# 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 [6]. 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 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. This chapter presents the challenges, innovations and lessons in developing the PLOW system.

**MOTIVATING EXAMPLE**
Information extraction from the Web is a routine action for many users, and travel arrangement (e.g., booking hotels/flights/cars) is one of time-consuming activities that require collecting information from multiple resources. Figure 1 shows a sample dialogue in which a user teaches PLOW how to find close hotels near an address for his/her travel using a popular website such as mapquest.com.

In Figure 1, user actions (described in bold italic texts) are accompanied by user's natural language description. Note that a user in a normal setting without PLOW would not perform the actions that highlight a text or an area with mouse click and drag (underlined texts). However, those actions are easy-to-perform special actions that tell PLOW which information a user is interested in.

While user actions on a browser provides useful information, it is very difficult for a system to identify key high-level task information from the demonstration alone:

- *Identifying a task goal*: What is the final goal of the current task? What is the input and the output?
- *Identifying correct parameterization*: What values are required to perform the task? Is a value a constant or a variable? For variables, what is the relation among them?
- *Identifying iteration control structure*: What is the boundary of iteration? What actions to repeat? When to stop?
- *Identifying task hierarchy*: When a new (sub) task starts or ends?

With PBD based on conventional machine learning approaches such as [3,14,16], to identify the above key aspects of a task, multiple examples will be required to generalize learned knowledge. However, to build a task from a single demonstration, PLOW exploits the rich information from the play-by-play description by a user.

1. USER: “LET ME TEACH YOU HOW TO FIND HOTELS NEAR AN ADDRESS”
2. USER: "GO TO THIS WEBSITE"
3. USER: [***Enters a URL, http://www.mapquest.com, in the browser and hits enter***]
4. USER: “PUT HOTELS HERE”
5. USER: [***Types “hotels” in the business type field***]
6. USER: "YOU NEED TO KNOW THE ADDRESS"
7. PLOW: “WHAT IS THE ADDRESS?”
8. USER: [***Types an address, “40 South Alcaniz Street, Pensacola, FL”, in a special GUI for inputs***]
9. USER: "PUT THE STREET ADDRESS HERE”
10. User: [***Types the street address, “40 South Alcaniz Street”, in the street field***]
11. USER: “PUT THE CITY HERE”
12. USER: [***Types “Pensacola” in the city field***]
13. USER: “PUT THE STATE HERE”
14. USER: [***Types “FL” in the state field***]
15. USER: “CLICK THE SEARCH BUTTON”
16. USER: [***Presses a button with a label “Search”***]
17. USER: “HERE IS THE LIST OF HOTELS”
18. USER: [***Highlights an area with a list of hotels***]
19. PLOW: “CAN YOU CHECK IF I UNDERSTAND THE LIST?”
20. PLOW: *<Displays a special GUI that shows the contents of the hotel list>*
21. USER: “THIS IS THE HOTEL NAME”
22. USER: [***Highlights the name of a hotel in the special GUI***]
23. PLOW: *<Identifies hotel names for the other hotels and display the names>*
24. USER: “THIS IS THE HOTEL ADDRESS”
25. USER: [***Highlights the address of a hotel***]
26. PLOW: *<Identifies hotel addresses for the other hotels and display them>*
27. USER: “THIS IS THE HOTEL DISTANCE”
28. USER: [***Highlights the distance for a hotel***]
29. PLOW: *<Identifies distances for the other hotels and display them>*
30. USER: “CLICK THE NEXT LINK TO GET MORE RESULTS”
31. USER: [***Click the link labeled “next”***]
32. PLOW: “WHEN SHOULD I STOP SEARCHING?”
33. USER: “WHEN THE DISTANCE IS GREATER THAN TWO MILES”
34. PLOW: “I WILL TRY THAT”
35. PLOW: *<Visits multiple pages and extracts information (i.e., hotel name/address/distance) until the iteration termination condition is met and displays the results in the special GUI>*
36. PLOW: “I AM FINISHED”
37. USER: “I AM DONE”

Figure 1: A Dialogue to Find Hotels Near an Address

Another major point is PLOW’s proactive interaction with a user in a natural and unobtrusive manner (in Figure 1, upper

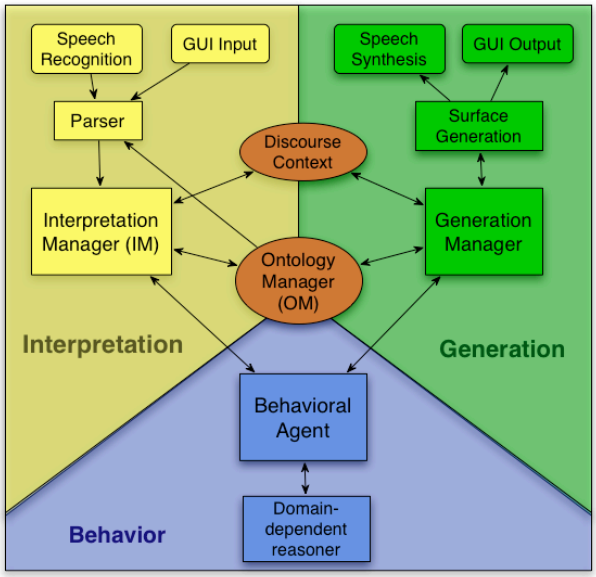


Figure 2: TRIPS architecture

case texts and the actions in angled brackets, both labeled with PLOW). For instance, PLOW makes queries (Line 32), reports its status (Line 36), and verifies its knowledge of the task under construction by presenting what it recognizes (Line 19 ~ 20) or executing an action or a set of actions that it has just learned (Line 23, 26, 28, 35).

