# Learning Collaborative Tasks on Textual User Interfaces

**Nate Blaylock** and **William de Beaumont** and **Lucian Galescu** and **Hyuckchul Jung**
Florida Institute for Human and Machine Cognition
Pensacola, Florida, USA
{blaylock,hjung,wbeaumont,lgalescu}@ihmc.us

**James Allen** and **George Ferguson** and **Mary Swift**
University of Rochester, Dept. of Computer Science
Rochester, New York, USA
{james,ferguson,swift}@cs.rochester.edu

## Abstract

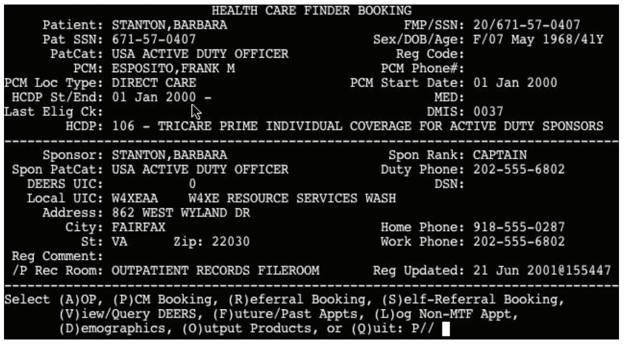
A novel 'play-by-play' based procedure learning system has been successfully applied to automate Web tasks. In the transition to a non-Web domain, for a system widely used at US military hospitals, we addressed new challenges by enabling the system to learn collaborative tasks and understand unstructured application environment. This paper presents our approach and its promising evaluation results.

## Introduction

Our daily activities typically involve the execution of a series of tasks, and we envision personal assistant agents that can help by performing many of these tasks on our behalf. To realize this vision, agents need to have the ability to learn task models, ideally from one or very few examples. Researchers have attempted to learn these models by observation, creating agents that learn through observing the expert's demonstration (Angros et al. 2002; Lau and Weld 1999; van Lent and Laird 2001; Faaborg and Lieberman 2006). However, these techniques require multiple examples of the same task, and the number of required training examples increases with the complexity of the task.

Previously, we presented work on the PLOW system (Jung et al. 2008; 2010), which is able to learn tasks on the web from only a handful of examples (often even with a single example) through observation accompanied by a natural language (NL) 'play-by-play' description from the user. The play-by-play approach in NL enables our task learning system to build a task with high-level constructs that are not inferable from observed actions alone. The synergy between the information encoded in the user's NL description and the system's basic knowledge of observed actions makes it possible to learn, from a simple sequence of actions, a complex task structure that reflects the user's underlying intentions in the demonstration.

While the PLOW system has been successfully applied to various tasks on the Web (e.g., finding restaurants/flights/etc.), the tasks performed by the system were mostly performed in a batch style in that, given a task invocation request, the system asks for some inputs and then the system executes the task until it gets the final results to present or hits problems (that can be corrected by a user in a debugging session).



```
                        HEALTH CARE FINDER BOOKING
     Patient: STANTON,BARBARA                         FMP/SSN: 20/671-57-0407
     Pat SSN: 671-57-0407                         Sex/DOB/Age: F/07 May 1968/41Y
      PatCat: USA ACTIVE DUTY OFFICER                Reg Code:
         PCM: ESPOSITO,FRANK M                     PCM Phone#:
PCM Loc Type: DIRECT CARE                     PCM Start Date: 01 Jan 2000
 HCDP St/End: 01 Jan 2000 -                               MED:
Last Elig Ck:                                            DMIS: 0037
        HCDP: 106 - TRICARE PRIME INDIVIDUAL COVERAGE FOR ACTIVE DUTY SPONSORS
--------------------------------------------------------------------------------
     Sponsor: STANTON,BARBARA                       Spon Rank: CAPTAIN
 Spon PatCat: USA ACTIVE DUTY OFFICER              Duty Phone: 202-555-6802
   DEERS UIC:           0                                 DSN:
   Local UIC: W4XEAA    W4XE RESOURCE SERVICES WASH
     Address: 862 WEST WYLAND DR
        City: FAIRFAX                              Home Phone: 918-555-0287
          St: VA      Zip: 22030                   Work Phone: 202-555-6802
 Reg Comment:
 /P Rec Room: OUTPATIENT RECORDS FILEROOM       Reg Updated: 21 Jun 2001@155447
--------------------------------------------------------------------------------
Select (A)OP, (P)CM Booking, (R)eferral Booking, (S)elf-Referral Booking,
       (V)iew/Query DEERS, (F)uture/Past Appts, (L)og Non-MTF Appt,
       (D)emographics, (O)utput Products, or (Q)uit: P//
```

