PLOW: A Collaborative Task Learning Agent

James Allen, Nathanael Chambers[†], George Ferguson[‡], Lucian Galescu, Hyuckchul Jung, Mary Swift[‡], William Taysom

Institute for Human and Machine Cognition, 40 S Alcaniz St, Pensacola, FL
[†]Dept. of Computer Science, Stanford University, Stanford, CA
[‡]Dept. of Computer Science, University of Rochester, Rochester, NY
jallen@ihmc.us

Abstract

To be effective, an agent that collaborates with humans needs to be able to learn new tasks from humans they work with. This paper describes a system that learns executable task models from a single collaborative learning session consisting of demonstration, explanation and dialogue. To accomplish this, the system integrates a range of AI technologies: deep natural language understanding, knowledge representation and reasoning, dialogue systems, planning/agent-based systems and machine learning. A formal evaluation shows the approach has great promise.

Introduction

To further human-machine interaction, there is a great need to develop collaborative agents that can act autonomously yet still collaborate with their human teammates. In this paper we present PLOW, an intelligent automated software assistant that helps people manage their everyday tasks; our focus will be on how such agents can acquire the task models they need from intuitive language-rich demonstrations by humans. PLOW uses the same collaborative architecture to learn tasks as it does to perform tasks. The system displays an integrated intelligence that results from sophisticated natural language understanding, reasoning, learning, and acting capabilities unified within a collaborative agent architecture. In this paper, we report on recent work on how this agent combines its capabilities to learn new tasks. A recent evaluation shows great promise: our system can quickly learn new tasks from human subjects using modest amounts of interaction.

Background

In previous work, researchers have attempted to learn new tasks by observation, creating agents that learn through observing an expert's demonstration (Angros et al. 2002, Lau & Weld 1999; Lent & Laird 2001). These techniques require observing multiple examples of the same task, and the number of training examples required increases dramatically with the complexity of the task. To be effective, however, collaborative assistants need to be able to acquire tasks much more quickly – typically from a single example, possibly with some clarification dialogue. To enable this, in our system the teacher not only demonstrates the task, but also gives a "play-by-play" description of what they are doing. This is a natural method that people already use when teaching other people, and our system exploits this natural capability. By combining the information from understanding with prior knowledge and a concrete example demonstrated by the user, our system can learn complex tasks involving iterative loops in a single short training session.

Task Learning Challenges

As in all learning, a core challenge in task learning is identifying the appropriate generalizations, which are key to allow the learned task model to be successfully applied to new arguments in new contexts. When viewing this as a traditional learning-by-observation problem using traces of executions, an additional challenge involves identifying parts of the task models that are not explicit in the observed actions. For instance, if a user executes a conditional step equivalent to the form, "if A then B else C", then the trace will only contain one action, B or C, depending on whether the "hidden" condition A is true. Even if we have multiple training examples we still only observe that either B or C is performed. Identifying that A is the relevant condition from the context is an exceptionally hard problem that requires considering the entire context of the demonstration as training data.

In our approach, we identify the condition A from one example *because the demonstrator often mentions the condition in the play-by-play commentary*. When viewed from the perspective of human learning and/or integrated agents, this is a completely natural and ordinary example. Our agent fuses the information from the language understanding with the observed demonstration to construct these complex task models.

There are many other aspects of tasks that are difficult to learn but become much easier with a play-by-play. Some of the key ones include:

- *Identifying the correct parameterization*: is a value in the training an example of a parameter value or a constant? What is the relation between the values used in the training? What is the input or the output?

- *Identifying the boundaries of iterative loops*: is an observed action the start of a loop, a normal step, or the end of a loop?
- *Loop termination conditions*: what is the condition that caused the termination of the loop?
- *Hierarchical structure*: what subtasks were performed in the demonstration?
- *Task goals*: what is the end goal of the task?

In all these cases, the user frequently provides exactly the information that is needed in their running play-by-play. By combining language understanding and learning from examples, PLOW can identify intended procedures from just a single demonstration.

