Massachusetts Institute of Technology

Department of Electrical Engineering and Computer Science

Proposal for Thesis Research in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

Title: Natural Language Assistant for Routine Web Tasks

Submitted by: Gabriel Zaccak _____

(Signature of author)

Expected Date of Completion: May 31, 2015

Laboratory where thesis will be done: CSAIL

Brief Statement of the Problem:

The Web is an integral part of our lives more than ever, not only for accessing information but for performing many of our real-life tasks. We book flights, rent films, pay our bills, manage our bank accounts, access our calendars and order groceries using Web applications that utilize an enormous, constantly growing collection of Web services. Google, Yahoo, and other search engine providers are extensively researching new ways to enhance user experience by managing and utilizing these services to offer new services beyond search to the future searcher. For example, Hipmunk[1] offers a sophisticated and uniform searching interface for flights by utilizing different Web services from different airlines websites. As another example, Google Ride Finder[2] allows users to find a taxi, limousine, or shuttle using real time position of vehicles. To do that, Google Ride Finder uses Google Maps Web services for the location functionality and taxi reservation Web services.

So far, users interact manually with these services through a Web browser. While such manual interactions often enable us to complete our tasks successfully, the process is very repetitive and is tiresome. Users are forced to repeatedly perform the same sequence of

[1] http://fwww.hipmunk.com

[2] http://labs.google.com/ridefinder

operations: going to the same Web pages, scrolling to the same positions, filling in the same forms fields, and following the same hyperlinks. Moreover, users are often unaware of the wealth of web services that exist on the Web and are unable to combine several web services into a new task.

To reach better user interaction and sophisticated utilization of web services to perform our daily tasks, a shift in user paradigm is necessary. We believe that a language-based interface is an ideal candidate for this shift. Natural language is well suited for humans and is expressive, concise and easy to use. To access Web applications through natural language we need tools that automatically process and "understand" Web applications and reason about their underlying data and services. In addition, the natural language interface needs to interpret user requests concerning any topic, allow unrestricted user interactions, and handle complex requests involving multiple services. Current efforts in natural language interfaces to Web application provide information seeking solutions but require adaptations to handle new domains. Their Web application processing tools only deal with small fractions of the application site and do not represent relationships among the different fractions or between other applications. In this thesis, we propose to design and develop an end-to-end personal assistant dialogue system that helps casual users perform their real life Web tasks. We will build on past research efforts and solve their shortcomings to process and "understand" whole Web applications, model relationships between them to support complex tasks involving multiple services, and design a general framework to support execution of tasks. In addition, to make the interaction more intuitive and fluent, our system will learn users' habits and preferences by observing their browsing actions.

1 Introduction

The World Wide Web is an integral part of our daily lives and it is hard to imagine our world without it. The World Wide Web, originally invented as a tool for communication and information management, is changing the way we seek information, do business, exchange ideas, communicate and socialize with one another, and entertain ourselves. Every day new services become available on the Web involving real life tasks; some are very popular such as renting a car, booking a flight, buying tickets for a show, and ordering food, and some are less common services such as monitoring pets activities[3], buying customized shoes, ordering catering service, or designing a logo for an event.

Currently, the user paradigm is to manually interact with these services through a graphical interface, typically a browser, or through personalized client programs implementing the service's API. Having a task in mind, we go to the service's site either directly entering the URL in the browser or using a search engine to find the service's URL. Usually the process starts by visiting a search engine since the number of services available is already too large for anyone to remember or know of their existence. The interaction with the service is performed using the keyboard to fill in form fields and the mouse to follow links until we have successfully executed the task. While such manual interactions often enable us to complete our tasks successfully, the process is very repetitive and is tiresome. Users are forced to repeatedly perform the same sequence of operations: going to the same Web pages, scrolling to the same positions, filling the same forms fields, and following the same hyperlinks.

In the future, users will interact with services and construct real life tasks through natural language. Natural language is well suited for humans and is expressive, concise and easy to use. Much research is carried out towards natural language interfaces and information access on the Web and more systems are becoming publicly available; however none address general task execution involving manipulation of data and services. Current solutions provide domain specific dialogue systems such as Gruenstein et al.'s multimodal restaurant guide, City Browser [3], or information seeking dialogue systems such as Allen et al.'s PLOW [5].

To better illustrate the problem, let's examine the following complex task where a user would like to arrange his travel plans to New York and make a dinner reservation. Let's assume

[3] http://www.sniftag.com/

that sub-tasks involved are: (1) booking a round-trip flight from Boston to New York for tomorrow, (2) renting a car for one day, and (3) making a reservation at a French restaurant for two. Today, in order to complete the task the user is required to interact with multiple Web applications.

To complete the first sub-task, unless the user is familiar with a specific Web application's site, he typically starts by visiting a search engine site and search for travel services using the task related keywords such as "airline", "book a flight", or "cheap flights". The search engine provides the user with a list of candidates matching the entered keywords. The user can select a Web application to use from the list or refine the keywords for a better search. Among the search engine's results for the "airline" keyword are United Airlines[4], American Airlines[5], and Continental Airlines[6]. Once the user determines which Web application to use, he clicks on the Web application link and gets redirected to the application's site. For this example, the user selects Continental Airlines. On the Continental site, the user identifies the relevant form for the task and starts filing the fields with the appropriate information. He has to fill the following options: (1) either round trip or one way (2) origin (3) destination (4) date and time for departure and return date (in the case of round trip) (5) number of passengers (6) economy, business or first class cabin (7) search option to prioritize price, schedule (8) search for non-stop flights. After filling this information he clicks on the "Search" button to submit the form. Then the Web application displays a list of available flights satisfying the constraints of the user if possible or asks the user to constrain his search if the values are too vague or unavailable. In this case, when the user fills in the destination field "New York", the application will present to the user a clarification dialogue to select which airport he prefers to fly to and submits the form again. Then the user selects from the list of available flights (Figure 1) by clicking on the "Select" button of the corresponding flight. The application has a predefined plan to follow (Figure 2) to perform the user's task and guides the user through it. After selecting the departing and returning flights, it will ask the user to select the preferred seats, his information, and payment method and show a summary or an invoice of the flight for his records.