Figure 1: A screenshot of the CHCS system

### The Application Domain

The Composite Health Computer System (CHCS) is a textual user interface system used throughout the US Military Health System for booking patient appointments and other tasks. In CHCS, the keyboard is used to navigate through a complex web of menus and screens in order to book and cancel appointments, look up patient information, and perform other administrative tasks. A typical screenshot of the system is shown in Figure 1.[1] In the top part of the figure, spacing is used to make visual tables. The bottom shows a mnemonics-based menu and the cursor awaiting user keyboard input.

In developing and evaluating the system, we were given access to the actual CHCS server in use at Naval Hospital Pensacola, although, to protect live data, testing was done on a fictitious database used for training staff on CHCS use.

### Challenges

In this paper, we discuss our work on porting the PLOW system to the appointment booking domain in CHCS. In moving to this domain, we addressed the following challenges:

[1]The patient information shown here and in other figures is part of a training database and is fictitious.

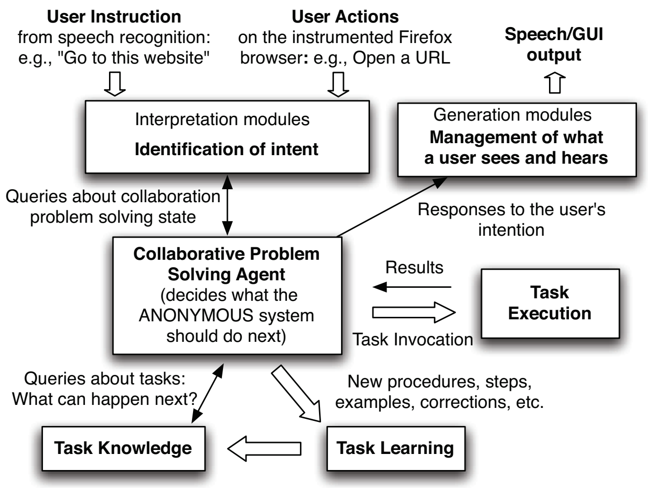
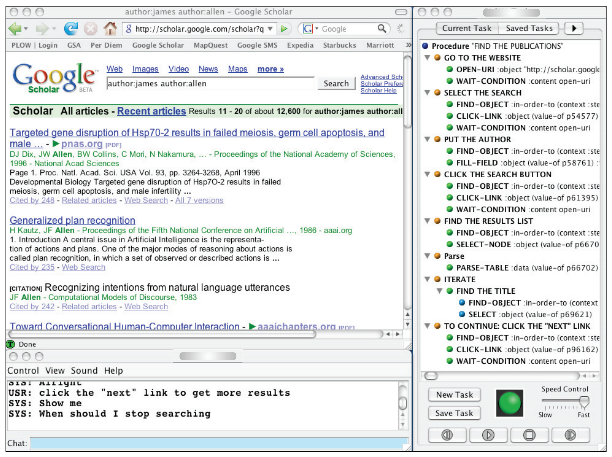


Figure 2: Information flow in the PLOW system



author:james author:allen

**Scholar** **All articles** - Recent articles Results **11** - **20** of about **12,600** for **author:james author:all**

Targeted gene disruption of Hsp70-2 results in failed meiosis, germ cell apoptosis, and male ... - pnas.org [PDF]
DJ Dix, JW **Allen**, BW Collins, C Mori, N Nakamura, ... - Proceedings of the National Academy of Sciences, 1996 - National Acad Sciences
Page 1. Proc. Natl. Acad. Sci. USA Vol. 93, pp. 3264-3268, April 1996
Developmental Biology Targeted gene disruption of Hsp7O-2 results in failed
meiosis, germ cell apoptosis, and male infertility ...
Cited by 248 - Related articles - Web Search - All 7 versions

Generalized plan recognition
H Kautz, JF **Allen** - Proceedings of the Fifth National Conference on Artificial ..., 1986 - aaai.org
1. Introduction A central issue in Artificial Intelligence is the representa-
tion of actions and plans. One of the major modes of reasoning about actions is
called plan recognition, in which a set of observed or described actions is ...
Cited by 235 - Web Search