To give an idea of what tasks we are trying to learn, Figure 1 shows ten questions that were used in the system evaluation. The tasks were designed by an outside group and unknown to the developers prior to the test. We will discuss how well the system did later in the paper.

1. What <businesses> are within <distance> of <address>?
2. Get directions for <integer> number of restaurants within <distance> of <address>.
3. Find articles related to <topic> written for <project>.
4. Which <project> had the greatest travel expenses between <start date> and <end date>?
5. What is the most expensive purchase approved between <start date> and <end date>?
6. For what reason did <person> travel for <project> between <start date> and <end date>?
7. Find <ground-transport, parking> information for <airport>.
8. Who should have been notified that <person> was out of the office between <start date> and <end date>?
9. Summarize all travel and purchase costs for <project> between <date> and <date> by expense category
10. Which projects exceeded the current government maximum allowable expense for travel costs?

Figure 1: Previously unseen tasks used in the evaluation

The PLOW System

While language greatly enhances the training, this is not to say that task learning becomes easy to accomplish. To create an effective learning system, we need to integrate deep language understanding, reasoning, dialogue and machine learning, integrated within a collaborative agent architecture. This section gives a brief overview of the system.

The Interface

PLOW learns tasks that can be performed within a web browser. These are typically information management tasks, e.g., finding appropriate sources, retrieving information, filing requisitions, booking flights, and purchasing things. Figure 2 shows PLOW's user interface. The main window on the left shows the Mozilla browser, instrumented so that PLOW can monitor user actions. On the right is the procedure that PLOW has learned so far, summarized back in language from the task model using the system's language generation capabilities. Across the bottom is a chat window that shows the most recent interactions. The user can switch between speech and keyboard throughout the interaction.

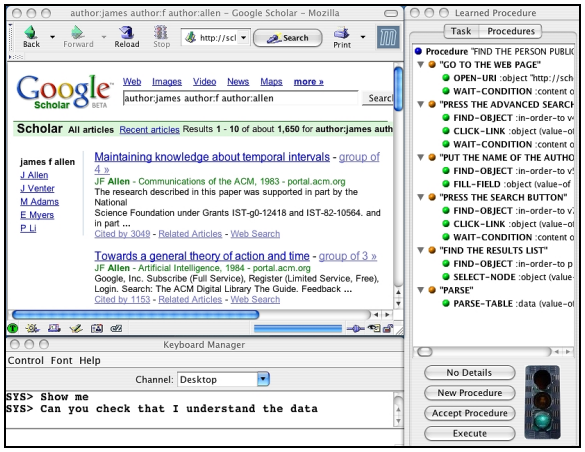

Figure 2: The PLOW Interface

The Agent Architecture

A high-level view of the PLOW agent architecture is shown in Figure 3. The understanding components combine natural language (speech or keyboard) with the observed user actions on the GUI. After full parsing, semantic interpretation and discourse interpretation produce plausible intended actions. These are passed to the collaborative problem solving (CPS) agent, which settles on the most likely intended interpretation given the current problem solving context. Depending on the actions, the CPS agent then drives other parts of the system. For example, if the recognized user action is to demonstrate the next step in the task, the CPS agent invokes task learning, which if successful will update the task models in the knowledge base. If, on the other hand, the recognized user intent is to request the execution of a (sub)task, the CPS agent attempts to look up a task that can accomplish this action in the knowledge base. It then invokes the execution system to perform the task. During collaborative learning, the system may actually do both – it may learn a new step in the task being learned, but because it already knows how to do the subtask, it also performs that subtask for the user. This type of collaborative execution while learning is critical in enabling the learning of iterative steps without requiring the user to tediously demonstrate each loop through the iteration.

