure learned and entered by two different subjects:







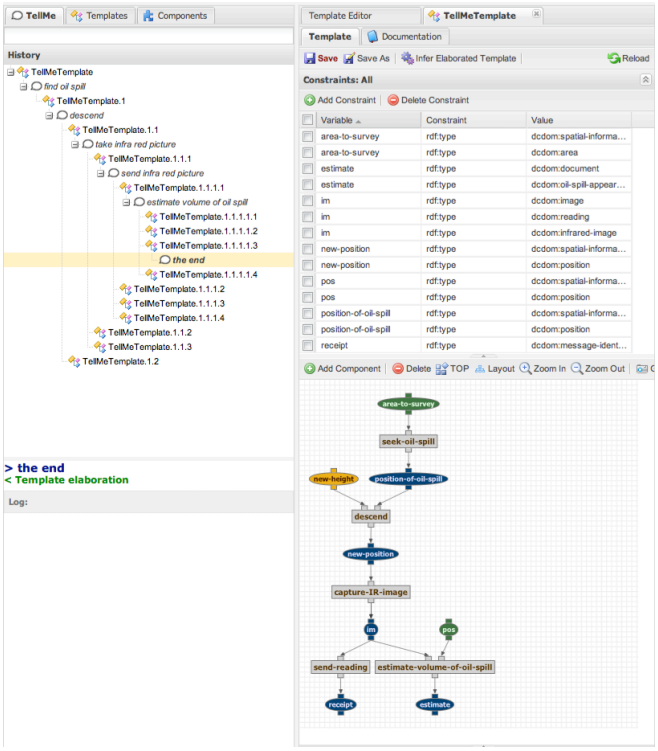
In the first one, "record videos" is followed by "send videos". Since the send action can only send one document at a time, the system created an iteration where for each video there is a send action. In the second one, "take a video" followed by "send alert", where there is no plural, resulted in an iteration where an alert is sent for each video.

All subjects were able to make most of their instructions understood. The table below shows the number of utterances entered by several subjects for one of their procedures, and the subset of those utterances that was accepted by the system:

Subject	Total utterances	Accepted
S1	9	6
S2	15	14
S3	39	17
S4	3	3
S5 (A)	41	19
S5 (B)	15	13
S6	27	10

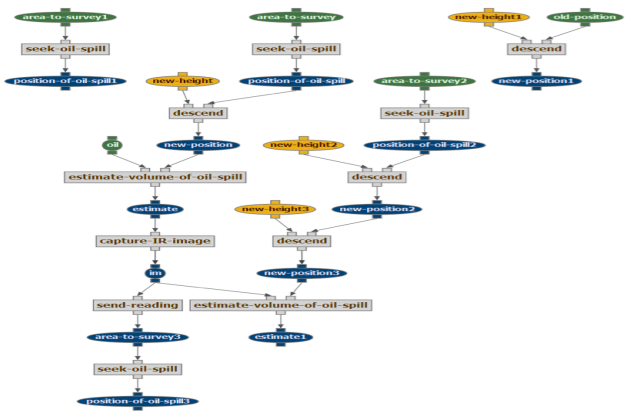
There is wide variability across subjects. There is also great improvement as the user becomes more familiar with what can be expressed, as shown by the two procedures shown for subject S5.

We also wondered if users would be able to interact with the system to correct the system's chosen interpretation when several are possible. In the following example of instruction, the subject inspected three alternative options and selected the third one, which represented what they had intended to convey:



Also visible in that screen shot are the constraints that TellMe learns about the objects in the procedure, together with the topology of the procedure itself. These constraints are a significant part of what is being learned by the system.

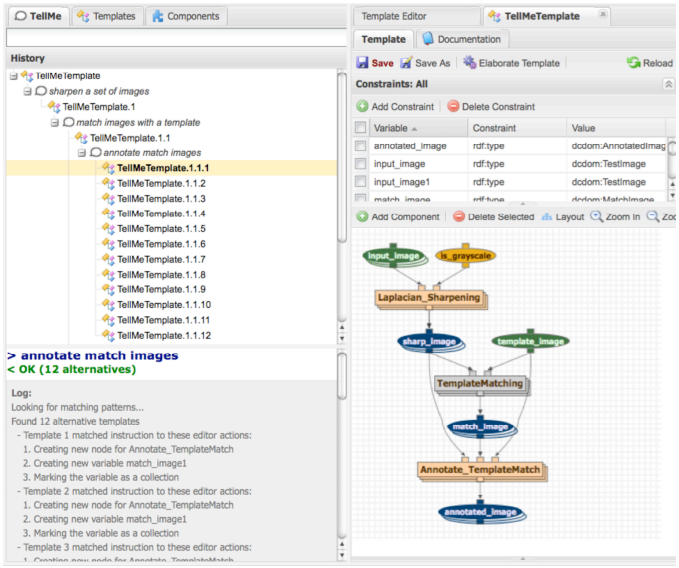
The majority of subjects understood the procedures as the interface was showing them. The dataflow across the steps was not initially intuitive, but after seeing a few examples they would create a procedure that had all the steps interconnected. Some users never stumbled across such examples, so their procedures contained steps that were not integrated with the rest. Below is an example:



