An Efficient Plan Execution System for Information Management Agents

Greg Barish, Dan DiPasquo, Craig A. Knoblock, Steven Minton
Integrated Media Systems Center,
Information Sciences Institute,
Department of Computer Science
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
{barish, dipasquo, knoblock, minton}@isi.edu

ABSTRACT

Recent work on information integration has yielded novel and efficient solutions for gathering data from the World Wide Web. However, there has been little attention given to the problem of providing information management capabilities that closely model how people interact with the web in productive ways - not only collecting information, but monitoring web sites for new or updated data, sending notifications based on the results, building reports, creating local repositories of information, and so on. These needs are unique to the dynamic nature of information in a networked environment. In this paper, we describe Theseus, an efficient plan execution system for information management agents. Through its plan language, Theseus supports a number of capabilities which enable practical information management, including repeated and periodic query execution, conditional plan declarations, query result aggregation, and flexible communication of results. The Theseus executor system focuses on efficiency, with support for data pipelining, and dataflow-based, event driven parallel execution. With Theseus, users can automate the complex but practical ways in which they interact with the web, for both information gathering and management.

1. INTRODUCTION

Gathering information from the World Wide Web is a research problem that has been receiving substantial attention in recent years. There now exist a number of promising systems [9, 13, 14] and approaches towards automating this process, including work on data extraction [15, 17], query planning [1, 16], data materialization [2], and methods for handling data inconsistency [3].

While gathering data is unquestionably an important task, there are also challenges related to the effective management and use of this data. We believe that information gathering is a piece of a larger puzzle called *information management*, a problem which involves topics such as conditional plan execution, continuous querying, progressive query result aggregation, and the linking of other actions to the results of queries. The problem of web information management thus encompasses issues which are at the heart of how users query the web today to retrieve meaningful information and the way such data is put to practical use.

For example, consider how people use the web today for locating houses for sale which meet a particular set of criteria (*e.g.*, price and location). This process means more than simply executing a particular query once and returning a long list of data. More likely, searching for a house means executing that same query periodically, say on a daily basis, over the course of a few weeks or months. Perhaps it even entails changing the query over time if only a few houses are found. Furthermore, the search process usually involves gathering only new or updated listings (meeting the specified criteria) upon every query execution. Users are rarely interested in being reminded of houses about which they have already been notified. Furthermore, with the explosive growth in mobile networking, there are many users who would prefer to have their query results distributed through various messaging means (*i.e.*, pager, cellular phone, fax) and reported in a variety of formats (*i.e.*, XML, HTML, WML, text, voice). Finally, many users want to do more than simply be notified of results. It is often desirable to have newly gathered information trigger a variety of other actions. For example, if a very specific house search yields a result, a user may want to immediately send an automated e-mail to the corresponding real-estate agent, declaring interest in the house and suggesting a time at which to meet (based on the users' personal schedule, also kept online).

The information management paradigm is obviously not limited to those looking for a new house. There are numerous other instances where such automation is not only useful, but perhaps essential: newswire tracking, online auction participation, and stock/portfolio management, to name a few. Users want more than to simply retrieve data. They want to be able to monitor web sites, to receive query result updates periodically when useful information in retrieved, and to link other actions to the results of these continuous queries. The dynamic nature of the web invites this style of information management.

In this paper, we describe Theseus, an efficient plan execution system for agents which addresses many such challenges. Based on a parallel dataflow-based architecture, the Theseus executor is designed for high performance and information throughput. Its plan language supports the expression of loops, conditionals, and synchronization primitives. Through its language and execution system, Theseus enables agents to perform useful information management tasks, such as periodic execution, query result aggregation, and flexible result communication, as a way of addressing practical ways in which users interact with the web.

1.1. Challenges

Perhaps the most basic challenge of web information management is to design an infrastructure which enables many of the features described above. Such a system requires plan operators which lie beyond those for simply specifying a query. Users need to have operators which can communicate information to them via a variety of devices. There is also a requirement to store past query results, so that future queries can distinguish new or updated data from that which has already been seen. Enabling information management also involves empowering users so that they can specify plans which execute periodically and conditionally.

Beyond the need of enabling basic information management, there is the challenge of making the execution system efficient. Data integration already faces substantial performance challenges, primarily due to the nature of integrating remote data sources. For example, there are the costs of network latencies for accessing external web sites, the costs associated with navigating through multiple web pages to gather a set of logical data, and the cost/risks of availability, performance, and reliability aspects of these external sites. Since it is not feasible to control any of these external variables, an interesting challenge is to design an execution system which is efficient despite these constraints.

