Task Learning by Instruction: Benefits and Challenges for Intelligent Interactive Systems

Jim Blythe, Prateek Tandon and Mandar Tillu

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

Abstract

The CALO desktop assistant aims to provide assistance through many AI technologies, including several techniques for learning to perform tasks. Based on our experiences implementing Tailor, a tool for task learning by instruction (TLI) in Calo, we explore the requirements for integrating TLI more closely into the assistant. The benefits of integration include a more coherent user experience and more powerful combined approaches for learning and interacting with a task-based assistant. We consider the TLI component both as one of the components for communicating with the user about tasks, and as a component for learning procedure knowledge. Components in both these groups need to share significant capabilities for task generation and recognition and for processing information from the user about tasks to be learned. We discuss strategies for invoking tasks by name that can also be used in task learning, enabling an integration of task learning by instruction and by demonstration.

Introduction

Calo's intelligent desktop assistant integrates a number of useful office tools, including a calendar, email and a web browser, with a general task execution engine and a variety of tools for machine learning. The task execution engine, SPARK (Morley & Myers 04), is able to automate many tasks that might be performed by the user through a coordinated set of steps, provided it has a process definition. For example, arranging travel might involve finding suitable hotels and flights on the web and initiating the appropriate travel requests within the user's organization.

In the Tailor project we are investigating task learning by instruction (TLI) as part of the Calo intelligent desktop assistant, integrated with the SPARK execution engine with the aim of enabling an agent to learn how to perform or improve its task performance through instructions from the user. Within Calo, learning by instruction is one of several different task learning capabilities under development, and one aim is to integrate the learning approaches to exploit their respective strengths and overcome their weaknesses.

In this paper we review the status of TLI in Calo, and explore how this and related capabilities could be most productively integrated for a coherent user experience with the Calo desktop system. We consider the capability from two viewpoints. First, we consider TLI as one of a related set of capabilities for interacting with an intelligent task-performing agent through language. These capabilities include the potential use of a controlled language for tasking, utterance understanding and disambiguation, dialog for resolving tasking and during execution, and explanation of the state of task execution in addition to learning by instruction. These capabilities need to be aligned for the user to have a coherent interaction with an intelligent assistant.

Second, we consider TLI as one of a related set of task-learning capabilities, including learning by demonstration (possibly annotated by text or another means) and more passive observation of actions. While in some cases these represent alternative ways for a system to learn about tasks to be performed, more often these capabilities are complementary. In some situations, one learning technique is more appropriate than others, while in other situations several techniques may be combined to learn tasks more quickly, and/or with fewer errors. As before, we analyze the benefits of tight or loose integration and any requirements to allow several modes of task learning to interact. Finally we explore ways to specify tasks in the assistant that can be used both for invoking tasks and learning by instruction.

An example of task learning with Tailor

In the following example, the user wishes to create an agent behavior to find hotels within a certain distance of a meeting, to use while planning a trip. The new procedure should ask for or be passed the meeting and the distance