Having completed the first task, the user needs to find a rental service to rent a car. The time and date of the rental depends on the departing flight arrival and returning flight departure. He starts a similar process searching for a rental service and goes through similar steps to

[4]http://www.united.com
[5]http://www.aa.com/
[6]http://www.continental.com

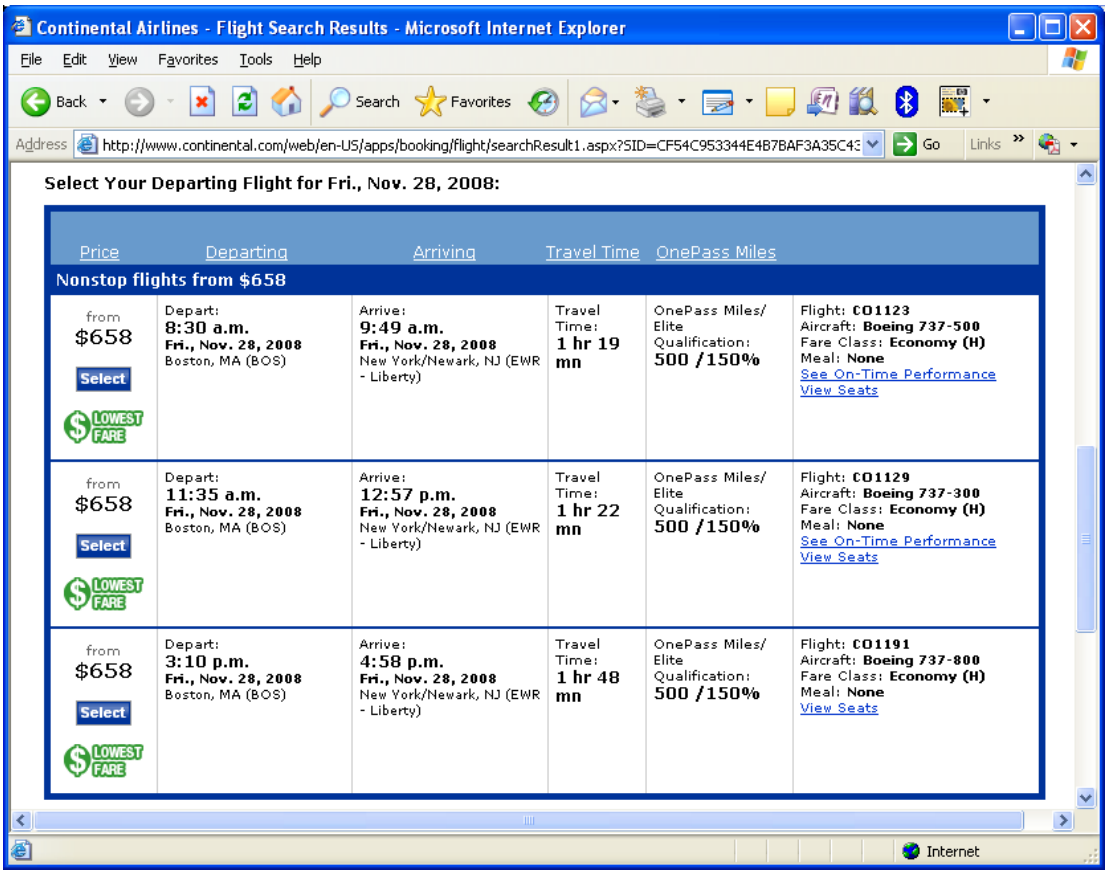

Figure 1: Example of available flights on Continental Airlines site

Figure 2: Continental Airlines Web application workflow for flight booking

execute the car renting sub-task as for the flight booking task. The user can achieve the first two sub-tasks using a vacation service application which combines multiple tasks into one task and manages the time and date dependency between the tasks. Most online travel companies offer such a service and the user can choose package types that include any subset of: booking a flight, renting a car and booking a hotel. In our example, using such a service reduces the user time spent completing the first two tasks by nearly half since most of the information that is required to complete the tasks is the same (origin/destination city, time and date, user information and payment method).

To complete the third sub-task, the user visits again the search engine site and searches for keywords such as "reserve a table" or "restaurant reservation". The search engine provides the user with a list of candidates such as OpenTable[7] and RestaurantReservations[8]. The user chooses the application he prefers and goes through a similar process to the previous tasks but with some different underlying data and constraints. He needs to choose the location, cuisine type, number of people, time of dining, and price range. In addition, the user might visit another site before making his reservation to read a review about a specific restaurant in mind to satisfy other constraints such as ambience and service quality. The additional information can be integrated into the application itself using Web mashup tools such as Intel Mash Maker [16] or Huynh et al.'s Potluck [22]. Mashups let casual users create an augmented Web application that combines data from additional sources. Both user mashups and applications such as the vacation service combine multiple applications to create a single integrated tool. They combine content from the different sources typically through Web services. Web services are APIs for applications over the Web. Web services allow computer programs to access and perform predefined actions on Web applications' underlying data.

Ideally, we should be able to perform the complex three part task described above via natural language. There are multiple ways to phrase such a task but the user can describe it in one sentence, e.g., "Book a flight from Boston to New York tomorrow afternoon returning on Sunday at night, rent a car during my stay, and reserve a table at a French restaurant for party of two at 8pm". To achieve this goal we need to answer the following questions:

- How to interpret and map a user request to a set of tasks?

[7] http://www.opentable.com
[8] http://www.restaurantreservations.com

- How to create a task bank?
- How to represent knowledge and relationships between tasks and sub-tasks?
- How to help users perform their task without disturbing their workflow?

These questions individually correspond to several existing fields of research that have been explored by the academic community; the challenge to create a general solution remains unaddressed. In the following paragraphs we will give a small overview and challenges for each question.

Given the above user's request, we need to interpret and decompose it into sub-goals, interact with the user in case some information is missing, resolve dependencies between each sub-goal, and generate a chronological list of actions with correct parameterization to be executed (Chapter 2). Conversational human computer interaction is an established field of research with the ultimate goal to design and build systems that achieve human conversational performance. In our case, the interaction between the system and the user is focused on accomplishing concrete tasks. Such dialogues are called practical dialogues. Even though practical dialogues cover much of the potential applications for human computer interaction, they do not capture the extent of full human conversation. Therefore, sufficient understanding of such dialogues while still complex, is feasible [1, 6]. In addition, Allen et al. [4] have shown that for practical dialogues, language interpretation and dialogue management components are independent of the domain of the task being performed. They designed and build a generic dialogue framework that can be adapted to new applications by specifying only the domain and task models. We believe that since our tasks are Web related we can relax the domain independence hypothesis even further and design a generic framework that doesn't need to be adapted for each new domain and uses only one domain-independent ontology.