Finally, a related challenge has to do with providing a means for simplifying the declaration and execution of complex information management tasks. For example, one complex task has to do with collecting a logical set of data from multiple web pages, accessing each set of tuples in various ways (forms, “next page” links, etc.). Querying a set of logical data (such as houses for sale) from a web site might involve a number of steps: (a) filling out the initial web-based query form, specifying price and location, to return the first page of results, (b) extracting the data from this initial results page, (c) detecting the potential presence of “NEXT PAGE” links, (d) following those links, (e) extracting subsequent data, and finally (f) accumulating all of that data so that a single result can be returned to the user. With many existing data integration mechanisms, this type of result accumulation interleaved with navigation is not a simple task, the complexity of which is described further in [9]. Theseus aims to make this type of complex information gathering and management easy to specify, as well as efficient to execute.

1.2. Theseus

Theseus has evolved from research related to the Ariadne [14] project at USC. Ariadne is an *information mediator* that integrates multiple heterogeneous data sources, including local databases, web sources, and knowledge bases so that the combined data can be accessed from a single, logical model. To extract data from the web, Ariadne uses data source *wrappers* to query web sites as if they were SQL databases.

Ariadne provides a framework from which to build information integration applications. We believe Theseus is a logical next step: it builds on the integration Ariadne enables, allowing users to do something useful with information that is gathered. By designing a system specifically for the execution of information management plans, we can better address complex integration and efficiency challenges.

2. MOTIVATING EXAMPLE

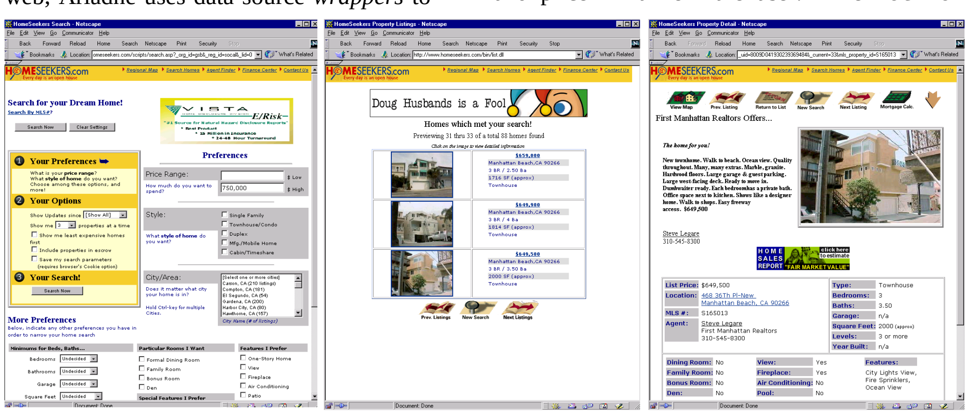
To describe our motivations for designing Theseus, we now consider an example application. We will focus on monitoring the HomeSeekers web site (*http://www.homeseekers.com*), which allows users to locate available houses for sale. In our example, we will monitor the site for the ongoing availability of houses which match our location and price constraints.

The initial HomeSeekers page consists of a form-based query interface, shown in Figure 2.1(a). Submitting this form returns a page containing up to three houses, as in Figure 2.1(b). At the bottom of this page, there may also be a "Next Listings" URL that leads to another page of three houses, and so on - until all the houses that match this query are shown. A further complication arises because in order to get detailed information from each of these house listings, we must follow an additional URL for each house. This detail page is shown in Figure 2.1(c).

By simply examining the layout of the HomeSeekers site, we can identify the need for conditionals and looping when performing data extraction. For example, notice that Figure 2.1(b), the listings page, shows that users can only view three houses at a time before needing to click on the “NEXT” link. At some point, we will have reached the last page of results and there will be no such link. Thus, support for conditional execution is necessary. Furthermore, extracting the results means collecting the URL and whatever information is present on the listings page. Later, or in an interleaved fashion, the details of each house will need to be extracted. This requires either iterating through each set of three houses or eventually looping through the entire accumulation.

Another observation of the HomeSeekers example is that we will be making multiple data retrievals. For example, we will be collecting house listings as well as detailed information about each house. Since network access tends to be a major bottleneck in data integration systems, it would be preferable to parallelize as much of this data gathering as possible.