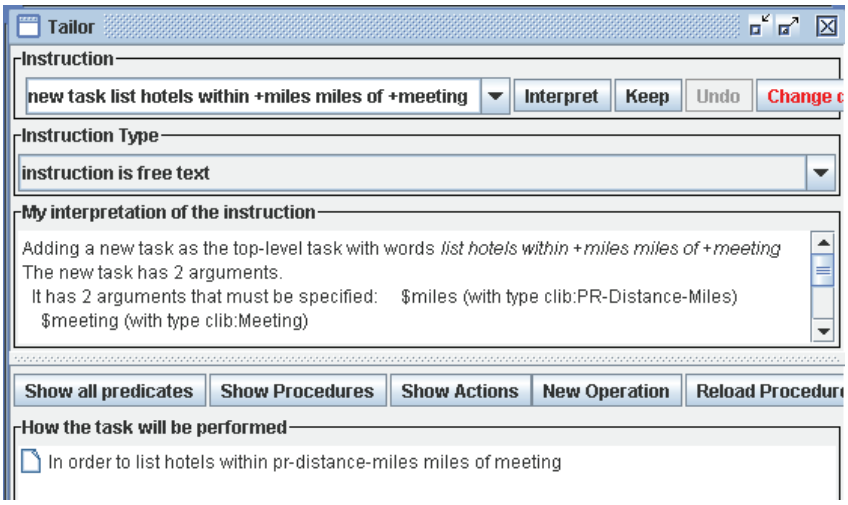

Figure 1. Tailor suggests input types for the initial procedure

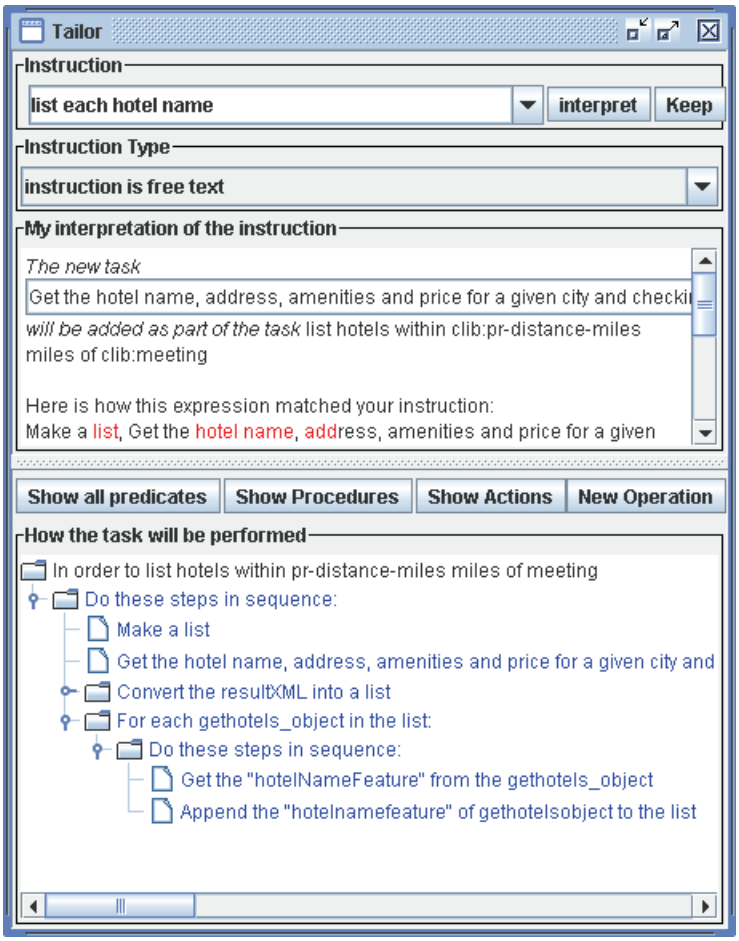

Figure 2. A match is suggested for the user instruction that contains several steps

threshold. We assume the agent already knows procedures to find hotels within a city given its name and state, and to find the distance between two locations given by a street address and zip code.

Figure 1 shows Tailor's initial screen. The user has indicated that the new procedure should accept a meeting and a distance in miles, and can be referred to as "list hotels within +miles miles of +meeting", where +miles is an input variable representing the distance and +meeting represents the meeting. Tailor matches the variable descriptions to types in the desktop ontology. This allows the procedure to be invoked with the appropriate

information by other components in the desktop. For example, the meeting can be specified via the Calendar.

In figure 2, Tailor has created the basic procedure and suggested an initial procedure body based on the user instruction "list each hotel name". This procedure uses an information extraction agent defined earlier to find available hotels based on the city and the meeting dates, the iterates over the hotels to build a list of their names.

This is Tailor's highest-ranked suggestion for the procedure body, since it is one of the smallest fragments of executable code that matches the user's request, but it is not the only match. The menu in the middle window shows alternatives, some for example that pick a single hotel from the list rather than show them all. In many cases the smallest matching fragment is a good choice, but Tailor relies on the user to pick their preferred match since there may be hard or soft constraints that are not reflected in the agent's world model. Sometimes, as here, there can be many matches of the same size, leading to a significant matching problem. We discuss this problem in more detail later.