Typically, a dialogue system deals with only one user request at any moment in time. To handle the complex task as shown in the example above, the system needs to decompose the task into smaller sub-tasks and resolve the various dependencies between those tasks. Katz et al. [25] demonstrated syntactic and semantic strategies to handle such complex questions in START, a high-precision question answering system. After breaking the request into smaller goals, we need to map each sub-request to the relevant task to be performed. Some question answering systems such as START map questions to actions via natural language

annotations, which are content-describing phrases and sentences. Dialogue systems use various parsers to construct a semantic interpretation of the user input. TRIPS [1] and PLOW [5] semantic representation use a linguistically based form, called the Logical Form (LF). Other approaches use a standard procedure language [9, 38], a variant of PRS, and others use proprietary representations. We believe that integrating both language annotations to find the relevant task and semantic parsing to map the request parameters will achieve better abstraction and make the domain independence framework easier to design.

After interpreting the user request, the system needs to have access to a task bank. This leads us to the creation and internal representations of Web tasks in the task bank (Chapter 3.) The tasks can be user fed via learning by observation or automatically generated by crawling and wrapping Web applications. As we have seen from the detailed walk through an earlier example, a lot of the sub-tasks are the same such as user information, and payment method. In our solution we will use such information and leverage previously learned task similarities to achieve a better and faster way to discover and add new tasks to the task bank.

To automatically discover an application's available tasks we need to design and build a site wrapper. The site wrapper component will crawl the application's site, discover the underlying data and actions (Web forms, and links), infer the semantics of the underlying data and actions, and generate a list of tasks from the structured tree of actions. In addition, the component will assign natural language annotations and keywords for each task to be used by the task manager to map a user request to task from the task bank.

A Wrapper can be seen as a set of specialized programs that extracts data from a data source such as a Web site and manipulates the information into a suitably structured format to enable further processing. Previous research efforts on creation wrappers were solely concerned with information seeking and extracting and not with execution of real life tasks on the Web.

Various machine learning techniques were used towards developing automated wrapper generation systems. In addition, wrapper generation systems specialize in particular types of data sources. There are systems that deal with semi-structured data, such a WIEN [29], STALKER [39], and Wrapster [47]. Wrappers are generated from a set of examples by computing a generalization that explains the observations. There are also more general wrapper generation systems, such as RAPIER [11, 12] and WHISK [43] that extract information from

unstructured text data. Given a hand-tagged template, these systems learn pattern-matching rules to extract the tagged information in the template. We will extend our previous work of wrapping semi-structured websites [47] and enhance it to handle Web applications. Web applications are themselves semi-structured in a sense; they present information about objects in a similar structure and, in addition they allow actions to be performed on those objects. In addition, we will explore various deep Web crawling techniques [36] to get a representative sample of the application's site which we will use to discover the site's objects and actions.

What if the user request does not have a corresponding task to execute from the task bank. To handle such a case, the system needs to have a task learning component that observes the user actions and infers the correct parameterization of demonstrated task (Chapter 2.) The problem has been addressed in previous work using various approaches to learn from specific examples provided by users as they perform a task. Such approaches include programming by demonstration, learning apprentices, case-based reasoning, and feature-based induction. The challenge is to build a system which does not disturb the user's workflow and able infer the task parameters from fewer examples. Adaptive Programming Environment (APE), a software assistant[9], uses machine learning techniques to learn user habits by watching what the user is doing, and then offers to complete repetitive tasks on his or her behalf. Blythe [9] developed "Tailor", a task learning system, to modify task information through instruction using standard procedure language. "Tailor" improves Huffman's [21] work by reasoning about an abstract process description and assuming no domain knowledge. Other approaches have used problem-solving methods, interdependency analysis and smart editors to help user work with procedure representation [10, 28]. However, those approaches require users to make implementation-level decisions about changes to procedures, during which it is easy to lose track of changes or make mistakes.

In this thesis, we propose to develop an end-to-end personal assitant dialogue system that helps casual users perform their real life Web tasks. A high-level view of our proposed system architecture is shown in Figure 3. The system will consist of multiple components related to different field of research such as deep language understanding, discourse and dialog, knowledge representation, human-computer interaction, Web information extraction, and discovery of underlying structure of Web applications. The base and important part of our work is the knowledge representation of the Web applications and their underlying data and services so that our system will generalize well to unseen sites and able to handle