[CITATION] Recognizing intentions from natural language utterances
JF **Allen** - Computational Models of Discourse, 1983
Cited by 242 - Related articles - Web Search

Toward Conversational Human-Computer Interaction - aaaichapters.org [PDF]

```
SYS: Alright
USR: click the "next" link to get more results
SYS: Show me
SYS: When should I stop searching
```

```
Procedure "FIND THE PUBLICATIONS"
  GO TO THE WEBSITE
    OPEN-URI :object "http://scholar.google
    WAIT-CONDITION :content open-uri
  SELECT THE SEARCH
    FIND-OBJECT :in-order-to (context :ste
    CLICK-LINK :object (value-of p54577)
    WAIT-CONDITION :content open-uri
  PUT THE AUTHOR
    FIND-OBJECT :in-order-to (context :ste
    FILL-FIELD :object (value-of p58761) :
  CLICK THE SEARCH BUTTON
    FIND-OBJECT :in-order-to (context :ste
    CLICK-LINK :object (value-of p61395)
    WAIT-CONDITION :content open-uri
  FIND THE RESULTS LIST
    FIND-OBJECT :in-order-to (context :ste
    SELECT-NODE :object (value-of p6670
  Parse
    PARSE-TABLE :data (value-of p66702)
  ITERATE
    FIND THE TITLE
      FIND-OBJECT :in-order-to (context
      SELECT :object (value-of p69621)
  TO CONTINUE: CLICK THE "NEXT" LINK
    FIND-OBJECT :in-order-to (context :ste
    CLICK-LINK :object (value-of p96162)
    WAIT-CONDITION :content open-uri
```

Figure 3: The original web-based PLOW system

- *Learning Collaborative Tasks*: The batch task execution model described above does not work for some tasks in the appointment booking domain. We extended PLOW to learn and execute collaborative tasks, where certain actions are assigned to the user.
- *Learning in an Unstructured Application Environment*: CHCS uses a terminal-based textual user interface, as opposed to the Web interface used by PLOW. We extended the system to interface with textual user interface systems, including work on understanding screens that are presented as text, not backed by a structured model such as the Document Object Model (DOM).

Moving to a new domain also provided an opportunity to test the robustness and applicability of the PLOW system across different domains by checking *which parts of the system we were able to reuse in the new domain*. In this paper, we first briefly discuss the original PLOW system. Then, we will present our approach for the above challenges and its results including an evaluation of the system with end-users with no programming experience.

## The PLOW System

The PLOW system is an extension to TRIPS: our collaborative problem solving-based dialog system (Allen et al. 2000). The system's core components include speech recognition, a robust semantic parsing system, an interpretation manager, an ontology manager, and generation. Figure 2 shows a high-level view of the PLOW system. At the center lies a collaborative problem solving agent (CPS) that computes the most likely intended intention in the given problem-solving context. 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.

While the system's core reasoning modules are designed to learn tasks in various domains (e.g., office applications, robots, etc.), it focused on tasks on the web, which involve actions such as navigation, information extraction, and form-filling with extended ontology to cover web browsing actions. Figure 3 shows the original system's user interface. The main window on the left is the Firefox browser instrumented with an extension that allows the system to monitor user actions and execute actions for learned tasks.

The instrumentation provides the system with a high-level model of browser interaction, allowing the system to access and manipulate the browser through the tree-like DOM of the webpage. On the right is a window 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 allows both speech and typed interaction from the user.

While detailed NL processing/reasoning is not presented here due to space limitations, NL provides rich information about the structure and elements of a task to learn. For instance, a single user utterance such as "*let me show you how to find publications for an author*" tells the system (i) the task goal, (ii) the final results (e.g., a set of publications) and (iii) a required input (i.e., an author). For a step, the user's play-by-play description of a browsing action provides semantics for the action. Just saying "*put the author's name here*" before typing text in a text field helps the system to figure out that the field requires a name role of an author.

Furthermore, NL provides key information for an iterative structure that normally requires numerous examples for a system to infer from pure observation. Utterances such as "*this is a list of articles*" and "*select next to get more results*" indicate the boundary of a loop as well as the context for an action for a next round. The semantic information encoded in NL also enables the system to reliably identify objects in non-static web pages. For instance, for the action to select a link labeled as "advanced scholar search", if a user describes the action by saying "*select the advanced search link*", the system can learn the relation between the semantic concept with its linguistic forms and the link's attributes in the DOM. Linguistic variation and similar ontological concepts are also considered in finding such a relation.