The user adds two more instructions to use the distance as part of the procedure: "find the distance between the hotel and the meeting" and "add if the distance is less than +miles". The first instruction leads Tailor to include a primitive task that finds distances from the Yahoo web site, based on a street address and zip code for each location. Tailor automatically finds the required data for the meeting and for each hotel. The second instruction uses the distance in a threshold test.

The final procedure definition is shown in Figure 3, both as Tailor shows it to the user and in executable form. The procedure is also registered with the execution engine in executable form as shown on the right hand side.

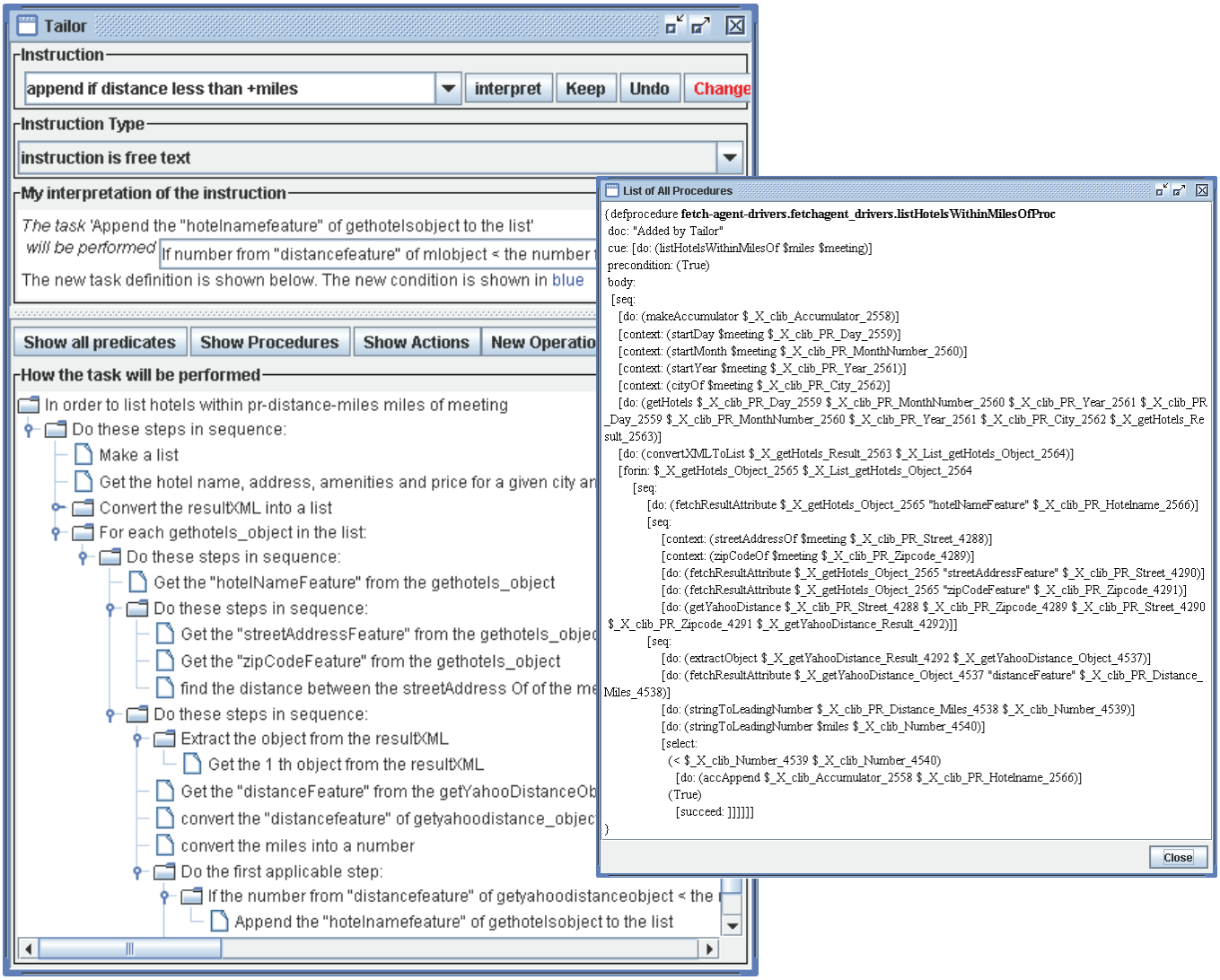

Figure 3. The final executable procedure after four instructions, with the executable version shown to the right.

