Going beyond PBD: A Play-by-Play and Mixed-initiative Approach

Hyuckchul Jung, James Allen, William de Beaumont, Nate Blaylock
[†]George Ferguson, Lucian Galescu, [†]Mary Swift

Institute for Human and Machine Cognition
40 South Alcaniz Street, Pensacola, FL
{hjung, jallen, wbeaumont, blaylock, lgalescu}@ihmc.us

[†]Computer Science Dept., Univ. of Rochester
PO Box 270226, Rochester, NY
{ferguson, swift}@cs.rochester.edu

ABSTRACT
An innovative task learning system called PLOW (Procedure Learning On the Web) lets end-users teach procedural tasks to automate their various web activities. Deep natural understanding and mixed-initiative interaction in PLOW makes the teaching process very natural and intuitive while producing efficient/workable procedures.

INTRODUCTION
The web has become the main medium for providing services and information for our daily activities at home or work. Many web activities involve the execution of a series of procedural steps involving Web-browser actions. Programmatically automating such tasks to increase productivity is feasible but out of reach for many end users.

Programming-by-demonstration (PBD) is an innovative paradigm that can enable novice users to build a program by just showing a computer what a user intends to do [3]. However, in this approach, numerous examples are often needed for the system to infer a workable task.

We aim to build a system with which a novice user can teach tasks by using a single example without requiring too much or too specialized work from the novice user. This goal poses significant challenges because the observed sequence of actions is only one instance of a task to teach and the user's decision-making process that drives his/her actions is not revealed in the demonstration.

To achieve this challenging goal, we have developed a novel approach in which a user not only demonstrates a task but also explains the task with a play-by-play description. In the PLOW system, demonstration is accompanied by natural language (NL) explanation, which provides the system with useful information to identify the following key aspects in building complex task models: (i) the goal of the task; (ii) the hierarchical structure including the boundary of (nested) iterations; (iii) parameter identification; and (iv) control constructs and conditions.

The power of NL makes it possible for PLOW to infer a task structure that is not easily inferable from observations alone but represents what the user intended. Furthermore, the semantic information encoded in NL enables PLOW to reliably identify objects in dynamic HTML files.

Another key aspect that makes PLOW more efficient is the mixed-initiative interaction that dramatically reduces the complexity of teaching a task by proactively initiating execution for verification and asking timely questions to solicit necessary information to build the task. In this position paper, we briefly introduce and discuss the challenges, innovations and lessons in developing the PLOW system. Refer to [1, 2, 5] for detailed information.

PLOW ARCHITECTURE & INTERFACE
PLOW is an extension to TRIPS [4], a dialogue-based collaborative problem solving system that has been applied to many real world applications. The core components of TRIPS include a speech recognition system, a robust parsing system, an interpretation manager (IM), an ontology manager (OM), and a surface generator. In PLOW, TRIPS supports deep natural language understanding and dialogue management.

Figure 1 shows a high-level view of the PLOW system. At the center lies a CPS (Collaborative Problem Solving) agent that computes the most likely intended intention in the given problem-solving context (based on the interaction with IM in TRIPS). CPS also coordinates and drives other parts of the system to learn what a user intends to build as a task and invoke execution when needed.