**System Reusability** For the new CHCS domain, most of the PLOW system components were reused, with minor additions to the lexicon and the ontology for the new domain, as well as additions for handling collaborative behavior; the one extension requiring significant effort was the switch from the DOM-backed instrumentation of the browser interface to the instrumentation of the text-based terminal interface. Retaining most NLP and task learning components with little or very minor updates, the system enabled intuitive and natural dialogs in the new domain at the level of complexity shown in the sample dialog below.

User: *Let me show you how to find patient records*
.....
User: *Put the patient search info here*
⟨Types the last name and hits ENTER⟩
User: *This is a list of patients*
⟨Highlights a patient list by mouse click and drag⟩
User: *This is the selection number*
⟨Highlights the number in the front of a row⟩
User: *This is the patient name*
⟨Highlights the patient name⟩
User: *This is the SSN*
⟨Highlights the Social Security number⟩
User: *Hit enter to get more results*
⟨hits ENTER⟩
**System: When should I stop searching?**
User: *Get all of them*
.....

Table 1: Part of a dialog to teach a task to extract patient information (user actions between angled brackets)

## Adding Support for Textual User Interfaces

The first challenge of moving to the CHCS domain was moving from a Web environment to a terminal-based textual user interface. As with the web-based PLOW system, we found that, in textual user interfaces, the level of interaction between the terminal and the system needed to be at a sufficiently high level so that observed actions roughly corresponded to what was being talked about in the user's utterances. Additionally, the system needed a higher-level representation of displayed text on the screen in order to support information extraction for task execution.

### Terminal Instrumentation

To access the CHCS system, we instrumented an open-source terminal emulator, JCTerm, to allow PLOW to observe user actions and screen updates and report them at a high level, and to allow it to control the interface.

**Action Observation and Control** At a low level, all user commands in a terminal system are of the form of keystrokes, or rather, ASCII characters sent to the system. For better understanding, the instrumented JCTerm models commands at a higher-level, including string input (cf. form filling), sending of control codes, and screen navigation (using `RETURN`, arrow keys, `PAGE UP/DOWN`, etc.)

**Screen Update Modeling** In the web version of PLOW, a browsing action may cause one or more page loading events (e.g., some websites show multiple intermediate pages before presenting search results). In such a case, explicit wait conditions are included into a task model based on the number of observed page loadings so that the following action would not be performed on an intermediate or a partially loaded page. Task learning in a textual user interface encounters the same type of problem — knowing when a screen update is done. However, terminal interfaces receive a continuous stream of commands to update parts of the screen (without explicit ending signals). We addressed this screen update issue by distinguishing major (new screen or partial scrolling) from minor (key echo) screen updates.

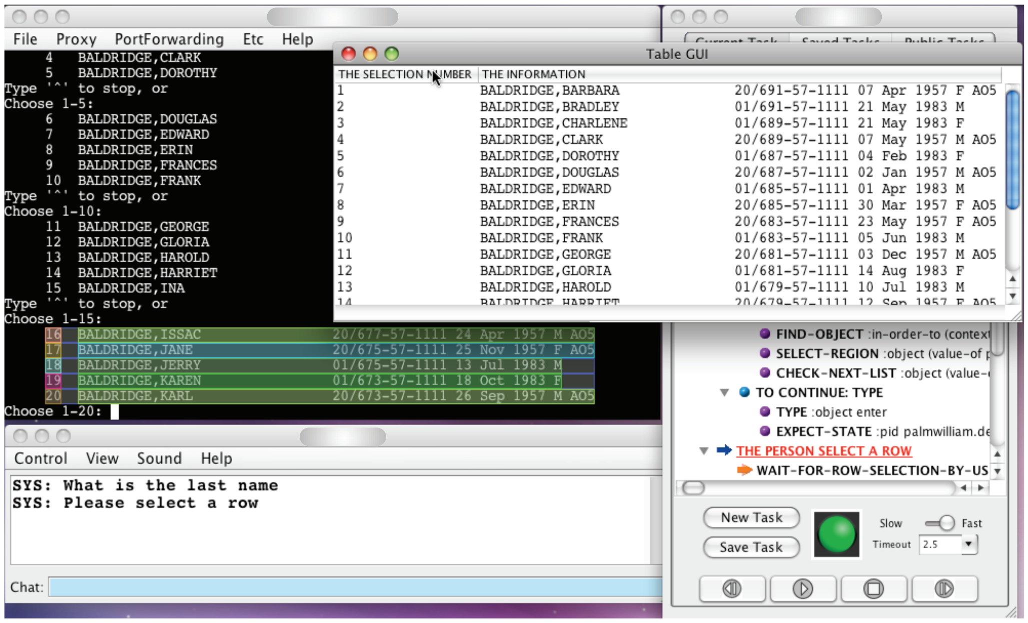
In our task model, each action that causes a major update results in a waiting condition for the final resulting screen state (represented by the position of the cursor and its surrounding text). During execution, for each waiting condition, the system checks if the current screen state matches the state identified in learning whenever it observes a burst of screen update activities followed by a silent period (e.g., 250 ms). While the waiting conditions in the Web system worked very well, identifying terminal screen states was not robust. Failures or false-positives to recognize screen states caused problems, making the system wait unnecessarily or proceed prematurely. (This problem got exacerbated in a slow network environment, putting our system at a significant disadvantage as described in Section *"Evaluation"*.)