The solution contains a number of steps that Tailor added while composing the procedures for finding hotels and computing distances to achieve the user's goals. Since these are both wrapper agents that return information from online sources encoded in XML, Tailor added steps to convert the information into a list of hotel objects that can be manipulated in a SPARK procedure, and added calls to additional steps to find the required information about each hotel. It also extracts the distance from the returned XML block and converts it to a number in order to compare it with the threshold. In this way Tailor helps the user refer to a growing set of available procedures by meaningful names, combine them appropriately and encode the procedure correctly in SPARK syntax.

In its current form, Tailor's interface lacks several features we desire in an intelligent assistant. We are currently working to reduce its reliance on the user for the correct types for input variables in a new task, improve its reasoning about processes for making good coding choices and giving salient warnings to the user, and to improve its response when it has difficulty understanding the instruction. In this paper we consider Tailor as a component technology in a larger intelligent assistant, and an important one since no other learning approach can currently produce such complex procedure definitions correctly from a small number of interactions without using instruction. Based on our experiences with Tailor, we discuss how task learning by instruction (TLI) can be integrated with other capabilities to provide the assistant with the ability to adapt its procedure knowledge.

Task learning by instruction as a component method of task learning

Within Calo, a number of different methods for task learning are under development. These include PrimTL, a method to learn information access procedures that are not inspectable by the execution engine (Lerman et al. 06), as well as task learning by demonstration, by discussion and by observation, which along with Tailor learn composite procedure descriptions. We can characterize the composite task learning components along two dimensions. First, they differ by the amount of user interaction required to learn a procedure. TLI is at one end of the spectrum, relying on direct information from the user for all its knowledge. Learning by discussion combines a demonstration of the task with direct information from the user indicating substeps, intermediate results and other information. Learning by demonstration alone requires significantly less user interaction than TLI, although the user must structure the demonstration, indicate when the procedure is beginning and when it has been achieved. Learning by observation requires the least amount of user intervention, relying on observed data from executions of the procedures.

Task learning methods can also be distinguished by the nature of the information that is passed. Tailor in its current form uses no examples and relies on abstract information about how the task is performed. Pure learning by observation would use no information other than traces of executions. The other systems use a mix of examples and more general information.

Both dimensions impact the cases where the different learning tools may be appropriate. If the task being performed is not easily articulated by the user, such as UI manipulation, then TLI is probably too intrusive and may also lead to inaccurate procedure knowledge. In cases where users can more easily talk about tasks than perform them, however, such as purchasing equipment, TLI may be faster and ultimately less obtrusive. In many cases, though, more than one learning technique can be applied, and they may be complementary. Learning by demonstration may quickly pick up the essential actions and structure of a task, while one or two instructions may correct slight errors in generalization.

In most cases, several different task learning techniques will be combined when the task is learned as efficiently and unobtrusively as possible. It is not hard to imagine cases where the most natural way for users to present information changes several times during the same task. In calendar-related tasks, for example, a learning episode might begin when the user rejects a suggested time slot for a meeting through the GUI, and re-assigns the meeting with a gesture. However the proper generalization of a new procedure describing the task may rely on semantic information about the meeting type and participants that can only efficiently be described through text. Finding the best interpretation of the text, however, may require using information from traces of previous similar meetings to weight the alternative interpretations correctly.