[9]http://www.cincomsmalltalk.com

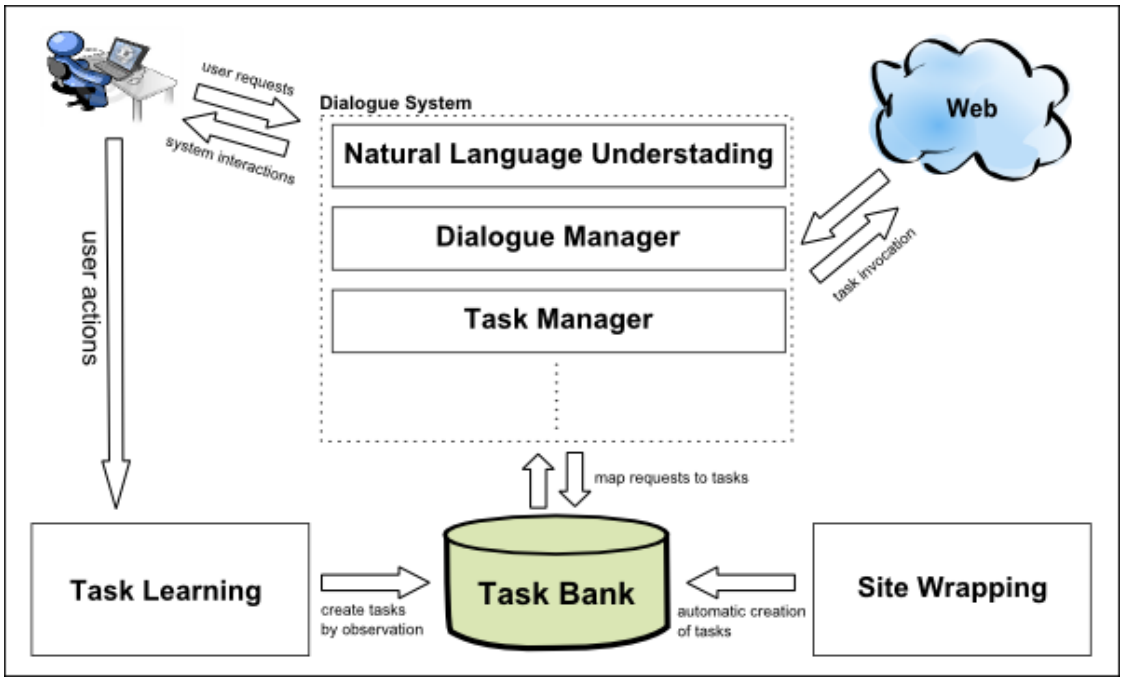

Figure 3: Proposed Framework Architecture

tasks involving multiple Web applications. We will also focus on how the system can learn to perform tasks by observing the user actions and give semantic meaning to those tasks. In addition, we expect that user guidance for learning new tasks will decrease as the system is provided with enough training. Chapters 2 and 3 describe our proposed approach and related work. In chapter 4, we define the thesis's scope and assumptions, list the data that we will evaluate on our system, summarize our expected contributions and present a detailed work timeline.

2 Natural Language Interface

Computer programs that interact with humans whether via speech or text are called dialogue systems. Our goal is to design and build a dialogue system that approach human performance in natural language interaction to perform real life Web tasks. The system will handle requests for any Web application and no initial training is required by the user. With each interaction the system will update the user profile and learn about his preferences and habits. During the task solving process his profile will be used to reduce the number of clarification interactions and make them more personal. In addition to natural language interactions, our solution will leverage the browser display for interactions that are better suited for graphical display such as displaying a list of airports to choose from.

Much research is conducted in the speech recognition community towards developing conversational dialogue systems for various applications and domains. Similar to Allen et al. [6], we are concerned with specific type of dialogue, called practical dialogue, where the system and the user interactions are focused towards accomplishing a task.

User: Book a flight from Boston to New York for tomorrow afternoon
System: Choose which airport in New York you prefer to flight to (the system displays the list of airports in New York in the browser)
User: JFK (the user can also click on the corresponding item from the list displayed in the browser window)
System: Select the return data
User: one-way ticket
System: Select you flight (the system displays a list of flights to choose from)
User: The one leaving at 2pm (the user can also click on the flight he wants in the browser window)
System: (the system shows a confirmation summary for the reservation)

Figure 4: An Example interaction.

Figure 4 shows a typical conversation with the system. Even though it is a simple interaction, it shows most of the capabilities that our solution will cover. From the first sentence, we can see that the system needs to reason time to handle time expressions such as "tomorrow" and "afternoon". We will implement a time conceptual domain model to reason about time. Other conceptual domain models will be added as necessary. In the above interaction, the Web application itself already has a geographical conceptual model and is able to list airports in the vicinity of New York city; therefore, our solution does not need to reason

about locations. In addition, we can notice that the system expects a return date for the travel. This information is encoded in the task description with the task specific properties such as "one-way ticket". Finally, after the user chooses the flight that he prefers, the system did not ask him for the payment method since it has his information already stored in his profile from previous interactions.

We believe that plan-based dialogue system is well suited to for the performance we want to achieve. We will investigate and build a domain independent ontology which does not require adjustments for new domains and uses natural language annotations to map requests to tasks. We will build upon previous work and augment them to support Web tasks. The following modules delineate the dialogue system main components:

1. Interpretation Manager: This component receives user interaction, text and actions, and has the following responsibilities: update the discourse context, semantically parse the sentences, and send them to the task manager. We will experiment with various parser and semantic representations.

2. Task Manager: The task manager (TM) receives the interpreted user requests, breaks them into sub goals and identifies the set of corresponding tasks from the task bank to execute. It matches the user interpreted requests with the tasks descriptions and natural language annotations generated earlier during the site wrapping step. It takes into account discourse history and context and decides which task to execute next. The TM is also responsible to verify that all the parameters to perform the current task are available. It reports successes and failures to the Generation Manager.

3. Generation Manager: This component is responsible for the interactions with the user. There are multiple types of situations where the system needs interact with the user to display task summary/confirmation or address difficulties executing the intended user request. Our system will handle the following cases: if there is no task that matches the user request, if the request has missing parameters, and if the parameters are incorrect in type or value.

2.1 Related Work

2.1.1 Dialogue System

A dialogue system is a computer program that engages in a conversation with a human in natural language. The main components of dialogue systems are natural language understanding, dialogue manager, and natural language generator. Other components might be used depending on the type of input/output of the dialogue system such input decoder for a spoken dialogue system. Dialogue systems are developed for various reasons and they can be classified along the complexity of the tasks they can perform: finite-state, frame-based, sets of contexts, plan-based models, and agent-based models. The finite-state systems, the simplest of them all, are very popular today in voice billing systems. They follow a predefined procedure of prompts for the user. At each prompt they accept a specific type of value.

Next in complexity are the frame-based systems which contain most of the spoken dialogues systems built until now. This approach is based on the slot-filling concepts where slots represent containers for information that must be extracted from the user for the system to perform the action. This approach is domain dependent and handles a single task. It has been used for systems providing information about current movies [13], about train schedules [44], and about weather [48]. The simplicity of such domains makes it possible to build robust language processing capabilities. Most information can be extracted using simple patterns specific to the domain.

The sets of context approach generalizes the frame based approach to support multiple domains. With such systems, the user can book a flight and rent a car. The task is represented as a series of context, each represented using the framed-based approach. In addition, the system needs to identify when the user is changing context or modify a previously discussed context which can be quite challenging. Xu et al. [46] developed a flight scheduling dialogue system where a user can change a first leg of a trip after discussing the second leg.

The last two levels of complexity require the system needs to maintain an explicit model of the tasks and the world and reason about them. The language and the interaction become harder these approach need to model a problem solving process that the user interact with. In the plan-based approach, the dialogues involve interactively constructing a plan with the user. The agent-based approach has in addition to monitor operations in a dynamically

changing world such as emergency rescue coordination.

Dialogue models Traditionally, dialogue systems have distinguished between their different knowledge representations and have identified dialogue models, task models, domain knowledge and user models. Dialogue systems may incorporate some but not all these different models, and no hard boundaries are set between the various models. Furthermore, the variety of dialogue system architectures that incorporate various models, has led to confusion when it comes to the purpose and contributions of specific models. The categorization of the various knowledge representation is more of a framework used to analyze existing dialogue approaches and dialogues systems implementation than a theoretical approach to the development of dialogue system [15].