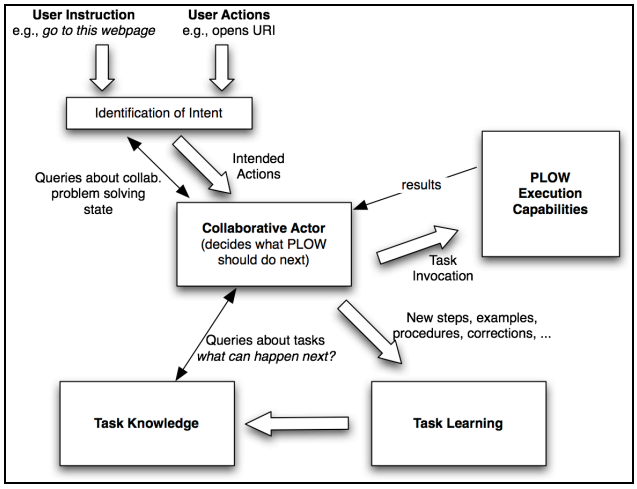


Figure 1: PLOW Architecture

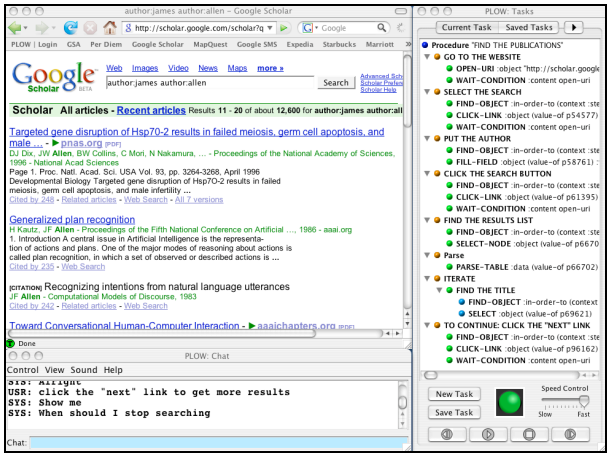


Figure 2: The PLOW Interface

While the core reasoning modules of PLOW are domain/application-independent, PLOW focuses on tasks that can be performed within a web browser. Figure 2 shows PLOW's user interface. The main window on the left is the Firefox browser instrumented so that PLOW can monitor user actions and execute actions for learned tasks. On the right is a GUI that summarizes a task under construction, highlights steps in execution for verification, and provides tools to manage learned tasks. A chat window at the bottom shows speech interaction and the user can switch between speech and keyboard anytime.

TEACHING WEB TASKS WITH PLAY-BY-PLAY

Primitive Actions and Dynamic Web Objects

Consider a case in which a user is teaching PLOW how to find hotels near an address. At one step, a user would say, "*Put the city here*" and type a city name into a text field named "City". Here, the observed action from the browser instrumentation is an action that fills a text (e.g., "Pensacola") into a text field. However, the result of language understanding provides PLOW useful information that can be used to generalize the observed action instance. First, the user description gives a semantic concept for the object in the current action (i.e., *CITY). The semantic information is useful in generalizing the action. For instance, if a full address was given earlier and the typed city name matches a city part of the full address, PLOW can encode their dependency relation into the current task.

Second, the semantic description helps PLOW to find dynamic web objects in future execution. In PLOW, each web page is logically represented by its DOM (Document Object Model), a tree-structured object model of the screen content, and the instrumentation accesses and manipulates the underlying DOM nodes. In demonstration, from the DOM tree structure of a web page, PLOW (i) finds a reference node with a text that is closest to the semantic concept of the action object or its ontological/linguistic variation (e.g., town, municipality, etc.) and (ii) computes the relation between the reference node and the node for the object accessed in the observed action.

In execution, PLOW looks for a reference node based on the method in (i) and finds the node for the object to be acted upon with the relation computed in (ii). Therefore, even when there is a web page format change, this approach enables PLOW to find a node as long as there's no significant local change around the node in focus. Refer to [2] for detail.

Task Hierarchy

PLOW uses simple heuristics to identify the beginning/end of a sub task. Any statement that explicitly identifies a goal (e.g., "*Let me show you how* …") is seen as the beginning of a new (sub) task. User's explicit statement such as "*I'm done*" or another goal statement indicates the end of the current (sub) task. Our anecdotal experience is that users easily get familiar with this intuitive teaching style.

Parameter Identification

Identifying parameters is challenging even for a simple task and it is almost impossible with only a single observation without special domain knowledge. When an object is used in a task, the system should determine if it is a constant or a variable. In the case of a variable, it also has to figure out the relation between variables. Figure 3 shows how natural language plays a critical role in PLOW's parameter identification, enabling it to identify parameters from a play-by-play single demonstration. Furthermore, TRIPS' reference resolution capability also identifies the relation between parameters: e.g., after saying, "*Select a check-in date here*", if a user says "*Put the date*", the reference resolution considers both dates are equal.

Utterance (Action)	Interpretation	Key features
Let me show you how to find hotels near an address	hotels → output	- Bare plural - Object of an information producing action "find"
	an address → input	- Indefinite - No decision action
Put hotels (Type "hotels")	Hotels → constant	- Bare plural - Identical to the typed text in the action
Put the zipcode (Type "32502")	a zipcode → related to the address input	- Definite - Zipcode is a role of an address in Ontology

Figure 3: Interpretation of Noun Phrases

Control Constructs

Conditionals

Conditionals have a basic structure of 'if X, then do Y', optionally followed by 'otherwise do Z'. However, the action trace for conditionals includes only one action, either Y or Z, based on the truth-value of the condition X. In general, identifying X is very difficult, since the entire