Language Processing

Language understanding and dialogue management is accomplished using the TRIPS system (for details, see Allen et al. 2001, Ferguson & Allen 1998), which provides the architecture and domain-independent capabilities for supporting dialogue-based, mixed-initiative problem solving in a range of different applications and domains. Its cen-

tral components are based on a domain-independent representation, including a linguistically based semantic form (the Logical Form, or LF), illocutionary acts, and a collaborative problem-solving model. Domain independence is critical for portability between domains: the system can be tailored to individual domains through an ontology mapping between the domain-independent representations and the domain-specific representations (Dzikovska et al. 2003).

The parser uses a broad coverage, domain-independent lexicon and grammar to produce the LF, a detailed description of the semantic content of the input. The LF is a flat, unscoped representation that supports robust parsing and interpretation. It consists of a set of terms that describe the objects and relationships evoked by an utterance. The LF includes surface speech act analysis, dependency information, and word senses (semantic types) with semantic roles derived from a domain-independent language ontology. The parser disambiguates word senses based on linguistically motivated selectional restrictions encoded in the semantic types and roles. Word senses that have domain-specific mappings are tried first by the parser to improve parsing speed and accuracy.

The TRIPS system has been described in detail previously, so we will not go into more detail here. Also not discussed are the speech recognition issues, where we use dynamically changing language models that continuously add new words found in the context (details forthcoming), and our generation capability, which combines symbolic and statistical models to produce real-time broad coverage generation (Chambers 2005). This paper focuses on how deep understanding of language is used for task learning, rather than how language processing is accomplished.

Instrumentation

To enable effective learning from demonstration, getting the right level of analysis for the instrumentation is critical. It is relatively easy, for instance, to obtain a trace of the mouse movements and gestures that a user performed, but this does not help to understand the significance of what action is being performed, nor can it be used to execute the task later in a different context. Similarly, tracing the back-end API calls of the GUI identifies what the user did, but not how they did it. This makes it impossible to identify the critical decision-making processes needed to handle new examples.

Our instrumentation takes the middle ground. We interpret user actions in terms of the underlying DOM representation that the browser uses to generate the GUI contents of the window (e.g., clicking buttons, filling in fields, selecting objects on screen) and we record key events that occur in response (e.g., new page loaded). If the web consisted of pages derived from well designed basic HTML sources, this would be fairly straightforward. But life is not like that. Significant effort is needed to handle the range of phenomena found in real-life web pages.

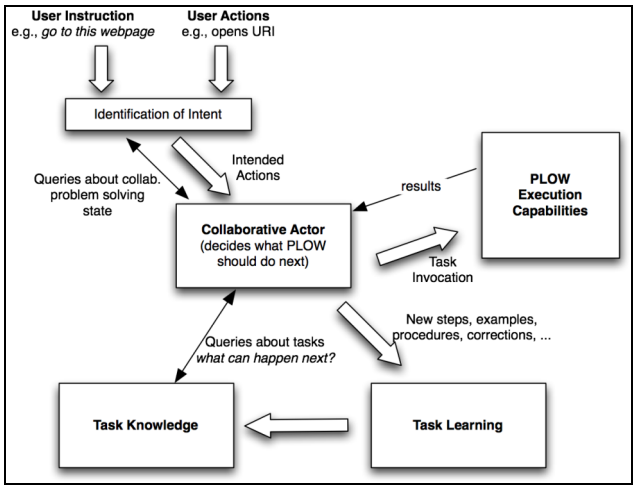

Figure 3: The Collaborative Agent Architecture

Learning Tasks

This section considers a series of challenges in task learning and shows how PLOW addresses these issues using a combination of language understanding, intention recognition, reasoning and machine learning. The problems can be divided into learning perception/actuation actions (i.e., "primitive" or "non-decomposable" actions), and learning the higher-level control structures, including identifying hierarchical structure, control structures, and parameterization.

Learning Primitive Actions

Consider a situation in which the user is teaching PLOW how to fill in a search form at a bookseller's site (perhaps as part of a book-buying task); the user says, "*Put the name here*" and then fills in the author's name in the form field as shown in Figure 4 (left-hand side). The results from the language understanding and GUI instrumentation are also shown.