The dialogue model is the component that is responsible on how to respond to the user based on the user input and dialogue history. Common approaches to dialogue modeling are dialogue grammars and plan-based models. They reflect the behavior of the dialogue system and the types of dialogue it handles. Plan-based approaches try to model the user intentions as goal and infer non-linguistics intentions behind the user utterances. Dialogue grammars are based on adjacency pairs and used in simple human interactions such as information retrieval tasks where the user needs to speak his intentions.

Domain models hold the knowledge of the referred world in the discourse. Having an agreed upon domain knowledge help the system resolve ambiguities and interpret the user request easier. Dalhback et al. [14] make a distinction between domain models and conceptual models. The domain model represents the structure of the world, while the conceptual model represents the conceptual relationship between the objects in the domain. The amount of domain knowledge and type vary from system to system depending on the need and complexity of the system. For example, the dialogue system LINLIN used a both conceptual and domain knowledge. Their domain model contains geographical information and their conceptual model was used to reason between departure times and departure places. Those domains allow the system to answer queries about which is the nearest bus stop to a particular place [18].

Task models are responsible for completing the user task. Dahlback et al. [14] define a task as "some real-world non-linguistic activity that is directed towards achieving a particular goal, and can be broken down into small steps, each having its own goal". The task model is

in charge of the negotiations with the user and deciding whether all the required information to complete is present. Furthermore, Fkycht-Eriksson [17] distinguished between the user's task (e.g. information access from an electronic programmer guide) and the system's task (e.g. controlling a device.) Having an task model separated from the dialogue model can make the dialogue system more fluent and efficient.

User models represent the user's goals and plan, capabilities, attitudes, and knowledge. User models can be used for various purposes, i.e. the system can adapt its interaction depending on the familiarity of the user with the underlying domain. Despite their importance, user models are not common in dialogue systems.

Recently, Allen et al. [5] started en effort towards creating a generic framework for dialogue systems. They introduced Plow, a collaborative task learning agent, which allows the user to teach the computer to perform tasks on the web. Their language and dialogue management is accomplished using the TRIPS system (Allen et al. 2001, Ferguson & Allen 1998). Trips central components are based on a domain independent representation such as a linguistically based semantic form (the Logical Form (LF), 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 [15]. Our system will built on these ideas towards a system that will learn to perform tasks by observing the user activities on the web.

2.1.2 Question Answering

Question answering systems are in a way similar to dialogue systems. Given a collection of documents question answering systems retrieve answers to questions posed in natural language. They return precise answers, as opposed to search engines that return lists of whole documents. Complex question answering require the integration of nuggets of information from multiple data resources.

START[10], a high-precision question answering system, is very interesting for our research since it uses the Web as part of its knowledge base [27, 23, 24]. START uses a system called Omnibase [26] as its uniform access interface to the Web. The main portion of Omnibase

[10] http://start.csail.mit.edu

is a database of scripts that extract information from various websites. Omnibase uses an object property value relational model, and the execution of a script generates the value of a predefined property from a predefined site. A wrapper for a site constitutes all the scripts that belong to it, with their corresponding property names.

Recently, a large number of QA system emerged following two direction. One direction is to built QA system for the TREC QA track [45], training and testing the systems on predefined corpus. They develop their own search engines and answer extraction techniques on top of the corpus. The other direction used the Web as the potential answer source and use generic search engines, such as Google[11], to retrieve information related to the question and post-process Web documents to extract answers for the user questions.

The TREC conference offers an exciting environment for competitive research on Question-Answering. However, the questions that can be answered from the fixed text corpus as in TREC are limited. Search engines quality has improved significantly and they offers a promising source for question answering [41].

Agichtein et al.[2] and Glover et al. [19] presented a technique on how to learn search engine specific query transformations for question answering. Their idea is that the current query interfaces of most generic search engines do not provide enough capabilities for direct question answering in natural language. They transformed the user questions into a certain format which include domain specific information to improve the chances of getting high quality documents from the search engine.

Many of the Web-based QA systems use similar techniques as in the TREC conference [30]. Newer approaches do not use deep natural language parse [40] since such approaches are a very slow and not usable on the Web. They introduced a probabilistic phrase reranking method and implemented it in the NSIR[12] Web QA system.

[11] http://www.google.com
[12] http://tangra.si.umich.edu/clair/NSIR/html/nsir.cgi

3 Task Generation

For the natural language task engine to perform user requests, it needs access to a task bank. The task bank is a database of tasks populated automatically via Web applications wrapping techniques, or inferred by the system observing user Web actions. Each task, i.e. booking a flight ticket, can execute a particular set of actions on a Web application. To execute a task, the dialogue system's task manager needs to fill the corresponding task required parameters. Therefore, each task must contain a list of parameters, their meaning and their type. For example, the parameter attributes for the "Origin" input field in the "book a flight" task will contain (1) the input value type as a city or an airport, and (2) its semantic meaning as departing location. In addition, each task will store natural language annotations, content-describing phrases and sentences of the task, te be used later by the task manager to map the user's request to the corresponding task to be performed.

To manage the tasks in the task bank we need to implement a Web user interface where the user can edit and add new applications to be wrapped. It will allow the user to inspect and update his stored profile. In addition, since our graphical user interface is a Web application the user will be able to manipulate it using natural language.

3.1 Site Wrapping

Given a site, the system will crawl the site, discover the underlying data and actions (Web forms), infer the semantics of the underlying data and actions, and generate a list of possible tasks.

We will extend our previous work of wrapping semi-structured websites [47] and enhance it to handle Web applications. Web applications are themselves semi-structured in a sense; they present information about objects in a similar structure, and they allow actions to be perform on those objects. The following steps delineate the site wrapping components:

1. Crawl the site: We will explore all pages we can get to from the main page through hyperlinks and forms. The form's fields descriptions are a good indicator of what type of argument each field expects. We will use those features, apply machine learning techniques to determine the possible values for those fields, and discover the sites

objects. Sometimes, it is impractical to discover all the site underlying pages since the site could contains information about millions of products such as flights, movies, restaurants, books, etc. Our goal in this step to get a representative sample of the site pages which we will use in later steps to discover the site's objects and actions.