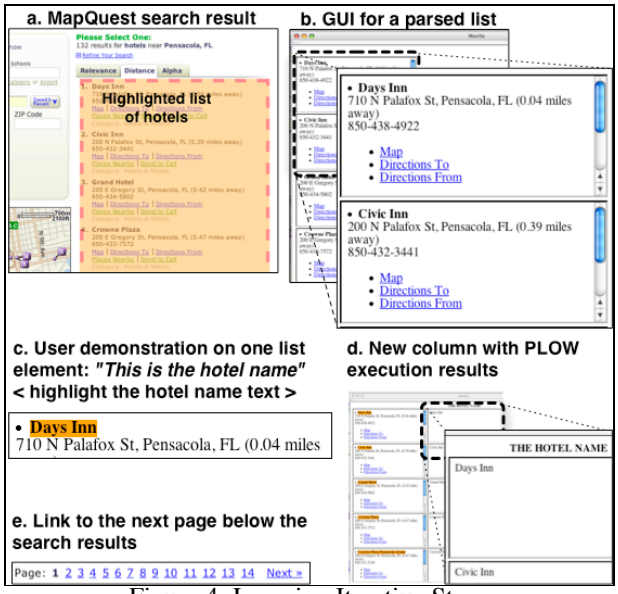


Figure 4: Learning Iterative Steps

context of demonstration should be checked and reasoned about. However, in the play-by-play demonstration, when a user specifies a condition, PLOW can interpret correctly the condition from language.

Iteration

The main difficulty in identifying iterative procedures from a single example is that the action trace (a sequence of actions) alone does not fully reveal the iterative structure. For iteration, a system needs to identify these key aspects: (i) the list to iterate over; (ii) what actions to take for each element; (iii) how to add more list elements; and (iv) when to stop. For a system to reason about these aspects on its own, in addition to repetitive examples, full understanding of the action context (beyond observed actions) and special domain knowledge will be required (e.g., what and how many list items were potentially available, which ones were included in the observed actions, how and when web page transition works, etc.). Furthermore, a user would not want to demonstrate lengthy iterations. In PLOW, natural language again plays a key role. As shown below, we designed the system GUI and dialogue to guide a user through the demonstration for iteration: mixed-initiative interaction with proactive execution and simple queries makes the process much easier and intuitive.

In Figure 4, a user is teaching PLOW how to find hotels near an address. When the user highlights a list of results (Figure 4a) and says, "*Here is a list of results*", PLOW infers that an iteration over elements in the list will follow. Then, PLOW enters into an iteration-learning mode with the goal of identifying the key aspects stated above. First, by analyzing the DOM structure for the list object, PLOW identifies individual elements of the list and then presents the parsed list in a dedicated GUI window with each element (essentially a portion of the original web page) contained in a separate cell (Figure 4b). This GUI-based approach lets the user quickly verify the list parsing result and easily teach what to do for each element. Note that list and table HTML objects that contain the desired list may also be used for other purposes (e.g., formatting, inserting ads, etc.), so it is fairly common that some irrelevant information may appear to be part of the list; PLOW uses clustering and similarity based techniques to weed out such information.

After presenting the parsed list, PLOW waits for user's demonstration for an element. For instance, the user says, "*This is the hotel name*", and highlights the hotel name in one of small cells in the GUI (Figure 4c). Given this information, PLOW learns the extraction pattern and *proactively* applies the rule to the rest of elements (Figure 4d). Note that a composite action (e.g., navigating to a page from a link, extracting data from the new page and so on) can be also defined for each element. If there is an error, the user can notify PLOW with the problem by saying, "*This is wrong*", and show a new example. Then, PLOW learns a new extraction pattern and reapplies it to all list elements for further verification. This correction interaction may continue until a comprehensive pattern is learned.

Next, the user teaches PLOW how to iterate over multiple lists by introducing a special action (e.g., "*Click the next link for more results*" – see Figure 4e). This helps PLOW to recognize the user's intention to repeat what he/she demonstrated in the first list on other lists. Here, to identify the duration of the iteration, PLOW asks for a termination condition by saying, "*When should I stop searching?*" For this query, it can understand a range of user responses such as "*Get two pages,*" "*Twenty items*", "*Get all*" and also conditions based on information extracted for each element, as in "*Until the distance is greater than 2 miles*". In the case of getting all results, the system also asks for how to recognize the ending, and the user can tell and show what to check (e.g., "*When you don't see the next link*" or "*When you see the end sign*"). For verification, PLOW executes the learned iterative procedure until the termination condition is satisfied and presents the results to the user using the special GUI. The user can sort and/or filter the results with certain conditions (e.g., "sort the results by distance", "keep the first three results", etc.).

UTILIZING & IMPROVING TAUGHT WEB TASKS

Persistent and Sharable Tasks

After teaching a task, a user can save the task into a persistent repository. Figure 5 shows the "Saved Tasks" panel in the PLOW interface that shows a list of a user's private tasks. A pop-up menu is provided for task

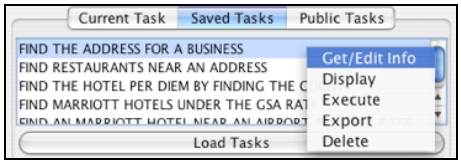


Figure 5: Task Management

management, and one of its capabilities is exporting a task to a public repository for sharing the task with others. A user can import shared tasks from the "Public Tasks" panel.