### Screen Understanding

Useful tasks are able to not only echo commands (such as macros), but also understand and extract screen content, either for providing results to the user, or to recognize values needed for conditional statements in the task. The Web PLOW system contained a module that used semantic information from both the page and the user's description, along with the DOM structure of the page to learn patterns for extracting and parsing parts of the webpage as part of the learned task (Chambers et al. 2006).

We implemented a similar capability in the terminal PLOW system. We extended JCTerm to allow the user to use the mouse to highlight rectangular regions of interest on the screen. The system was extended to use information from the screen, paired with the language description to learn a pattern for extracting and parsing the same region of interest during execution. Figure 4 shows highlighting on the terminal screen and resultant table (in the GUI window to the right) the system extracted during execution.

It is important to note that finding these regions in CHCS is not as trivial as just extracting a rectangle with the same screen coordinates as were used in learning. The terminal screen in Figure 4 shows the CHCS patient finding subsystem. Here, matching patients from the search screen are displayed in groups of five, and the user can either choose one of the listed numbers by typing in the number, or can hit `RETURN` to get the next five search results. The screen in the figure shows parts of the first four lists of results, which are successively extracted by the system in the middle of a loop. Because this list scrolls up, instead of loading a new



```
     4    BALDRIDGE,CLARK
     5    BALDRIDGE,DOROTHY
Type '^' to stop, or
Choose 1-5:
     6    BALDRIDGE,DOUGLAS
     7    BALDRIDGE,EDWARD
     8    BALDRIDGE,ERIN
     9    BALDRIDGE,FRANCES
     10   BALDRIDGE,FRANK
Type '^' to stop, or
Choose 1-10:
     11   BALDRIDGE,GEORGE
     12   BALDRIDGE,GLORIA
     13   BALDRIDGE,HAROLD
     14   BALDRIDGE,HARRIET
     15   BALDRIDGE,INA
Type '^' to stop, or
Choose 1-15:
     16   BALDRIDGE,ISSAC                   20/677-57-1111 24 Apr 1957 M AO5
     17   BALDRIDGE,JANE                    20/675-57-1111 25 Nov 1957 F AO5
     18   BALDRIDGE,JERRY                   01/675-57-1111 13 Jul 1983 M
     19   BALDRIDGE,KAREN                   01/673-57-1111 18 Oct 1983 F
     20   BALDRIDGE,KARL                    20/673-57-1111 26 Sep 1957 M AO5
Choose 1-20:
```