2. Page classification: Using machine learning techniques, we will cluster the pages discovered in the previous step into similar types.

3. Wrapper induction: We will generate a template by aligning the pages of each type, identify their content, and semantically annotate the properties.

4. Generate the task graph: The task graph is like a site map. We will generate the task graph where pages are the nodes and actions are the edges. Each path in the graph will represent a task.

5. Tasks semantic annotation: After building the task graph we will automatically label the tasks with a meaningful descriptions and generate natural language annotations. We will experiment with machine learning techniques using as features - context and parameters of the actions consisting the tasks.

3.2 Task Learning

The task learning component has two responsibilities. First, given a set of user actions in the Web browser, the component will identify the variables from the actions, infer the semantics for those variables, and assign a list of possible natural language annotations for the new learned tasks. Second, it will use the contexts and the values of each actions to learn the user habits, and infer his preferences to improve the overall interaction with the system. We will experiment with various pattern recognition techniques to identify the variables in the actions, and annotate them with semantic information. We will also update the user profile with the learned preferences.

3.3 Related Work

3.3.1 Wrapper Generation

Wrapper generation is the creation of wrappers which contains scripts that extract and integrate data from data sources, mostly from Web data sources due to the large amount of data available on the World Wide Web. A Wrapper can be seen as a set of specialized programs that extracts data from a data source such as a Web site and manipulates the information into a suitably structured format to enable further processing. Usually after creating a wrapper for a Web site, the wrapper is post-processed and annotated with semantic properties. This process enables the extracted data to be further manipulated by other specialized programs. For example, an answer to "What are the 10 richest countries?" might require the question answering system to request the GDP field of all countries from a country site wrapper, sort the answers from richest to poorest, and present the 10 highest answers.

Various machine learning techniques were used towards developing automated wrapper generation systems. In addition, Wrapper generation systems specialize in particular types of data sources. There are systems that deal with semi-structured data, such a WIEN [29], STALKER [39], and Wrapster [47]. Wrappers are generated from a set of examples by computing a generalization that explains the observations. There are also more general wrapper generation systems, such as RAPIER [11, 12] and WHISK [43] that extract information from unstructured text data. Given a hand-tagged template, these systems learn pattern-matching rules to extract the tagged information in the template.

Recently, different approaches have been developed for semi-structured detail pages and list pages. A detail page is a page that focuses on a single item. A list page is a page that lists several items. The field of research that deals with list pages is also known as data record extraction. It has been shown by Liu et al. [35] that one input page is sufficient for this task. Wrapper generation systems for list pages first identify data record boundaries and then generate patterns to extract the data records their properties.

Other efforts were conducted towards interactive wrapper generation also known as visual wrapper generation. Such an approach provides specialized pattern specification languages to help the user construct data extraction programs. Interactive wrapper generation systems

hide their complexities under a graphical user interface. Systems that follow this approach include WICCAP [33], Wargo [42], Lixto [8], DEBye [31], Dapper[13], etc. Those systems were the beginning of a new research field called mashups described in the next section.

A mashup is a web application that integrates data from more than one source into a new integrated view of the data. The new view creates a new way to view data from the independent sources into non-existent view that fit the user needs. Most well known mashups are those using Web map services by adding location information to any data source that contains location such as restaurants and real-estate data, thus creating new non-existent services.

Recently, the creation and use of mashups have become very popular over the Web and the more editors and tools have emerged to help non-expert users create their own mashups and augment simple sites to rich and interactive services. Examples include Intel Mash maker [16], Google Mashup Editor [14], Potluck [22], etc.

Even though the field of mashup creation in not concerned with natural language interfaces, we are interested in mashup creation tools since they integrate information from multiple different data sources which can help our system perform complex user requests requiring fusion between multiple Web services. For example, Ennals et al. [16] showed how a casual user can augment a travel site with airplane legroom information and search for flights using this new non-existent attribute using a airplane legroom data source.

3.3.2 Learning by Demonstration

To perform user requests, either answering a question or retrieving information requires the computer systems to have knowledge. This knowledge a computer program has, is not infinite and in many cases the system cannot perform the user request. Task learning is the field of research where user teach the system interactively how to perform new custom tasks or modify old ones.

The problem has been addressed in previous work using various approaches to learn from specific examples provided by users as they perform a task, and include programming by

[13]http://www.dapper.net
[14]http://code.google.com/gme/

demonstration, learning apprentices, case-based reasoning, and feature-based induction. Programming by demonstration (PBD) [34, 32] also known as Programming by examples, is a technique that creates an executable program from example data and/or sequence of examples operations/actions given by the user. The PBD framework allows users with little programming skill to automate repetitive tasks without learning any complicated programming skill to automate repetitive task. This approach can be inefficient if examples are hard to formulate, for instance in complex domains. Huffman et al. [21] used informal language to tell the system what to change and had the system modify the procedure appropriately without the need to make low-level decisions about the implementation. Blythe [9] developed tailor a task learning system to modify task information through instruction using standard procedure language. Tailor improves Huffman's work and reasons about an abstract process description and it does not assume domain knowledge. Other approaches have used problem-solving methods, interdependency analysis and smart editors to help user work with procedure representation [10, 28]. However, those approaches requires users to make implementation-level decisions about changes to procedures, during which it is easy to lose track of changes or make mistakes.

In addition, task learning was used in adaptive Programming Environment (APE), a software assistant that watches what the user is doing, draws on machine learning to learn user habits, and afterward offers to complete repetitive tasks on his or her behalf. The goal of a APE framework was: (1) design an assistant with a minimal amount of user's intervention, (2) able to replay and automate complex repetitive tasks (3) give right suggestions at the right moments. This approach address a flaws in PBD where teaching the system disrupts the user workflow. Predictive interfaces [20] and learning interface agents [37] observe the user while he manipulates the environment. They try to learn from the correlations between situations the user has encountered and the corresponding commands he has performed, and to predict after each new command what the next one will be. They assist him by afterward predicting and suggesting some commands to perform automatically. Such assistants for the Web exist, but are very limited. WebWatcher [7], an assistant for the World Wide Web, suggests links of interest to the user.

Plow [5] uses 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 the collaborative learning, the system may do both parts. It may learn a new step in the task being learned, but because it already knows how to do the sub-task, it also performs that sub-task for the user. This type of collaborative execution while learning is critical in enabling of iterative steps without requiring the user to tediously demonstrate each loop through the iteration.