PLOW now must combine the information about what the user said and the action they performed in order to learn a useful generalization. It starts a heuristic search through the DOM representation starting from the clicked node, looking for labels and tags that might be related to the semantic concept *FULLNAME, produced by the parser. In this case, it finds a text node with the content "Author's name" in a nearby subtree (the search space is marked with a dotted line in the figure). Using the lexicon and ontology, it can easily determine that "name" is relevant to the concept *FULLNAME. In other cases, it might find a label "author" or some other semantically related concept and identify that. The heuristic search combines a structural distance metric (applied to the path from the selected node to the candidate node) and a semantic metric (the semantic similarity of the label to the concepts in the utterance) and returns the best match. If no reasonable

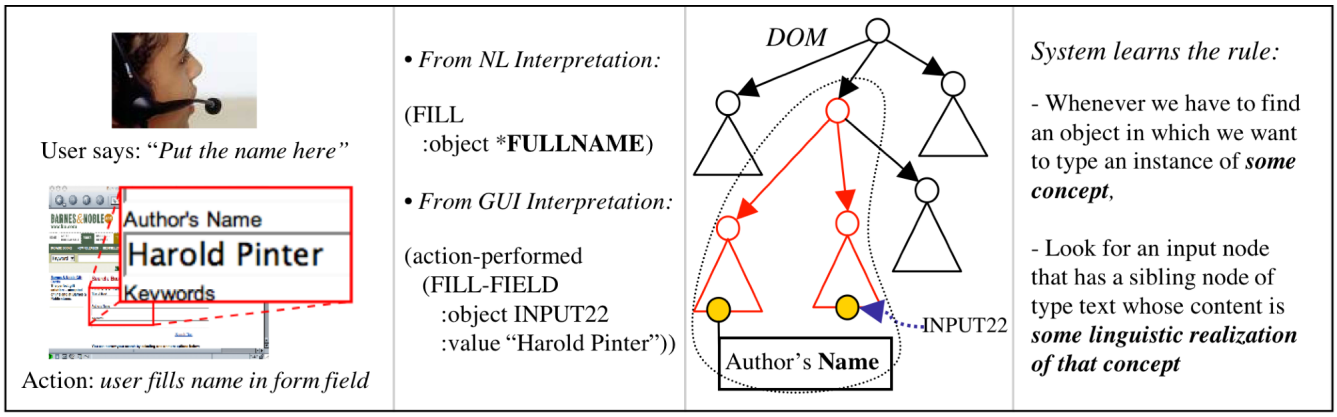

Figure 4: Learning to find and fill a text field

match is found, it resorts to using just structural properties of the DOM tree.

PLOW uses the best match to synthesize a retrieval rule for future use in finding the text field. It attempts to produce a rule that generalizes away from the specific ontological concept *FULLNAME and could apply to any concept (a natural language gloss of the rule is shown on the right hand side of Figure 4).

After learning this rule based on a single example, PLOW can not only perform the action "find the author field" on this site, but actually can find other text fields on the site (for example, the book title field). In an evaluation of this technique, we determined that after learning how to find the "books" tab, it successfully found other tabs 95% of the time on Barnes & Noble's website, and 98% on Amazon (for details and other examples, see Chambers et al, 2006).

Learning Effective Parameterization

One of the main challenges to learning even simple straight-line procedures is identifying the appropriate parameterization. When an object is used in a demonstration, the system has to be able to determine whether it is simply being used as an example *and* as an input parameter, whether it should be a constant in the procedure, or whether is has some relational dependency to other parameters already in the procedure. In addition, PLOW must determine which parameters are needed as the output parameters of the procedure.

With traditional techniques for learning from observation, it is impossible to identify such information reliably from one example. With additional information from language, however, we can generalize from one example quite effectively. Figure 5 shows excerpts from an actual dialogue for finding hotels near an address and the key features PLOW used to derive its interpretation. First, much information can be obtained from language through the definiteness feature. An indefinite noun phrase such as "*an address*" is very likely to be an input parameter, and a definite noun phrase is not. In general, definite noun phrases are resolved using TRIPS' reference resolution capability, capability, connecting the same instances of the parameters as they are used in the task. In the case of "*the zip code*", the reference resolution component handles the bridging reference using ontological information to interpret this as the zip code of the previously mentioned address.