Furthermore, the contextual information for an action enables PLOW to identify Web objects (e.g., textfield, link, etc.) in dynamic HTML pages. For instance, Line 11 (“*Put the city here*”) explains the action to type a city name into a field labeled with “City”. The NL description is used to find the city field in future execution with a new page format (e.g., new ads inserted at the top, reordering input fields in the search box, etc.).

With this play-by-play and mixed-initiative approach, PLOW is able to build a robust and flexible task from a single demonstration. Learned tasks can be easily improved and modified with a new example, and they can be also reused to build a larger task and shared with other users.

## PLOW ARCHITECTURE

PLOW is an extension to TRIPS [8], a dialogue-based collaborative problem solving system that has been applied to many real world applications.

### The TRIPS System

The TRIPS system provides the architecture and domain-independent capabilities for supporting mixed-initiative dialogue in various applications and domains. Its central components are based on a domain independent representation, including a linguistically based semantic form, illocutionary acts, and a collaborative problem-solving model. The system can be tailored to individual domains through an ontology mapping system that maps

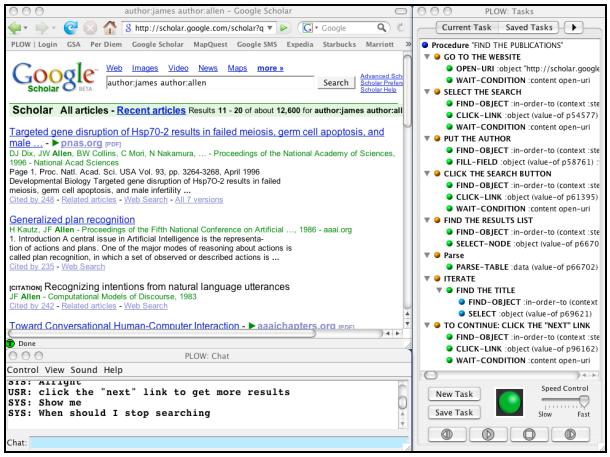


Figure 3: The PLOW Interface

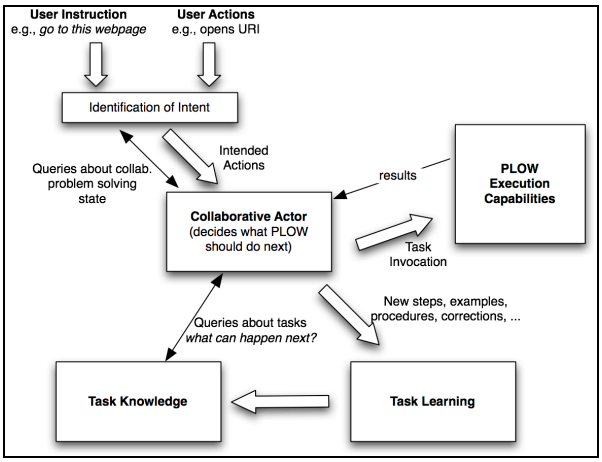


Figure 4: PLOW Architecture

domain-independent representations into domain-specific representations.

Figure 2 shows the core components of TRIPS: (i) a toolkit for rapid development of language models for the Sphinx-III speech recognition system, (ii) a robust parsing system that uses a broad coverage grammar and lexicon of spoken language, (iii) an interpretation manager (IM) that provides contextual interpretation based on the current discourse context, including reference resolution, ellipsis processing and the generation of intended speech act hypotheses, (iv) an ontology manager (OM) that translates between representations, and (v) generation manager (GM) and surface generator that generate system utterances from the domain-independent logical form.

The IM coordinates the interpretation of utterances and observed cyber actions. IM draws from the Discourse Context module as well to help resolve ambiguities in the input, and coordinates the synchronization of the user's utterances and observed actions. Then IM interacts with a behavioral agent (BA) to identify the most likely intended interpretations in terms of collaborative problem solving acts (e.g., propose an action, accept a problem solving act or ignore it, work on something more pressing, etc.) BA gets supports from additional reasoning modules specialized for each application domain and reports its status to GM that plans a linguistic act to communicate the BA's intentions to the user.

TRIPS components interact with each other by exchanging messages through a communication facilitator. Thu, they can be distributed among networked computers. The rest of this chapter will focus on the PLOW components. For further information about the TRIPS system, refer to [2,8].

### 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 3 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. Through the instrumentation, PLOW accesses and manipulates a tree-structured logical model of web pages, called DOM (Document Object Model). 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.