4 Discussion

To summarize, the work in this thesis includes the design and implementation of an end-to-end system to perform real life tasks on the Web using natural language. While there are still serious research and technical issues remaining to be overcome, dialogue-based user interfaces are showing promise. The proposed system architecture (Figure 3) consists of following components: dialogue system, site wrapping, task learning, and task bank. The dialogue system interacts with the user towards achieving the user underlying goal. After interpreting the user request it will access the task bank to find the set of actions corresponding to the user request. The task learning component observes the user actions at all times and infers new tasks to add to the task bank. It also learns user habits and preferences to provide more natural future interaction with the system. The site wrapping component feeds new tasks to the task bank by automatically crawling Web applications, discovering the underlying data and structure, and inferring the services that the applications provide.

Apart from evaluating each component with a held out data, we will evaluate the system as a whole. We will choose subjects that have never seen the system to interact with it to achieve a list of tasks and choose a new task of their liking to add to the task bank. We will provide the user with a survey to comment on the quality and intuitiveness of the system and whether the system successfully executed the tasks.

4.1 Scope and Assumptions

While attempting to solve this goal, we are very aware of the difficulty to develop an error-free system. Therefore, the system will always take into account user feedback and adapt. In addition, the user will be able to interrupt the system through the user interface and correct the system actions when necessary.

In some cases, to protect against bots, Web applications use various protection tools such as displaying distorted character sequence which the user needs to repeat in order to perform his requests. There is little we can do to handle these situations but in such a case the task manager will request from the user the missing information. In addition, our system need to be very careful dealing with user private information for security and user privacy concerns.

Concerning the system implementation environment, there are many Web browsers available; however, their rendering module differs from one to another. In addition, each browser provides different customization features. We choose to use Mozilla Firefox[15] to develop and test our user interface on since making the system compatible with all Web browsers require many little tweaks that are not the focus of this thesis.

Furhermore, even though it will be valuable to have the system suggest related tasks by observing the user actions and requests history, and notify the user about conflicts with previously executed requests, we will it leave for future work.

4.2 Data

There is an abundance of different applications on the Web such as Web mail, document editors, retail sites, and wikis. In this thesis, we will focus on retail sites. Web retail sites are online stores that that offer products or services. They vary in functionality, logic, and appearance and are used by millions of users every day. We will use a representative group from various available types of applications to evaluate our framework on. The framework we will develop should also apply to other types of Web application; however, Web applications such as Web mail have a simple and known set of actions for which it is not clear if a natural language interface is better than the existing structured interface.

We will choose a couple of applications from each of the following types to train and test our framework on:

- Internet travel companies, i.e. orbitz, travelocity
- Ticketing companies , i.e. fandango, ticketmaster
- Online reservation services, i.e. opentable,
- Dining in services, i.e. campus dining, diningin
- retail stores, i.e. newegg, amazon
- banking services, i.e. bankofamerica, citibank

[15] http://www.firefox.com

- renting services, i.e. netflix, blockbuster

4.3 Expected Contributions

The following is a list of proposed contributions:

1. Automatically discovering the underlying data and actions of Web application.
2. Formalizing a knowledge representation model of Web applications.
3. Constructing a general framework for performing Web tasks using natural language interface.
4. Learning hidden variables of Web tasks by observing user actions.
5. Determining system response type/format based to user query.
6. Demonstrating an interactive user interface to manage the wrapped sites, and their corresponding tasks that can be operated by casual users.

References

[1] Toward conversational human-computer interaction. *AI Mag.*, 22(4):27–37, 2001.

[2] Eugene Agichtein, Steve Lawrence, and Luis Gravano. Learning search engine specific query transformations for question answering. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 169–178, New York, NY, USA, 2001. ACM.

[3] Stephanie Seneff Alexander Gruenstein and Chao Wang. Scalable and portable web-based multimodal dialogue interaction with geographical databases. In *Proceedings of Interspeech 2006 ICSLP*, pages 453–456, Pittsburgh, 2006.

[4] James Allen, Donna Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent. An architecture for a generic dialogue shell. *Nat. Lang. Eng.*, 6(3-4):213–228, 2000.

[5] James Allen, Nathanael Chambers, George Ferguson, Lucian Galescu, Hyuckchul Jung, Mary Swift, and William Taysom. Demonstration of PLOW: A dialogue system for one-shot task learning. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, pages 1–2, Rochester, New York, USA, April 2007. Association for Computational Linguistics.

[6] James Allen, George Ferguson, and Amanda Stent. An architecture for more realistic conversational systems. In *IUI '01: Proceedings of the 6th international conference on Intelligent user interfaces*, pages 1–8, New York, NY, USA, 2001. ACM.

[7] Robert Armstrong, Dayne Freitag, Thorsten Joachims, and Tom Mitchell. Webwatcher: A learning apprentice for the world wide web. pages 6–12. AAAI Press, 1995.

[8] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual Web Information Extraction with Lixto. In *The VLDB Journal*, pages 119–128, 2001.

[9] Jim Blythe. Task learning by instruction in tailor. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 191–198, New York, NY, USA, 2005. ACM.

[10] Jim Blythe, Jihie Kim, Surya Ramachandran, and Yolanda Gil. An integrated environment for knowledge acquisition. In *IUI '01: Proceedings of the 6th international conference on Intelligent user interfaces*, pages 13–20, New York, NY, USA, 2001. ACM.

[11] Mary Elaine Califf. *Relational Learning Techniques for Natural Language Information Extraction*. PhD thesis, Department of Computer Sciences, University of Texas, Austin, TX, August 1998. Also appears as Artificial Intelligence Laboratory Technical Report AI 98-276 (see http://www.cs.utexas.edu/users/ai-lab).

[12] Mary Elaine Califf and Raymond J. Mooney. Bottom-up relational learning of pattern matching rules for information extraction. *Journal of Machine Learning Research*, 4:177–210, 2003.

[13] Jennifer Chu-carroll. Mimic: An adaptive mixed initiative spoken dialogue system for information queries. In *In Proceedings of the Sixth Conference on Applied Natural Language Processing*, pages 97–104. AAAI Press, 2000.

[14] Nils Dahlback and Arne Jonsson. Knowledge sources in spoken dialogue systems. In *In Proceedings of Eurospeech'99*, pages 1523–1526, 1999.