Learning Hierarchical Structure

For the challenge of identifying the appropriate task hierarchy, the PLOW system uses a simple strategy for identifying the beginning of new subprocedures: Any statement that explicitly identifies a goal, e.g., "*Now let me show you how to …*" or "*Now we need to find the zip code*", is treated as the beginning of a new procedure to accomplish the mentioned goal. In order to work effectively, however, the user needs to explicitly indicate when the subprocedure is completed (e.g., "*We're done here*" or "*We're done finding the zip code*"). This requirement may not be completely natural, but we have found anecdotally that it is easy to pick up and remains intuitive.

Learning Iteration

Learning iterative procedures in one shot is a significant challenge for several reasons, including the fact that users

Utterance	***Interpretation***	***Key Features***
Let me show you how to find hotels near an address	*hotels* -> Output Parameter (list of hotels)	- Bare plural - Object of information-producing action "find"
	an address -> input parameter of type address	- Indefinite - No deictic action
Put hotels here	*hotels* -> constant	- Bare plural - Identical to the text typed in the user action
Put the zip code here	*the zip code* -> function on address parameter	- Definite reference - Ontology (zip code is role of address)

Figure 5: Interpreting Noun Phrases

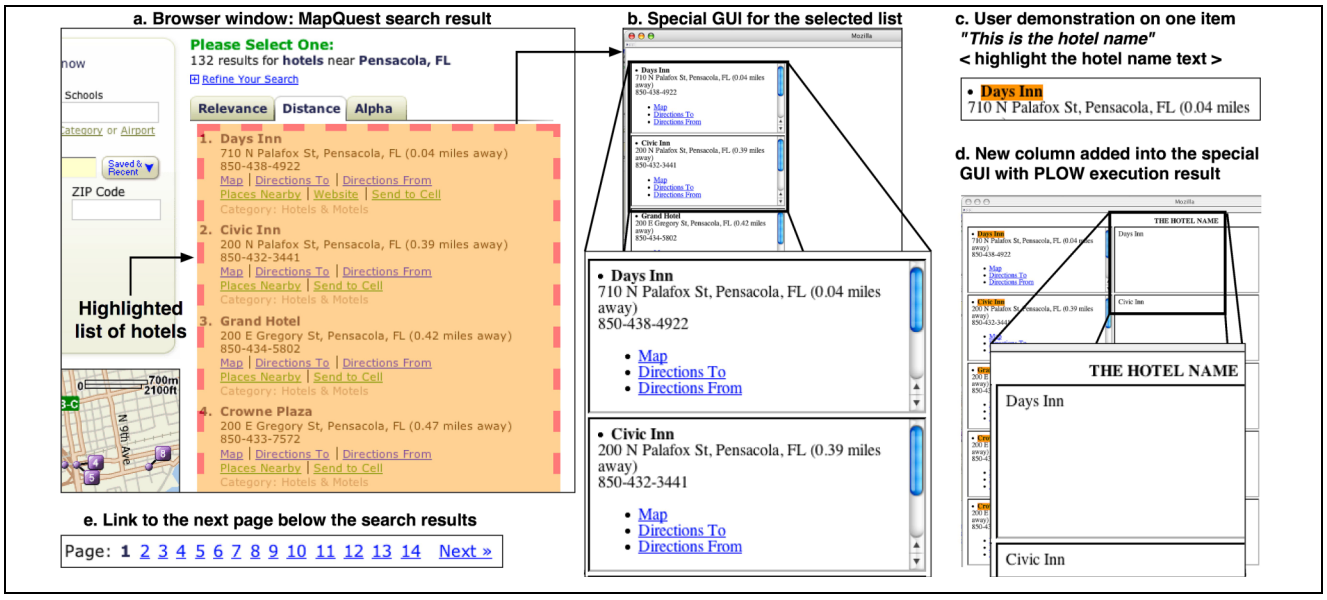

Figure 6: Learning Iterative Steps

typically do not want to demonstrate lengthy iterations. Unfortunately, task learning algorithms based on pure demonstration require many examples in order to induce the intended control structure. Again, language comes to the rescue, but we found this was not enough. To create a useful user experience, we needed to make the learning system much more active, using dialog to obtain additional information when needed.