The domain independent aspect of PLOW was recently demonstrated in the work for a Military Health System for appointment booking. Most of PLOW codes were reused for a system called CHCS that was a terminal-based but still widely used legacy system. The major work involved instrumenting the terminal environment (e.g., observing key strokes, checking screen update, etc.) as well as extending the ontology for the healthcare domain. From a user's point of view, only noticeable major change to adapt was the replacement of a browser with a terminal.

### Collaborative Problem Solving in PLOW

Figure 4 shows a high-level view of the PLOW system. At the center lies a CPS (Collaborative Problem Solving) agent that acts as a behavioral agent in the TRIPS architecture. The CPS agent (henceforth, called CPSA) computes the most likely intended intention in the given problem-solving context (based on the interaction with IM). CPSA 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.

The CPSA understands the interaction as a dialogue between itself and a user. The dialogue provides the context for interpreting human utterances and actions, and provides the structure for deciding what to do in response. In this approach, PLOW appears to a user as a competent collaborative partner, working together towards the shared goal of one shot learning.

To give an overview of the collaborative problem solving, assume that a user introduced a new step. CPSA first

```
(task :id <a unique identifier>
      :goal <task goal>
      :description <NL description of the task>
      :documentation <notes generated by PLOW
                      but editable by a user afterward>
      :trigger <triggering conditions, if any>
      :pre-condition  <required inputs
                      & propositions to satisfy>
      :post-condition <task outputs
                      & propositions to assert>
      :completion-condition <a system state for
            completion that includes a list of actions
            to be completed & propositions
            to be satisfied>
      :steps
       ((step :preconditions <a list of propositions
                              to satisfy>
              :id <a unique identifier>
              :description <NL description of the step>
              :name <step name>
              :parameters <a list of parameter
                           description>
             :actions
               ((action :name <action name>
                        :parameters <a list of parameter
                                     description>)
                  (action ...) ...))
         ....
        (step :name ....))
```

Figure 4: Abstract Task Model

checks if it knows how to perform the step and, if so, initiates a dialogue to find out if the user wants to use a known task for the step in the current task. If the user says so, CPSA invokes another dialogue to check if the user wants to execute the reused task or not. Depending on user's responses, CPSA shows a different behavior. In the case of execution, CPSA enters into an execution mode and presents results when successful. If failed in execution, PLOW invokes a debugging dialogue, showing where it failed.

In some cases, the system takes proactive execution mixed with learning, following an explicit model of problem solving. In particular, this type of collaborative execution during learning is very critical in learning iteration without requiring the user to tediously demonstrate each loop over a significant period. Refer to [2] for the background and the formal model of collaborative problem solving in TRIPS.

## TEACHING WEB TASKS WITH PLAY-BY-PLAY

### Task Representation

A task is built as a series of steps and each step may be primitive (i.e., a self-contained terminal action) or correspond to another task (i.e., calling a subtask). However, a task model is more than a collection of steps. A task model needs to contain information such as an overall task goal, pre/post-conditions, the relationship between steps, the hierarchical structure of a task among others.

Information in TRIPS is expressed in AKRL (Abstract Knowledge Representation Language), a frame-like representation that describes objects with a domain-specific ontology using cross-domain syntax. AKRL retains the aspects of natural language that must be handled by the reasoning modules. While the complete specification of AKRL is not presented in this chapter due to limited space, examples will show how AKRL is used in PLOW.

A task model should be designed in such a way that it is easily executable by the system as well as applicable to further reasoning. Figure 4 shows an abstract task model in PLOW. The model includes a task goal, supplemental records, various conditions (pre/post/triggering/completion-condition), and step description. Each step consists of the name/preconditions/parameters/primitive-actions. Action definition includes its name and parameters.

Currently, it is assumed that the steps listed in the task model are executed sequentially in the listed ordering. However, the task model will be extended to include ordering constraints for flexibility (e.g., mutually exclusive steps could be performed in parallel).

Some attributes in the task model could be inferred from step definition if needed. For instance, if a variable is used in a step but there is no preceding step that provides its value, the variable should be considered as a task input. However, explicit representation of input parameters in the task model makes task execution and reasoning faster and easier.

Following sections will show how a task model is constructed through the collaboration between a user and PLOW that involves multiple communicative acts and highly sophisticated reasoning in PLOW. Additional information for PLOW can be also found at [1,13].

### Task Goal Definition

The task model is incrementally built as a user performs play-by-play demonstration. Figure 5 shows a part of the task model built from the dialogue in Figure 1. Given the user utterance "*Let me teach you to find hotels near an address*" (Line 1 in Figure 1), TRIPS natural language understanding modules parse and interpret it. IM computes multiple hypotheses and sends a request below to CPSA for evaluation:

- *(**request** :content (**evaluate** :content (cps-act :id CA1268 :content (**propose** :who user :to plow :id x126 **:what v103** :as goal) :context ((reln **v103** :instance-of **Teach** :object **v114** :recipient v105 :agent v108) (the v108 :instance-of Person :equals user) (the v105 :instance-of System :equals plow) (reln **v114** :instance-of Find :object v115) (a v115 :instance-of Set :element-type v116) (kind v116 :instance-of Hotel :is-near v117)*