We believe that this issue can be understood by users better but that we need to ensure that they are aware of the dataflow connections among steps. The tightening heuristic helped significantly in exposing subjects to these dataflow links. In some cases, subjects left inputs of steps unconnected to the outputs of other steps, in effect making

them inputs of the overall procedure which was not intended. We believe a good remedy for this issue is that when the user is finished specifying the procedure, TellMe should explicitly mention what the current inputs of the procedure are, and have the user correct that if needed.

Below is a screenshot showing learning an image processing workflow that illustrates the generality of TellMe:



CONCLUSION

We have described an approach to learn procedures through tutorial instruction using a natural interaction with a human teacher. Our initial steps to creating this capability have resulted in an interface that non-programmers were able to use to specify procedures of reasonable complexity. Many challenges remain, including handling incorrect instruction, using more complex procedure representations, and managing large numbers of procedure hypotheses.

ACKNOWLEDGEMENTS

We would like to thank Paul Groth for many valuable discussions on this work. This research was funded in part by the Defense Advanced Research Projects Agency under grant HR0011-07-C-0060, and in part by the National Science Foundation under grant CCF-0725332.

REFERENCES

1. Bugmann, G., Lauria, S., Kyriacou, T., Klein, E., Bos, J., and K. Coventry. "Using Verbal Instructions for Route Learning: Instruction Analysis." *Proceedings of the Conference Towards Autonomous Robots (TIMR)*, Manchester, UK, 2001.
2. Castelli, V., Bergman, L.D., Lau, T., and Oblinger, D. "Sheepdog, parallel collaborative programming-by-demonstration." *Knowledge-Based Systems*, 23(2): 94-109, 2010.
3. Chen, J. and Weld, D.S. "Recovering from errors during programming by demonstration". *Proceedings of the ACM International Conference on Intelligent User Interfaces (IUI)*, Canary Islands, Spain, January 2008.
4. Clark, P., Thompson, J., Barker, K., Porter, B., Chaudhri, V., Rodriguez, A., Thomere, J., Mishra, S., Gil, Y., Hayes, P., Reichherzer, T. "Knowledge Entry as the Graphical Assembly of Components," *Proceedings of the International Conference on Knowledge Capture (K-CAP)*, 2001.
5. Fritz, C. and Gil, Y. "A Formal Framework for Combining Natural Instruction and Demonstration for End-User Programming." *Proceedings of the ACM International Conference on Intelligent User Interfaces (IUI),* Palo Alto, CA, February 2011.
6. Gil, Y. and V. Ratnakar. "Automating To-Do Lists for Users: Interpretation of To-Dos for Selecting and Tasking Agents." *Proceedings of the Twenty-Third Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*, Chicago, IL, July 2008.
7. Gil, Y., Gonzalez-Calero, P., Kim, J., Moody, J. and V. Ratnakar. "A Semantic Framework for Automatic Generation of Computational Workflows Using Distributed Data and Component Catalogs." To appear in *Journal of Experimental and Theoretical Artificial Intelligence*, 2010.
8. Gil, Y., "Human Tutorial Instruction in the Raw." Submitted for publication, 2010. Available from http://www.isi.edu/~gil/gil-ker10.pdf.
9. Groth, P. and Y. Gil. "A Scientific Workflow Construction Command Line." *Proceedings of the ACM International Conference on Intelligent User Interfaces (IUI)*, Sanibel, FL, February 2009.
10. Huffman, S. and J. Laird. "Flexibly Instructable Agents." *Journal of Artificial Intelligence Research*, 3, 1995.
11. Ko, A.J., Myers, B.A., and Chau, D.H. "A Linguistic Analysis of How People Describe Software Problems." *Proceedings of the Visual Languages and Human-Centric Computing Conference (VL/HCC)*, 2006.
12. Li, I., Nichols, J., Lau, T.A., Drews, C., and Cypher, A. "Here's what I did: Sharing and reusing web activity with ActionShot." *Proceedings of the 28th International Conference on Human Factors in Computing Systems (CHI)*, 2010.
13. Nardi, B. "A Small Matter of Programming." MIT Press, 1993.
14. Resnick, M., J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, Y. Kafai. "Scratch: Programming for All." *Communications of the ACM*, 11, 2009.
15. Thomaz, A. L. and C. Breazeal. "Teachable robots: Understanding human teaching behavior to build more effective robot learners." *Artificial Intelligence*, 2008.
16. Webber, B. L. Webber, N. I. Badler, B. Di Eugenio, C. W. Geib, L. Levison, M. Moore. "Instructions, Intentions and Expectations." *Artificial Intelligence*, 73(1-2), 1995.