Table GUI

```
THE SELECTION NUMBER  THE INFORMATION
1                     BALDRIDGE,BARBARA         20/691-57-1111 07 Apr 1957 F AO5
2                     BALDRIDGE,BRADLEY         01/691-57-1111 21 May 1983 M
3                     BALDRIDGE,CHARLENE        01/689-57-1111 21 May 1983 F
4                     BALDRIDGE,CLARK           20/689-57-1111 07 May 1957 M AO5
5                     BALDRIDGE,DOROTHY         01/687-57-1111 04 Feb 1983 F
6                     BALDRIDGE,DOUGLAS         20/687-57-1111 02 Jan 1957 M AO5
7                     BALDRIDGE,EDWARD          01/685-57-1111 01 Apr 1983 M
8                     BALDRIDGE,ERIN            20/685-57-1111 30 Mar 1957 F AO5
9                     BALDRIDGE,FRANCES         20/683-57-1111 23 May 1957 F AO5
10                    BALDRIDGE,FRANK           01/683-57-1111 05 Jun 1983 M
11                    BALDRIDGE,GEORGE          20/681-57-1111 03 Dec 1957 M AO5
12                    BALDRIDGE,GLORIA          01/681-57-1111 14 Aug 1983 F
13                    BALDRIDGE,HAROLD          01/679-57-1111 10 Jul 1983 M
14                    BALDRIDGE,HARRIET         20/679-57-1111 12 Sep 1957 F AO5
```

```
SYS: What is the last name
SYS: Please select a row
```

```
        FIND-OBJECT :in-order-to (context
        SELECT-REGION :object (value-of p
        CHECK-NEXT-LIST :object (value-
    TO CONTINUE: TYPE
        TYPE :object enter
        EXPECT-STATE :pid palmwilliam.de
  THE PERSON SELECT A ROW
      WAIT-FOR-ROW-SELECTION-BY-US
```

Figure 4: A screenshot of the PLOW system on a terminal running CHCS

page and displaying in the exact same region, a more complex algorithm is necessary.

The system learns and executes extraction patterns by first processing the screen in order to find important regions within it, and the relationships among their boundaries. It recognizes runs of characters with the same color and style attributes, and runs of singly-spaced words. Then it composes these runs into columns, if they are vertically adjacent and aligned on the left side. Other important regions include the entire screen, the cursor, and the area of the screen that has been updated since the last keyboard events.

The system then learns patterns for finding regions by building a graph of the relationships among the natural language representation, the neighboring important regions in the screen state, and the region to find. To do this, the system performs a best-first search over alignments between this graph and that of the current screen state and then extracts the analogous region in the current screen graph.

## Collaborative Tasks

In the initial version of PLOW, the system learned tasks that it itself executed. The model was that, when given an execution request, the system queried the user for any task parameters. Once the parameters were known, the system went off and executed the task, reporting the results to the user when execution was complete.

An important improvement made to the PLOW system is support for learning and executing *collaborative* tasks, where both the system and the user are assigned actions. Collaborative tasks allow a much wider range of behavior to be learned by the system. This was necessary in the appointment booking domain for two reasons: first, collaborative tasks allow the user and system to work together in execution more *efficiently and intuitively*. As an example, at the start of a booking task, the database entry for the patient needs to be identified from a patient's last name and his/her last four SSN digits. A collaborative task (shown in execution in Figure 4) will let the system ask for the last name, gather up search results that it learned to extract, and present them to the user for selection. Once selection is done, the selected patient's number will be automatically typed and the patient's information will be accessed and further processed by the system as taught before. This results in a more efficient interaction where the user needs only to provide the last name and then clicks on the matching patient from the result list in a clear view, with repetitive navigation and information-extraction steps performed by the system.

The second reason that collaborative tasks were necessary for this domain was that certain knowledge needed by the system to make decisions is sometimes only available after partial task execution and can be very difficult for the system to access/query. The example here is which timeslot should be chosen for the appointment. In non-collaborative tasks, where parameters are specified beforehand by the user, this is not possible. The available dates and times for appointments are dependent upon several factors such as the patient's primary care physician, the appointment type (e.g., initial visit, wellness, sick visit), and the availability/preference of the caller (the patient). Because these factors are not initially available/accessible and may need to be negotiated with the caller, it is impossible to build a workable task in a batch style. With collaborative tasks, the system can be taught a task where execution proceeds to the point where the list of available appointment dates and times becomes available. At this point, the user is assigned an action of choosing the desired appointment time, and then the system can finish the remaining steps of the booking task.

Note that NL and the system's contextual understanding play a key role for the system to learn collaborative tasks. For the above appointment selection example, during the teaching session, a user can (i) indicate when the user plans to take an initiative by saying, "*Let me choose an appointment*" and (ii) let the system know when his/her actions are completed by saying, "*I'm done*". During execution, at a step assigned to a user, the system would signal that it is the user's turn by saying something like "*I am waiting*" After the user's turn is complete, he/she would say "*I'm finished*" so the system could resume the task execution. A user does not necessarily have to always make such a closing remark for every user initiative step. If the system learned that a step was to simply select a row (like the case in Figure 4) from user's NL description (e.g., "*Let me select a row for him/her*") and related mouse/keystroke action(s), the system can figure out when the user is done by comparing observed actions in learning with the current user actions in execution.