(task :id p344
:goal ((reln v114 :instance-of Find :object v115) ⟶ (a)
(a v115 :instance-of set :element-type v116)
(kind v116 :instance-of **Hotel** :is-near v117)
(a v117 :instance-of Mailing-Address))
:description "find hotels near an address" ⟶ (b)
:documentation "id: p344, created: 03/03/2009 12:49:34, user: hjung" ⟶ (c)
:precondition (condition :inputs ((role :id p456 :task-id p344 :step-id p497
:value ***((a v117 :instance-of Mailing-Address))***)) ⟶ (d)
:postcondition (condition :outputs
((role … :value **((the v199 :instance-of Name :associated-with v200)** ⟶ (e)
**(kind v200 :instance-of Hotel))**)
(role … :value **((the v213 :instance-of Mailing-Address :associated-with v214)** ⟶ (f)
**(kind v214 :instance-of Hotel))**)
(role … :value **((the v243 :instance-of Distance :associated-with v244)** ⟶ (g)
**(kind v244 :instance-of Hotel))**)))
:completion-condition (condition :completed-actions (p456 … p786))
:steps … )

Figure 5: A Task Model Example

*(a v117 :instance-of Mailing-Address)) :channel desktop)) :reply-with IM126)*

CPSA reasons about the validity of the proposal and make a decision, accept or refusal. Here, the proposal is to teach a finding action the target of which is a set of hotels near a mailing address. When it is accepted, IM requests CPSA to commit to the proposal. Then, CPSA requests a task-learning module, henceforth called TL, to start the learning process:

- *(**request** :receiver TL :content (akrl-expression :content p126 :context ((reln p126 :instance-of **start-learn :task v114**) (reln **v114** :instance-of Find :object v115) (a v115 :instance-of Set :element-type v116) (kind v116 :instance-of Hotel :is-near v117) (a v117 :instance-of Mailing-Address))) :reply-with CPSA126 :sender CPSA)*

Receiving the above request, TL reasons about its feasibility. If feasible, TL accepts the task goal (Figure 5-a) and starts a learning process. Given TL's acceptance notification, CPSA updates its collaborative problem solving state and waits for the user to define a step.

The task model also includes other useful information such as task description (Figure 5-b) and documentation (Figure 5-c). Note that the task description is not the text directly from speech recognition. Instead, it is a text that the TRIPS surface generator produced from the internal representation of the task goal, which clearly shows that the system understands what a user said. The same goes for the step description in the task model. These reverse-generated NL description is used to describe the current task in the PLOW interface (the right side window in Figure 3). PLOW automatically generates the documentation part but a user can edit the text later.

A task may also have a trigger: e.g., when a user says, "*Let me teach you how to book hotels near an airport when a flight is canceled*", the even of a canceled flight (that can be notified in various forms) is captured as a trigger and recorded in the task model. While PLOW is running, if such an event is notified, PLOW finds a task with a matching triggering condition and, if any, execute it.

#### Task Step Definition

*High-level Step Description in Play-by-Play*

When a user describes a step by saying "*Go to this website*" (Line 2 in Figure 1), IM and CPSA collaboratively interpret and reason about the utterance. Then, CPSA requests TL to identify a step by sending a message that contains step description:

- *(**request** :sender CPSA :receiver TL :content (akrl-expression :content i132 :context (reln i132 :instance-of **identify-substep** :content **v127** :task-id p344) (reln **v127** :instance-of **Navigate** :**web-destination v128** :agent v132) (the **v128** :instance-of **Webpage**) (a v132 :instance-of System :equals PLOW))) :reply-with p327)*

Given the information in this request, TL creates a step that does not have actions yet and inserted the step definition into the current task model:

- *(**Step** :preconditions ((ordering :constraint (directly-after nil))) :name **Navigate** :parameters ((para :name **Web-destination** :value ((the v128 :instance-of **Webpage**))) :id p351 :description "go to the website" :actions null)*

When there is no specific precondition, the precondition part has a simple ordering constraint that normally indicates a step can be performed after the completion of a preceding step: for the first step, the preceding step is NIL.

*Primitive Actions of a Step*

Following the step description, a user performs a normal navigation action in the browser and the action is detected

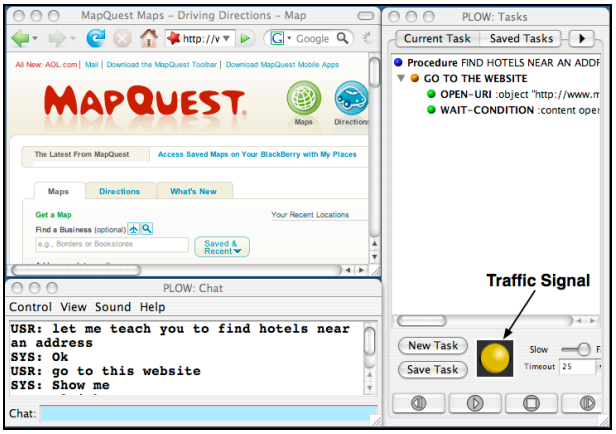


Figure 6: PLOW Interface after Step Demonstration

by the Firefox instrumentation. IM receives the action and checks with CPSA. Then, after checking the validity of the action, CPSA requests TL to learn the action:

- *(**request** :sender CPSA :receiver TL :content (akrl-expression :content i331 :context ((reln i331 :instance-of **identify-example** :content gui304 :step-id p351 :taks-id p344 :cps-act pursue-goal) (reln gui304 :instance-of **Open-URI** :actor nil :object "**http://www.mapquest.com**"))) :reply-with p377)*

Using the information in this request, TL extends the action part in the step definition above:

- *(**Step** … :actions ((**action** :name **Open-URI** :parameters ((para :name window :value (opened at step p351)) (para :name object :value "**http://www.mapquest.com**"))) (**action** :name **Wait-Condition** :parameters ((para :name content :value **Open-URI**)))))*

Note that TL inserts additional information into the action definition based on its domain knowledge. To handle multiple windows, a browser window to perform the current action is specified. In addition, an action to wait for complete web page loading is inserted. Without such synchronization, subsequent actions could fail, in particular, on a slow network (e.g., trying to select a menu when the target menu does not appear yet). In navigating to a link, there can be multiple page loading events (e.g., some travel websites show intermediate web pages while waiting for search results). PLOW observes how many page loading events have occurred and inserts waiting actions accordingly.

Figure 6 shows the PLOW interface after this step demonstration. The right side window for the current task under construction has a traffic signal light at the bottom portion. The signal changes colors (green/red/yellow) based on PLOW's internal processing state and its expectation of the application environment, telling if it is deemed OK for a user to provide inputs to PLOW (green) or not (red). Yellow implies that PLOW is not sure since, in this case, there can be multiple page loading events controlled by the web site server.

*Dynamic Web Objects in Primitive Actions*

In Figure 1, there is a step created by saying "*Put the city here*" and typing a city name into a text field labeled with "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 semantic description helps PLOW to find the text field in a dynamic HTML file.

Figure 7 is a screenshot of Firefox DOM Inspector that shows DOM nodes and their attributes/structure accessed by PLOW for its learning how to identify dynamic objects. For the step to put a city, PLOW finds a match for "city" in one of the attributes of the INPUT node (i.e., id="startCity"). PLOW learns the relation between the semantic concept and the node attribute as a rule for future execution. Linguistic variation (e.g., cities) or similar ontological concepts (e.g., town, municipality) are also considered for the match. Right after learning this new rule, PLOW verifies it by applying the rule in the current page and checking if the object (i.e., a text-field) found by the rule is equal to the object observed in demonstration.

PLOW also uses other heuristics to learn a rule. For instance, when the node identified in the demonstration does not have any semantic relation, it finds another reference node traversing the DOM tree and, if found, computes the relation between the node observed in demonstration and the reference node found elsewhere. With this sophisticated approach, even when there is a web page format change, PLOW is able to find a node as long as there are no significant local changes around the node in focus. For further information of PLOW's dynamic web object identification, refer to [3].

**Parameter Identification**

Identifying parameters is challenging even for a simple task and, without special domain knowledge, it is almost

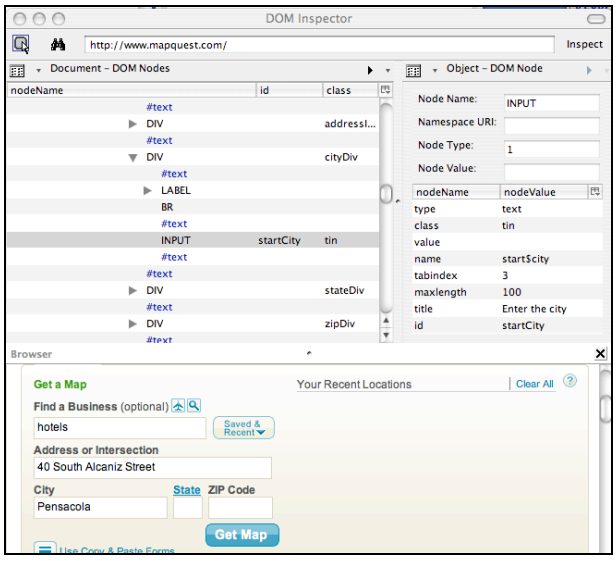


Figure 7: DOM structure of a Web Page

impossible with only a single observation. 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 8 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. For instance, the city instance in one step (Line 11 in Figure 1) is related to the address mentioned earlier (Line 1 and Line 6). The semantic concept CITY is a role of another concept ADDRESS in the TRIPS ontology. A special address parser helps to reason that the typed city name "Pensacola" in the demonstrated action (Line 12) matches a city part of the given full address provided by a user (Line 6 ~ 8). Without this dependency relation from language understanding and the verification by the address parser, PLOW will add the city as a separate input parameter. Note that, in the final task model, there is only a single input parameter, an address, (Figure 5-d).

NL description also helps to identify output parameters. From the utterances that specify which information to extract (Line 21, 24, 27 in Figure 1), PLOW figures out that the objects to find in those steps are related to the task output defined in the task definition (i.e., hotel in Line 1). Therefore, they are added as output parameters (Figure 5-e,f,g).