Figure 2.2 shows the Theseus plan for monitoring the HomeSeekers site. Essentially, the plan notifies us when it becomes aware of new houses which meet our criteria. The plan is invoked with a location and price limit from the user. The `Retrieve` operator (which



(a) **(b)** **(c)**

Figure 2.1: Querying HomeSeekers from the Web

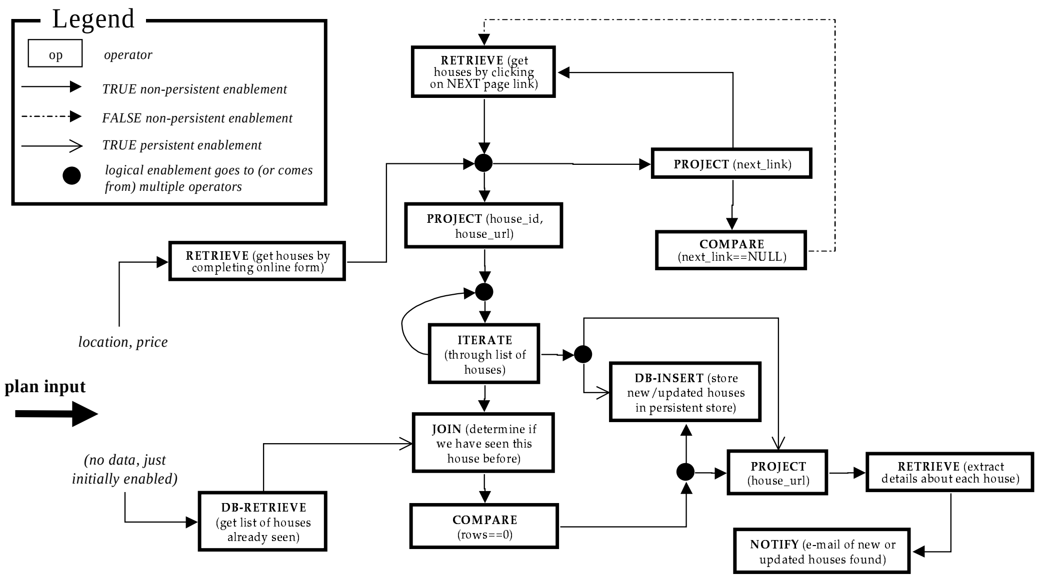


Figure 2.2: The Theseus HomeSeekers plan

retrieves data from web sites via a wrapper) takes these initial constraints, posts them to the initial HomeSeekers query form, and extracts a relation with *house_id, house_url*, and *next_link* attributes. `Retrieve` then passes this relation to two different loops. One loop, shown at the top of Figure 2.2 has the purpose of following the NEXT links at the bottom of each listings page, and passing these links to a second loop, shown near the bottom of Figure 2.2. This loop iterates through the listings sent from the first loop, comparing each with those stored in a local database, and then potentially extracts more detailed information for each new house. The plan shows the concurrent queuing of listings with the investigation of house details, a plan which assumes asynchronous, parallel execution and data pipelining for efficient performance.

3. SYSTEM DESIGN

The two key components of the Theseus system are its plan language and execution system. The former allows complex information management plans to be easily expressed while the latter supports a multi-threaded, event-driven architecture, supporting parallel execution and asynchronous data queuing for improved efficiency.

3.1. Plan Language

The Theseus plan language is composed of a set of *operators*, each of which is associated with a set of input, output, and error *enablements*. Operator execution is triggered by one or more enablements, some of which may be carrying data. An operator only executes after receiving all of its required enablements. Upon completing execution, an operator returns either TRUE, FALSE, or an error condition. Depending on the result state, another set of enablements may be activated.

A Theseus *plan* consists of a set of operators and their enablements. Plans can be viewed as parallel execution graphs, where operators act as nodes and enablements are edges. Each node can be modeled as a quintuple *node* = *<op, persistent-in, non-persistent-in, true-out, false-out, err-out>*, where *op* is the name of the operator, *persistent-in* is the set of persistent enablements required for execution, *non-persistent-in* is the set of non-persistent enablements required, and *true-out, false-out, err-out* are the sets of enablements which are activated based on the TRUE/FALSE/error execution result of that operator. The order in which enablements are activated describes both the execution path and the dataflow path.

A plan is initialized with a set of initial input enablements which trigger the execution of one or more operators in the plan. When these initial operators complete execution, they may generate new enablements which in turn may trigger the execution of other operators. This process continues until all previously enabled operators have completed execution.