## Evaluation

In our empirical evaluation, we were interested in two issues: (1) are people who have no programming experience able to successfully teach tasks to the system? and (2) can the executing of taught tasks improve the average call time for appointment booking? To test both of these, we conducted an experiment with two Navy Corpsmen from Naval Hospital Pensacola, who were regular users of CHCS but had no previous programming experience. Although the system does support speech input, in order to more closely control the system robustness, the Corpsmen were taught to interact with the system solely through the typed text interface. The following sections describe the experiment and then discuss the results.

### Teaching Evaluation

The Corpsmen were each trained for four hours on the system. They were then each given three tasks to teach the system and four hours to complete them. The three evaluation tasks were (1) *Find a patient's phone number*, (2) *Start booking an appointment*, and (3) *Book a primary care appointment*. The separation of the last two tasks is partly an artifact of the way the CHCS booking process works. All appointment booking shares a common set of preliminary tasks of navigating to the booking subsystem, searching for and selecting the correct patient, and then navigating to the appointment booking screen (which is shown in Figure 1). This task is the second task listed above.

Once that is done, there are several distinct ways an appointment is booked, depending mostly on the type of appointment to be booked. The third task listed above is for booking a patient an appointment with their primary care provider. Other appointment types (such as with a specialist), are done through other subsystems and require different tasks to be taught. The Corpsmen were given a written description of the tasks to teach and were allowed to ask any clarification questions about the tasks. They were then given four hours to teach the tasks in any order they wished. Both Corpsmen finished teaching the three tasks in less than two hours. After the teaching session, the tasks taught by the Corpsmen were evaluated for correctness. All three tasks were successfully taught by both Corpsmen.

### Execution Evaluation

After the teaching evaluation, a separate evaluation was performed to measure the impact of task use on the time spent on booking phone calls. Two confederates (who were not involved in the project) took turns making simulated phone calls to the Corpsmen in order to either book appointments as fictitious patients or as fictitious doctors wanting to look up a patient phone number. Three sets of 25 calls were made to each Corpsman. The first five calls of each set were used as a warm-up, and were not counted in the overall evaluation. In one set of calls, the Corpsmen used only the CHCS system to accomplish the tasks as they would normally do. In another set, the Corpsmen used PLOW with the tasks that they themselves taught in the task teaching phase. In the final set, they used tasks taught by researchers (who we are terming as "experts" in system use).

Table 2 shows the average time of calls for each of the three sets performed with remote access to the actual CHCS system in use. The average time for calls with the Corpsman-defined tasks was 8.7 seconds longer than without PLOW. Calls using expert-defined tasks were marginally faster than the calls with the Corpsman tasks, but still 6.0 seconds slower than calls without PLOW. Our analysis showed that the reason for this slowdown was primarily three factors. First was the system's timeout-based method for recognizing when the screen has finished updating (described in Section "*Screen Update Modeling*"). Almost every step had a major screen update to be checked and the system waited 250 ms before it started checking if the current screen matched the expected state, causing significant delay.

The second factor was frequent network slowdowns that caused long pauses (some where over 1 second long) in the middle of a screen update. When the Corpsmen used the CHCS system alone, they often proceeded to the next step as soon as they had enough information, even if the screen was not fully loaded. In contrast, the system had to wait for the update to finish before moving on. Furthermore, the intermittent screen updates increased the chance of false positives that made the system prematurely move ahead and have a failure in the next step. A failure recovery mechanism ensured the system would be able to re-synchronize the state of the task's execution with the state of the application; thus, the system was able to complete the tasks, but at additional time expense. In addition, only government-furnished computers were able to access to CHCS network. The computers used for the evaluation had 2.0GHz Intel Core Duo processors with only 1GB RAM and were measurably slower than our own off-the-shelf test machines.