### 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.

### 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 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.

Assume that a user adds a conditional step by saying "If a zipcode is available, put the zipcode here". Then, in the precondition part, the step definition will include the following proposition that states the existence of the zipcode property:

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

Figure 8: Interpretation of Noun Phrases

- (step … :preconditions (…((reln v380 :instance-of **Have-Property** :property v346 :force **true**) (a v346 :instance-of **Zip-code**))) …)

### 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 9, a user is teaching PLOW how to find hotels near an address. When the user highlights a list of results (Figure 9-a) 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 9-b).

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

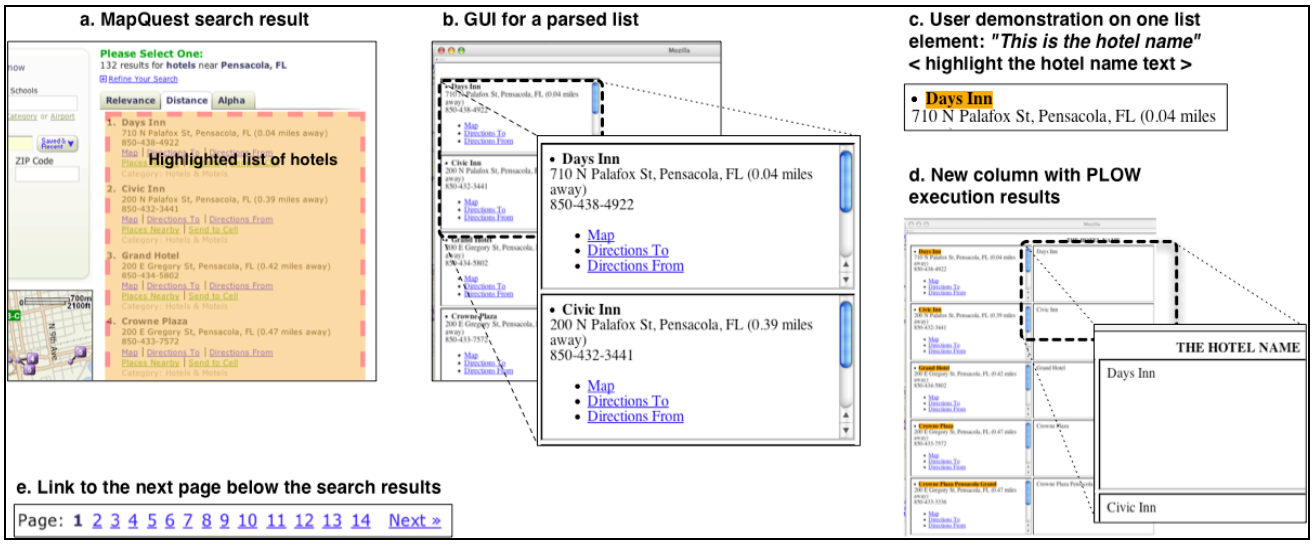


Figure 9: Learning Iteration

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 9-c). Given this information, PLOW learns the extraction pattern and *proactively* applies the rule to the rest of elements (Figure 9-d). 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 9-e). 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*".

The conditions can be defined on the 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 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.