When learning iterations, PLOW takes the initiative to help the user execute the task being demonstrated. For instance, after the user identifies the initial step of an iteration over a list, PLOW attempts to repeat the step over the rest of the items in the list. This provides a natural opportunity for the user to immediately correct problems that arise from generalizing from a single training example. Figure 6 shows an example. The user is teaching PLOW how to find hotels near an address. The user starts by highlighting a list of results (Figure 6a) and saying "*here is a list of results*." From the fact that the user identified a list, PLOW posits that the user is defining an iteration over the items in the list. It invokes its ability to parse the DOM tree and identify the individual elements, presenting the parsed results to the user (Figure 6b). The user can now operate on individual elements. In the example, the user highlights the title of the hotel and says, "*This is the hotel name*" (Figure 6c). Given this, PLOW then learns the extraction pattern and applies the rule to all the other elements, showing the results in a new column (Figure 6d). The user can browse through the results; if they find an error (typically that no information was extracted for an element), the user can signal an error (e.g., say "*this is wrong*") and then provide another example. PLOW then learns an extraction rule from these two examples. This process can continue, with the user providing more examples until a comprehensive pattern is learned.

In addition, the user can teach the system how to iterate over pages of information by identifying the element for the next page (e.g., "*Click the next button for more results*" – see Figure 6e). In this case, PLOW does not know the duration of the iteration and will ask the user how many results they wish to find. The system understands a range of responses including "*get two pages,*" "*twenty items*" and "*until the distance is greater than 2 miles*." The latter makes sense only if the user has taught the system how to extract the distance from the elements in the iteration, and it is supported by the system's ability to interpret extracted information (e.g., the text "2.56 miles") on the fly.

Using this mixed-initiative form of learning, we have found users can easily define simple iterations in a way that is fairly intuitive and easy to use.

Evaluation

While we have shown examples of how integrating language, dialogue, reasoning and learning has great potential for effective one-shot task learning, the real test is whether ordinary users can quickly learn to use the system to teach new procedures. There are many issues to be concerned about: (1) Do we have comprehensive enough natural language understanding capabilities so that users expressing information in intuitive ways are likely to be understood? (2) Can the system really learn robust task models from a single example? (3) Can the users easily determine whether the system is learning correctly as they are teaching the system? Significant engineering went into the PLOW 2006 system to address these issues.

In August 2006, we delivered a version of the PLOW system to independently contracted evaluators. At that point, we had developed the system to ensure that we (the developers) could effectively teach PLOW to learn how to answer seventeen pre-determined test question templates.

The evaluators recruited 16 test subjects, who received general training on how to use PLOW and many other applications that were part of the overall project evaluation. Among these were three other task learning systems: one learned entirely from passive observation; one used a sophisticated GUI primarily designed for editing procedures but extended to allow the definition of new procedures; and the third used an NL-like query and specification language that required users to have a detailed knowledge of HTML producing the web pages.

After training, the subjects then performed the first part of the test, in which they had to use different systems to teach some subset of the predefined test questions. Seven of these involved the PLOW system. Once the procedures were learned by the systems, the evaluators created a set of new test examples by specifying values for the input parameters to the task and then scored the results from executing the learned task models using predetermined scoring metrics individualized to each question. The PLOW system did well on this test, scoring 2.82 out of 4 across all test questions and the 16 subjects.