We performed a post-evaluation analysis of the timing in log data from the evaluations. Based on the analysis, we (conservatively) re-estimated the time of each call assuming correct screen state recognition was implemented as well as assuming the system was running on our own test comput-

| | Without System (Baseline) | Own Taught Task | Difference from Baseline | Expert Taught Task | Difference from Baseline |
|---|---|---|---|---|---|
| **Actual Ave. Call Time** | 55.7s | 64.4s | +8.7s | 61.7s | +6.0s |
| **Adjusted Ave. Call Time** | 55.7s | 49.4s | -6.3s | 46.7s | -9.0s |

Table 2: Average call times for execution evaluation

ers. The resulting adjusted average call times are shown in the second line of Table 2 and result in significant improvements over the previous calls. Our estimated adjusted call times were 6.3 seconds faster than just CHCS for Corpsman tasks and 9.0 seconds faster for expert tasks.

### Discussion

Due to the small size of the evaluation (the number of subjects, tasks, and calls), and the system slowdown problems, it is impossible to make definite statements about the system and its usability. However, the evaluation does show that it is possible for at least some non-programmers to effectively teach tasks to the system. It also seems to show that it may be possible to speed up appointment booking phone calls using tasks that are taught by the users themselves.

## 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 (Angros et al. 2002; Lau and Weld 1999; van Lent and Laird 2001; Faaborg and Lieberman 2006). However, a significant drawback of these approaches is that they require multiple examples, making them infeasible for one-shot learning.

Researchers also investigated techniques that do not require observation. For instance, (Garland, Ryall, and Rich 2001) proposed techniques to encode experts' knowledge with annotation. A scripting system with pseudo natural language was also developed to automate online tasks (Leshed et al. 2008). While these approaches are useful, without the help of demonstration observation, the task learning can be difficult for complex control constructs such as iteration and dynamic web object identification. Furthermore, the coverage of the pseudo languages used in these approaches is limited to handle complex control structures.

Note that task learning approaches described above are not designed to learn collaborative tasks and their application domain is either Web or special simulators. In contrast, our system works together with a user to learn collaborative tasks and the understanding of the terminal interface can be applied to other systems with similar textual interfaces.

## Conclusion

Collaborative tasks allow a wide range of behavior to be learned and performed by a system. This paper shows the possibility for non-programmer users to teach a system collaborative tasks with relatively little training. NL plays a critical role in making the interaction between a user and a learning system intuitive and non-intrusive. While the evaluation was done on a small scale, it revealed problems (to be fixed with better engineering) as well as great potential of this NL-based collaborative task learning.

## Acknowledgments

We would like to thank Naval Hospital Pensacola for their assistance with this project.

This work was supported in part by DARPA grant NBCH-D-03-0010 under a subcontract from SRI International.

## References

Allen, J.; Byron, D.; Dzikovska, M.; Ferguson, G.; Galescu, L.; and Stent, A. 2000. An architecture for a generic dialogue shell. *Journal of Natural Language Engineering* 6(3):1–16.

Angros, Jr., R.; Johnson, W. L.; Rickel, J.; and Scholer, A. 2002. Learning domain knowledge for teaching procedural skills. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*.

Chambers, N.; Allen, J.; Galescu, L.; Jung, H.; and Taysom, W. 2006. Using semantics to identify web objects. In *AAAI-06*.

Faaborg, A., and Lieberman, H. 2006. A goal-oriented web browser. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.

Garland, A.; Ryall, K.; and Rich, C. 2001. Learning hierarchical task models by defining and refining examples. In *International Conference on Knowledge Capture*.

Jung, H.; Allen, J.; Galescu, L.; Chambers, N.; Swift, M.; and Taysom, W. 2008. Utilizing natural language for one-shot task learning. *Journal of Logic and Computation* 18(3):475–493.

Jung, H.; Allen, J.; de Beaumont, W.; Blaylock, N.; Ferguson, G.; Galescu, L.; and Swift, M. 2010. Going beyond PBD: A play-by-play and mixed-initiative approach. In Cypher, A.; Dontcheva, M.; Lau, T.; and Nichols, J., eds., *No Code Required: Giving Users Tools to Transform the Web*. Morgan Kaufmann Publishers. To appear.

Lau, T., and Weld, D. 1999. Programming by demonstration: An inductive learning formulation. In *International Conference on Intelligent User Interfaces*.

Leshed, G.; Haber, E.; Matthews, T.; and Lau, T. 2008. Coscripter: Automating and sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.

van Lent, M., and Laird, J. 2001. Learning procedural knowledge through observation. In *International Conference on Knowledge Capture*.