To provide a coherent user experience while learning tasks, the assistant needs to be very agile in switching from one learning approach to another while presenting a unified interface to the user. This implies that the learning systems in Calo should ultimately share a good deal of their interface, and share information captured in gestures, in traces or through verbal interaction as well as sharing intermediate hypotheses about the tasks being learned. While the learning techniques available in the Calo assistant have been developed independently, to our knowledge this platform provides the best opportunity to explore this agile use of different learning methodologies, since the learning tools share a common platform and target language.

Such cases also highlight the need for textual representations for even purely graphical gestures made while using the assistant, since at some point these actions may need to be described while discussing a procedure at a more general level. Most actions taken within a GUI have interpretations at both a screen level (click here) and a task

level (attach the file to the email message) and both may be appropriate at different times.

Task learning by instruction as language-based interaction

The task learning capability in an interactive intelligent assistant needs to be fully integrated with its other capabilities to provide a seamless interactive experience for the user. We can view task learning by instruction as one of a set of related capabilities that allow the agent to communicate with the user about tasks through language. The user should therefore be able to refer to tasks in the agent in the same way when initiating them as when referring to tasks during TLI. This means the TLI and language-based tasking must have access to the same set of procedures as part of a global KB, and should provide the same search capabilities for understanding tasks. Similarly, the way that TLI describes learned procedures, and trees of intentions triggered by a goal, should be the same as the way the assistant refers to these procedures and intentions during other tasks such as explanation and dialog.

For example, one module that allows the user to task Calo through text is the Towel (Myers et al. 07), which uses a conversational style similar to Diamond Help (Rich et al. 05). This system accesses the same set of procedures as Tailor, but does not search for argument values based on the current problem-solving state in the same way. A coherent approach might also allow Towel to suggest composing a small number of steps and queries to solve a user goal that cannot be solved by a single procedure. Calo's explanation module is centralized across problem solving and inference (Pinheiro et al. 05), and can also incorporate knowledge of procedure modifications from Tailor into an explanation, but at present the two modules use different routines to generate presentations.

An interesting consequence of a unified interface for describing tasks for both learning and invoking them is that it can sometimes reduce the distinction between learning by instruction and learning by demonstration (TLD). In learning by demonstration, the teacher signals the beginning and end of a demonstration and performs a sequence of actions that constitute executing an instance of the procedure. The learning system records the actions and subsequently generalizes them to broaden the applicability of the learned procedure. Typically this includes turning some constants into variables and may include relaxing the ordering constraints on substeps and introducing alternative execution paths based on run-time conditions. In a TLI system such as Tailor, the user describes modifications in general terms, including new steps or conditions on steps. The two approaches are similar for steps that are demonstrated using the language-based interface and are not associated with branches in execution. Here the main difference is the time and initiator of generalization – after demonstration and from the system with TLD, and during instruction and from the user with TLI. In general, steps are more easily demonstrated directly with the tool's UI and instructions may be more complex. The unified view can form the basis of flexible combinations of these approaches, however, in which the user can add steps to a procedure using either method, and steps can be generalized either by the system or the user at any time in the process. This is a different integration point from that of learning by discussion as used in PLOW, for example, where steps are introduced through demonstration and language is used for annotation (Jung et al. 06).

Combinations of choices in specifying actions

Since the same interface should be used for invoking actions through language in the intelligent assistant and adding steps in TLI, it is worth considering how efficient and natural this interface can be. This can be a challenge when the actions to be invoked or included may have a large number of arguments that have some structure. For example, consider the instruction "list each hotel name", that matches a block of steps to create a list, locate available hotels given a city and requested dates, and add the name of each hotel to the list. The step to find hotels uses a wrapper for information extraction from the web, that requires a city name and start and end dates for the visit. The start and end dates are each represented by three parameters, for day, month and year, so the wrapper has seven inputs in all.

The system must find values for these seven inputs with no hints from the user. Searching through known inputs for objects of the correct type, Tailor finds a city name given as input, and the beginning and end dates of the meeting. However, Tailor does not currently know that the first day, month and year arguments of the wrapper should correspond to one date and the second set should correspond to another date. Thus it can find 64 possible sets of parameters for this task, even with only two dates and one city name.