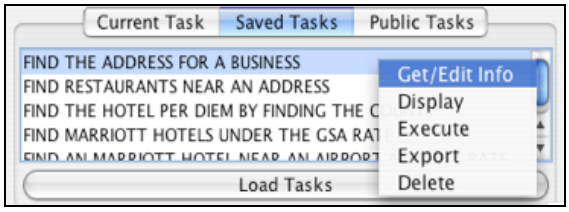


Figure 10: Task Management

### 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

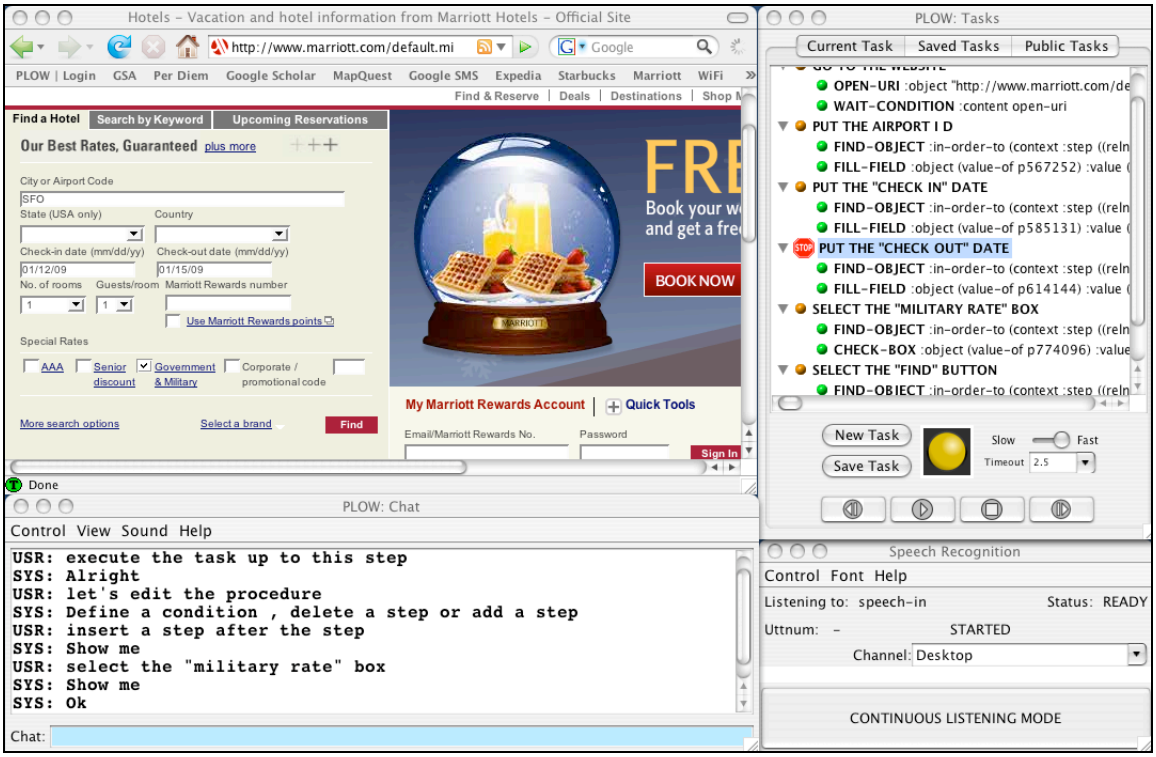


Figure 11: Task Editing

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. Figure 11 shows the interaction between a user (USR) and PLOW (SYS) for task editing that was to add a new step to select a check-box for a special military rate in booking a hotel. Note that, before the dialogue in the chat window, the user selected the step described as "Put the check out date" in the current task window (marked with a stop sign).

### 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.

In Figure 12, the task is to find hotel per diem rates for a city and a state. The failure occurred at the step to find a per diem rate list for Nebraska (marked with a ladybug). In

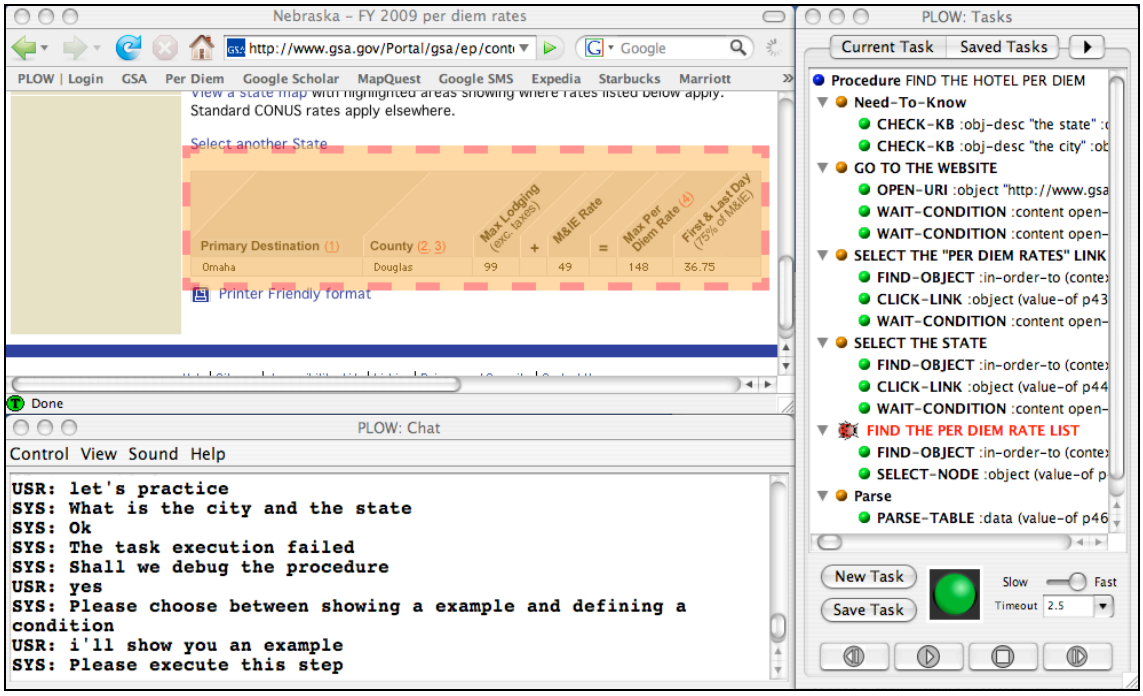


Figure 12: Failure Correction with a New Example

teaching the task, another state that has multiple destinations was used. However, this Nebraska table with one destination has a significant different structure and PLOW failed to identify the list. The dialogue in the chat window shows how PLOW and a user interact with each other from the start of task execution. As shown in the browser window, the user gives a new example by highlighting the per diem rate list for the failed step. Now, the constructed task has become more robust with more knowledge to handle different list structures.