Task Invocation

Tasks in the private repository can be invoked through the GUI (Figure 5) or in natural language (e.g., "*Find me hotels near an airport*"). If the selected task requires input parameters, PLOW asks for their values (e.g., "*What is the airport?*"), and the user can provide parameter values using the GUI or natural language. Users can invoke a task and provide input parameters in a single utterance, e.g., "*Find me hotels near LAX*" or "*Find me hotels near an airport. The airport is LAX.*" Results can also be presented via the GUI or in natural language. This NL-based invocation capability allows users to use indirect channels, as well. For example, we built an email agent that interprets an email subject and body so that a user can invoke a task by sending an email and receive the execution results as a reply.

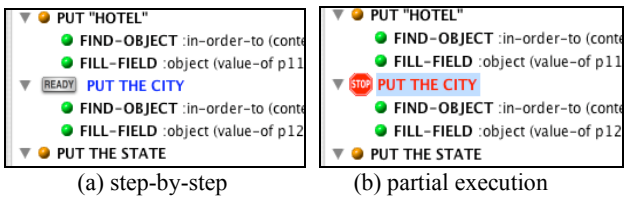
Here, given a user request, PLOW finds a matching task with its natural language understanding and ontological reasoning capabilities. A user does not necessarily have to use the same task description used in teaching. "*Get me restaurants in a city*" or "*Look for eatery in a town*" would select a task to find restaurants in a city.

Reusing Tasks

In teaching a task, existing tasks can be included as subtasks. When a user gives the description of a new step, PLOW checks if the step matches one of the known tasks; if a matching task is found, it is inserted as a subtask with parameter binding between the current task and the reused task. For instance, in one teaching session, a user has taught how to book a flight and wants to reserve a hotel. For a step introduced by saying, "*Book a hotel for the arrival date*", PLOW will check for a matching task for the step. If the user already has a task to reserve a hotel with a check-in date and a number of nights, PLOW will mark the step as reusing another task so that, in execution, the reused task can be called. PLOW will also infer that the arrival date should be bound to the check-in date and consider the number of nights as a new input parameter if there is no related object in the current task.

Editing Tasks

To fix obsolete tasks (e.g., to update them after web site changes) or to improve/simplify a task, PLOW lets a user add or delete steps. To reach a step to edit, PLOW supports (i) step-by-step execution (the default mode for verification) and (ii) partial execution up to a certain step. Figure 6 shows a GUI snapshot in which highlighted steps are the ones to be executed next. One can invoke the two modes by saying, "*Let's practice step by step*" and "*Execute the task up to this step*" (after clicking a step in the GUI) respectively. Setting up the action context (i.e., browser setting, extracted objects, available parameter values, etc.) with real execution is critical since the context is used in PLOW's reasoning for the action to edit.



(a) step-by-step (b) partial execution

Figure 6: Current Task Execution Status during Editing

Improving Tasks from Execution Failure

Execution failure from unnecessary or missing steps can be corrected by task editing. Major web site redesigns will sometimes trigger web object identification failures. When PLOW detects an execution error, it stops at the failed action, notifies the user and initiates a debugging process by asking for a new example from which it learns an additional extraction pattern.

EVALUATION

In 2006 and 2007, PLOW was evaluated along with other task building systems by an independent agency as a part of DARPA CALO project [6]. PLOW did very well in both tests, receiving a grade of 2.82 (2006) and 3.27 (2007) out of 4 (exceeding the project goals in both cases). Furthermore, PLOW was the system of choice among the subjects, and anecdotal comments from the subjects in 2007 were that PLOW's user convenience has significantly improved with more GUI/NL feedback in its interface.

CONCLUSION WITH BOOK CHAPTER SUGGESTION

PLOW demonstrates that NL is a powerful intuitive tool for end-users to build web tasks with significant complexity using only a single demonstration and the mixed-initiative interaction also makes the task building process much more convenient and intuitive. For the proposed book, we plan to provide more detailed description for PLOW, focusing on the NL aspects and the use of a GUI that provides intuitive views of the task structure during learning.

REFERENCES

1. James Allen et al., PLOW: A Collaborative Task Learning Agent, *Proceedings of the AAAI Conference on Artificial Intelligence: Special Track on Integrated Intelligence*, 2007
2. Nathanael Chambers et al., Using Semantics to Identify Web Objects, *Proceedings of the National Conference on Artificial Intelligence: Special Track on AI and the Web*, 2006
3. Allen Cypher, editor. Watch what I do: Programming by demonstration. MIT Press, Cambridge, MA, 1993
4. George Ferguson and James Allen, TRIPS: an integrated intelligent problem-solving assistant, *Proceedings of the National Conference on Artificial Intelligence*, 1998
5. Hyuckchul Jung et al., Utilizing Natural Language for One-Shot Task Learning, *Journal of Logic and Computation*, doi: 0.1093/logcom/exm071, Oxford University Press, 2007
6. DARPA CALO project: http://caloproject.sri.com