Our recent prototype created a menu with these 64 items, which is unacceptable to the user. In this case, simply asking the user to specify each argument in turn from a menu of possible values found by search is a more reasonable option, requiring six separate decisions by the user in this case. However this is still a tedious process, and it should not be necessary – there are only three different inputs to be specified. Alternative approaches rely on the structure between the six date-related arguments. This structure might (1) be represented by the developer of the wrapper, (2) discovered by analyzing previous instances of the wrapper or (3) guessed by heuristics that exploit locality.

In the first case, the user must tell the system how the arguments are related using a constraint language. The

PrimTL group are currently investigating composite data-types that can be learned from data that have several components, such as the date in this case. When this is available, the learned wrapper would have only three arguments, and we would find only four possible values in this example. A further constraint could be added by the user to specify that the first date, when hotel check-in occurs, should come before the second date, for check-out. This reduces the possible matches to one, given two dates. These relations could also be inferred from several instances of the step if data is available. In the third case, we might use heuristics to prefer sets of arguments that use similar ways to find values for variables that are adjacent in the wrapper specification. This heuristic can prefer the wrong solutions in some cases, but it requires neither examples nor complex constraints, and eliminates most of the unhelpful matches in many cases. Each of these approaches can be combined with incremental specification of the parameters by the user, so that as the user chooses values for some parameters, choices for remaining parameters are restricted to the consistent ones. We are currently exploring these alternatives in Tailor.

Conclusions

In this paper we have explored how task learning by instruction (TLI) should best be integrated in an intelligent assistant from two points of view: by viewing TLI as one of the components that communicates about tasks verbally and as one of the components that learns procedure knowledge. In both cases we argue that the components should share UI and reasoning modules in order to (1) present a seamless interface to the user, (2) to improve the scope of some of the components and (3) to allow agile integration of learning components. We are beginning to investigate a deeper integration between Tailor and the other task learning components in Calo, and to explore some of the capabilities of the integrated learning system. Tailor already learns procedures that make use of information integration procedures learned by PrimTL, and we have investigated a loose integration with Lapdog, a tool for learning by demonstration in Calo.

We also noted how a deeper integration places demands on the central procedure knowledge base used in Calo, for example that text generation and recognition knowledge be available to all components. The need to learn procedures for tasks that cross modalities within the assistant lead to a recommendation that even low-level graphical tasks should be representable with a verbal description. Within Tailor we are also working on using analogy to help Tailor's recognition and guidance of the user based on previous modifications and similar procedures.

Acknowledgments

We are grateful for discussions with the entire task learning group in the Calo project on integration, and in particular Craig Knoblock, Kristina Lerman, Dipsy Kapoor and Tom Russ. 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, J. An Analysis of Task Learning by Instruction, *Proc. International Joint Conference on Artificial Intelligence*, 2005a.

Blythe, J. Task Learning by Instruction in Tailor, *Proc. Intelligent User Interfaces*, 2005b

Jung, H., Allen, J., Chambers, N., Galescu, L., Swift, M and Tyson, W., One-Shot Procedure Learning from Instruction and Observation, *Proc. FLAIRS* 2006

Lerman, K., Plangrasopchok, A. and Knoblock, C., Automatically Labeling the Inputs and Outputs of Web Services, *Proc. American Association for Artificial Intelligence*, 2006.

Morley, D. and Myers, K., The SPARK Agent Framework, *Proc. Autonomous Agents and Multi-agent Systems*, 2004

Myers, K. et al., An Intelligent Personal Assistant for Task and Time Management, *AI Magazine*, forthcoming 2007.

Pinheiro da Silva, P., Hayes, P., McGuinness, D., Fikes, R. and Deshwal, P., *Explaining Problem Solver Answers*, Technical Report KSL-05-02, Stanford University.

Rich, C., Sidner, C., Lesh, N., Garland, A., Booth, S. and Chimani, M., DiamondHelp: A Collaborative Interface Framework for Networked Home Appliances, *Proc. American Association for Artificial Intelligence*, 2005