## 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 [18]. Sixteen human subjects received training on each system and they were given ten problems that can be performed on various web sites:

1. To whom should a travel itinerary be emailed?
2. List all publications from the work funded by a project
3. List top N candidates for given product specifications
4. Retrieve N product reviews for a product
5. List restaurants within a certain distance from an address
6. In what conference an article was published?
7. What articles are cited in a given article?
8. What articles cite a given article?
9. Who else is traveling to a location on the same day with a person of interest?
10. What roles does a person play in an institution?

PLOW did very well in both tests, receiving a grade of 2.82 (2006) and 3.47 (2007) out of 4 (exceeding the project goals in both cases). Furthermore, in a separate test in 2006, test subjects were given a set of new 10 "surprise" problems some of which were substantially different from the original ten problems. They were free to choose from different systems. But, PLOW was the system of choice among the subjects: 30 out of 55 tasks were created in the surprise test using PLOW and 13 out of 16 used PLOW at least once. PLOW also received the highest average score (2.2 out of 4) for the constructed tasks in the test. In addition to high test scores, anecdotal comments from the subjects in 2007 were that they were impressed by PLOW's user convenience with various GUI/NL interaction.

## RELATED WORK

A major technique in task learning is an observation-based approach in which agents learn task models through observation of the actions performed by an expert [3,14,16]. However, a significant drawback of these approaches is that they require multiple examples, making them infeasible for one-shot learning in most cases without very special domain knowledge.

Researchers also investigated techniques that do not require observation. [11, 15] proposed techniques to encode experts' knowledge with annotation. A collaborative scripting system (called Coscripter) with pseudo natural language was developed to automate online tasks and the information in the pseudo NL was used to identify web objects [17]. A specialized GUI system for task editing and modification was also developed as a part of the CALO project [4]. While these approaches are useful and novel,

without the help of demonstration observation, the task learning can be difficult for complex control constructs such as iteration and dynamic web object identification. Creo is a PBD system that can learn a task from a single example but it has significant limitation in the range of actions and web objects [8].

Many mashup systems were developed to extract and integrate information from the Web [7,10,12]. While they are powerful tools, their capability and complexity are positively correlated (i.e., complex interfaces are provided to provide advanced functionalities). Furthermore, there is limitation in handling dynamic objects and understanding extracted information for further reasoning.

## CONCLUSION

PLOW demonstrates that NL is a powerful intuitive tool for end-users to build web tasks with significant complexity using only a single demonstration. The natural play-by-play demonstration that would occur in human-human teaching provides enough information for the system to generalize demonstrated actions. Mixed-initiative interaction also makes the task building process much more convenient and intuitive. Without the system's proactive involvement in learning, the human instructor's job could become very tedious, difficult, and complex. Semantic information in NL description also makes the system more robust by letting it handle the dynamic nature of the Web.

While PLOW sheds more light on NL's roles and the collaborative problem solving aspects in the end-user programming on the Web, significant challenges still exist and new ones will emerge as application domains are expanded. Better reasoning about tasks, broader coverage of language understanding, and handling the dynamic nature of web contents will be needed to address the challenges.

## 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. James Allen, Nate Blaylock, and George Ferguson, A problem solving model for collaborative agents. *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2002
3. Richard Angros et al., Learning Domain Knowledge for Teaching Procedural Skills, *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2002
4. Jim Blythe, Task Learning by Instruction in Tailor. *Proceedings of the International Conference on Intelligent User Interfaces*, 2005
5. 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
6. Allen Cypher, editor. Watch what I do: Programming by demonstration. MIT Press, Cambridge, MA, 1993
7. Rob Ennals et al., Intel Mash Maker: Join the Web, *ACM SIGMOD Record*, Volume 36, Issue 4, 2007
8. Alexander Faaborg and Henry Lieberman, A Goal-Oriented Web Browser, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* , 2006
9. George Ferguson and James Allen, TRIPS: an integrated intelligent problem-solving assistant, *Proceedings of the National Conference on Artificial Intelligence*, 1998
10. Jun Fujima et al., Clip, Connect, Clone: Combining Application Elements to Build Custom Interfaces for Information Access, *Proceedings of the annual ACM symposium on User interface software and technology*, 2004
11. Andrew Garland, Kathy Ryall, and Charles Rich, Learning Hierarchical Task Models by Defining and Refining Examples. *Proceedings of the International Conference on Knowledge Capture*, 2001
12. David Huynh, Robert Miller, and David Karger, Potluck: Data Mash-Up Tool for Casual Users, *Proceedings of the International Semantic Web Conference*, 2007
13. 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
14. Tessa Lau and Dan Weld, Programming by demonstration: an inductive learning formulation. *Proceedings of the International Conference on Intelligent User Interfaces,* 1999.
15. Frank Lee and John Anderson, Learning to act: Acquisition and Optimization of Procedural Skill, *Proceedings of the Annual Conference of the Cognitive Science Society*, 1997
16. Michael van Lent and John Laird, Learning Procedural Knowledge through Observation, *Proceedings of the International Conference on Knowledge Capture*, 2001
17. Gilly Leshed et al., Coscripter: Automating & sharing how-to knowledge in the enterprise, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2008
18. DARPA CALO project: http://caloproject.sri.com