[15] Myroslava O. Dzikovska, James F. Allen, and Mary D. Swift. Integrating linguistic and domain knowledge for spoken dialogue systems in multiple domains. In *IJCAI*, Acapulco, Mexico, 2003.

[16] Rob Ennals, Eric Brewer, Minos Garofalakis, Michael Shadle, and Prashant Gandhi. Intel mash maker: join the web. *SIGMOD Rec.*, 36(4):27–33, 2007.

[17] Annika Flycht-eriksson. A survey of knowledge sources in dialogue systems. In *In: Proc. of IJCAI'99 Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, pages 41–48, 1999.

[18] Annika Flycht-eriksson and Arne Jonsson. A spoken dialogue system utilizing spatial information. In *In Proceedings of ICSLP'98*, 1998.

[19] Eric J. Glover, Gary W. Flake, Steve Lawrence, Andries Kruger, David M. Pennock, William P. Birmingham, and C. Lee Giles. Improving category specific web search by learning query modifications. In *SAINT '01: Proceedings of the 2001 Symposium on Applications and the Internet (SAINT 2001)*, page 23, Washington, DC, USA, 2001. IEEE Computer Society.

[20] Saul Greenberg, John J. Darragh, David Maulsby, and Ian H. Witten. Predictive interfaces: what will they think of next? pages 103–140, 1995.

[21] Scott B. Huffman and John E. Laird. Flexibly instructable agents. *Journal of Artificial Intelligence Research*, 3:271–324, 1995.

[22] David F. Huynh, Robert C. Miller, and David R. Karger. Potluck: Data mash-up tool for casual users. In *ISWC/ASWC*, pages 239–252, 2007.

[23] Boris Katz. Using English for Indexing and Retrieving. Technical Report AIM-1096, 1988.

[24] Boris Katz. Annotating the World Wide Web Using Natural Language. In *Proceedings of the 5th RIAO Conference on Computer Assisted Information Searching on the Internet (RIAO '97)*, Montreal, Canada, 1997.

[25] Boris Katz, Gary Borchardt, and Sue Felshin. Syntactic and semantic decomposition strategies for question answering from multiple resources. In *Proceedings of the AAAI 2005 Workshop on Inference for Textual Question Answering*, pages 35–41, 2005.

[26] Boris Katz, Sue Felshin, Deniz Yuret, Ali Ibrahim, Jimmy Lin, Gregory Marton, Alton Jerome McFarland, and Baris Temelkuran. Omnibase: Uniform Access to Heterogeneous Data for Question Answering. In *Proc. of the 7th Int. Workshop on Applications of Natural Language to Information Systems (NLDB '02)*, Stockholm, Sweden, June 2002.

[27] Boris Katz and Jimmy J. Lin. Start and Beyond. http://citeseer.ist.psu.edu/556306.html, 2002.

[28] Jihie Kim and Yolanda Gil. Deriving expectations to guide knowledge base creation. In *AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, pages 235–241, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[29] Nicholas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper Induction for Information Extraction. *In Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.

[30] Cody Kwok, Oren Etzioni, and Daniel S. Weld. Scaling question answering to the web. *ACM Trans. Inf. Syst.*, 19(3):242–262, 2001.

[31] Alberto H. F. Laender, Berthier Ribeiro-Neto, and Altigran S. da Silva. DEByE - Data extraction by example. *Data Knowledge Engineering*, 40(2):121–154, 2002.

[32] Tessa Lau, Lawrence Bergman, Vittorio Castelli, and Daniel Oblinger. Sheepdog: Learning procedures for technical support. In *In Proceedings of IUI 2004*, pages 109–116, 2004.

[33] Zhao Li and Wee Keong Ng. WICCAP: From Semi-structured Data to Structured Data. *Engineering of Computer-Based Systems*, 00:86, 2004.

[34] Henry Lieberman, editor. *Your wish is my command, Programming by example*. Morgan Kaufmann, 2001.

[35] Bing Liu, Robert Grossman, and Yanhong Zhai. Mining data records in Web pages. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 601–606, New York, NY, USA, 2003. ACM Press.

[36] Jayant Madhavan, David Ko, Lucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. Google's deep web crawl. *Proc. VLDB Endow.*, 1(2):1241–1252, 2008.

[37] Pattie Maes. Agents that reduce work and information overload. *Commun. ACM*, 37(7):30–40, 1994.

[38] David Morley and Karen Myers. The spark agent framework. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 714–721, Washington, DC, USA, 2004. IEEE Computer Society.

[39] Ion Muslea, Steve Minton, and Craig Knoblock. A hierarchical approach to wrapper induction. In *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, pages 190–197, New York, NY, USA, 1999. ACM Press.

[40] Dragomir Radev, Weiguo Fan, Hong Qi, Harris Wu, and Amardeep Grewal. Probabilistic question answering on the web. In *Journal of the American Society for Information Science and Technology*, pages 408–419, 2002.

[41] Dragomir R. Radev, Kelsey Libner, and Weiguo Fan. Getting answers to natural language questions on the web. *JASIST*, 53.

[42] Juan Raposo, Alberto Pan, Manuel Álvarez, Justo Hidalgo, and Ángel Viña. The Wargo System: Semi-Automatic Wrapper Generation in Presence of Complex Data Access Modes. In *DEXA '02: Proceedings of the 13th International Workshop on Database and Expert Systems Applications*, pages 313–320, Washington, DC, USA, 2002. IEEE Computer Society.

[43] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Journal of Machine Learning*, 34(1-3):233–272, 1999.

[44] Janienke Sturin, Els den Os, and Lou Boves. Dialogue management in the dutch arise train timetable information system.

[45] Ellen M. Voorhees and Dawn M. Tice. The trec-8 question answering track evaluation. In *In Text Retrieval Conference TREC-8*, pages 83–105, 1999.

[46] Wei Xu and Alexander I. Rudnicky. Task-based dialog management using an agenda. In *ANLP/NAACL 2000 Workshop on Conversational systems*, pages 42–47, Morristown, NJ, USA, 2000. Association for Computational Linguistics.

[47] Gabriel Zaccak. Wrapster: Semi-automatic wrapper generation for semi-structured websites. Master's thesis, MIT, 2007.

[48] Victor Zue, Stephanie Sene, James Glass, Joseph Polifroni, Christine Pao, Timothy J. Hazen, and Lee Hetherington. A telephone-based conversational interface for weather information. *IEEE Trans. on Speech and Audio Processing*, 8:85–96, 2000.