The second part of the test involved a set of 10 new "surprise" test questions not previously seen by any of the developers (see Figure 1). Some of these were close variants of the original test questions, and some were entirely new tasks. The sixteen subjects had one work day to teach whichever of these surprise tasks they wished, using whichever of the task learning systems they wished. As a result, this test reveals not only the core capability for learning new tasks, but also evaluates the usability of the four task learning systems.

PLOW did very well on this test on all measures. Out of the 16 users, thirteen of them used PLOW to teach at least one question. Of the other systems, the next most used system was used by eight users. If we look at the total number of tasks successfully taught, we see that PLOW was used to teach 30 out of the 55 task models that were constructed during the day (see Figure 7). Furthermore, the tasks constructed using PLOW received the highest average score in the testing (2.2 out of 4).

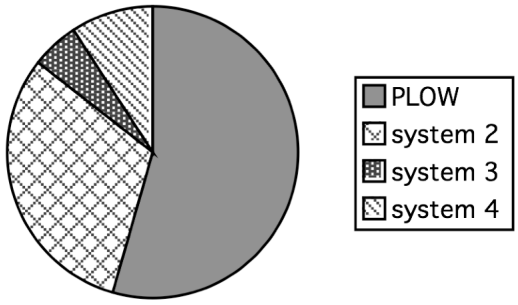

Figure 7: Systems Selected in Teaching Surprise Tasks

Concluding Remarks

By integrating natural language understanding, reasoning, learning, and acting capabilities, it is possible to develop systems that can learn complex task models in one short session with a teacher, using an intuitive language-based interface that requires relatively little training to use. While the PLOW system was preferred over the alternatives in the test, this does not mean, however, that the subjects found any of the current systems easy to use. Sixteen subjects put in a full work day to teach only 55 task models (just over 3 tasks per subject). Because of robustness issues in the software, and difficulties in being able to effectively confirm understanding as the tasks were being taught, the users had to persevere to construct good task models. Much of this, we believe, will be fixed by better engineering, improvements in the learning algorithms, and better mechanisms for displaying the information that PLOW has learned throughout the interaction.

References

Allen, J.; Byron, D.K.; Dzikovska, M.; Ferguson, G.; Galescu, L.; and Stent, A. 2001. Towards Conversational Human-Computer Interaction. *AI Magazine,* 22(4):27-37.

Angros, R.; Johnson, L.; Rickel, J.; and Scholer A. 2002. Learning Domain Knowledge for Teaching Procedural Skills, *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems.*

Blythe, J. 2005. Task Learning by Instruction in Tailor. *Proceedings of the International Conference on Intelligent User Interfaces.*

Chambers, N. 2005. Real-Time Stochastic Language Generation for Dialogue Systems. *Proceedings of the European Workshop for Natural Language Generation.*

Chambers, N.; Allen, J.: Galescu, L.; Jung, H.; and Taysom, W. 2006. Using Semantics to Identify Web Objects. *Proceedings of the National Conference on Artificial Intelligence.*

Dzikovska, M.; Allen, J.; and Swift, M. 2003. Integrating linguistic and domain knowledge for spoken dialogue systems in multiple domains. In Proceedings of IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems.

Ferguson, G., and Allen, J. 1998. TRIPS: An Integrated Intelligent Problem-Solving Assistant. *Proceedings of the National Conference on Artificial Intelligence.*

Lau, T.; Bergman, L.; Castelli, V.; and Oblinger, D. 2004 Sheep Dog: Learning Procedures for Technical Support, *Proceedings of the International Conference on Intelligent User Interface.*

Lau, T. and Weld, D. 1999. Programming by Demonstration: An Inductive Learning Formulation. *Proceedings of the International Conference on Intelligent User Interfaces.*

Lee F.; and Anderson, J. 1997 Learning to act: Acquisition and Optimization of Procedural Skill, *Proceedings of the Annual Conference of the Cognitive Science Society.*

Lent, M. and Laird, J. 2001. Learning Procedural Knowledge through Observation, *Proceedings of the International Conference on Knowledge Capture.*