3.1.1. Enablements

Enablements can simply be thought of as "signals" which are generated during plan execution. They are activated either at the start of execution or when various operators complete their execution. Each TRUE/FALSE/error result state of an operator can be associated with a set of enablements.

For example, consider the behavior of the `Project` operator. shown in the lower, right-hand side of Figure 2.2. If the execution of `Project` results in TRUE, the operator sends enablements which are consumed by the `Retrieve` operator that extracts house details. `Project` does not generate any enablements when it returns FALSE or encounters any error states. In contrast, the `Compare` operator (upper, right-hand side of Figure 2.2) only generates an enablement when it returns FALSE. The point here is to show that the execution status of an operator can be associated with different sets of output enablements. This is one way in which Theseus supports conditional execution.

Enablements may or may not carry data. For example, the `Project` operator (middle of Figure 2.2) associates a relation with the enablement sent to the `Iterate` operator, whereas all of the `Compare` operators in the plan simply provide enablements (no data). An operator producing a data-carrying enablement and an operator which consumes that same enablement (and the data it carries) describes how data is transmitted through the system from one operator to another.

3.1.1.1. Enablement Persistence

Enablements can either be *persistent* or *non-persistent*. The former, once generated, remain active for the duration of plan execution. Furthermore, if they carry data, that data is continually available for the operator to use upon every invocation. If, as execution progresses, another operator generates the same persistent enablement, but with new data, that new data is henceforth used for all future invocations of that operator.

Non-persistent enablements, on the other hand, are simply those which are only alive until they are consumed. Thus, once an operator executes based on enablements `E1` and `E2`, those enablements no longer exist in the system and must be regenerated in order for the operator to execute again. However, unlike persistent enablements, non-persistent enablements (signals and data) can be queued. Since Theseus is a parallel execution environment, it is often possible to have data queuing at multiple operators which accept non-persistent enablements. This creates the effect of asynchronous data pipelining during plan execution.

3.1.1.2. Uses of Enablements

Enablements have four functions in the Theseus plan language: to provide a mechanism for control flow, to provide synchronization during execution, to provide a way to pass data between operators, and to provide a way to label data in the system. In terms of control flow, enablements are the basis for how a plan is executed. Although many operators have an implicit number of enablements necessary for execution (corresponding to how many input arguments those operators require), the Theseus plan language allows one to associate additional enablements as pre-requisites for the operator to execute. These enablements then are simply being used as extra (non-data carrying) criteria necessary to trigger operator execution.

Enablements can also be used to implement looping behavior during plan execution. For example, notice that the `Iterate` operator in the HomeSeekers plan *enables itself* upon returning TRUE. To understand what is happening here, and how this is not an infinite loop, we need to briefly describe the semantics of `Iterate`. Upon execution, `Iterate` attempts to remove the first tuple from a relation. If this was possible (*i.e.*, the relation contained at least one tuple), `Iterate` returns TRUE and produces at least two new enablements: one containing the tuple extracted and the other containing the input relation minus that tuple. In our example plan, the latter object is sent back to `Iterate`. Obviously, at some point, all tuples in the relation will have been removed, at which point `Iterate` will return FALSE.

In a similar fashion, enablements can be used to provide synchronization during execution. For example, if the plan author wants operator `OP3` to execute after both `OP1` and `OP2` complete their execution, he simply adds an enablement `E1` to the set of output enablements for `OP1`, an enablement `E2` to the set of output enablements for `OP2`, and `E1` and `E2` to the set of input enablements for `OP3`. Thus, `OP3` will not execute until both `OP1` and `OP2` have completed their execution.

Finally, enablements are used as the basis for passing data between operators and as a way to uniquely identify that data. This is useful in order to identify dataflow through the system. Data labeling through enablements provides a way to organize inputs from multiple operators into a new operator for execution.

3.1.2. Operators

Operators are the mechanism for specifying functionality in Theseus plans. Typically, they take some input data, perform a useful operation on that data, and then return a new set of data or perform some external action (such as send e-mail containing that new data to a particular person).

For example, Theseus supports a `Project` operator, which corresponds to the relational algebra operator of the same name. As input, `Project` takes two arguments: a relation and a set of attributes to be projected. As output, `Project` provides a relation which contains only the projected attributes. `Project` thus requires at least two distinct enablements in order to execute: the relation to be projected and the projection criteria. To avoid cluttering the HomeSeekers plan in Figure 2.2, we omitted this second enablement.

Tables 3.1 summarizes most of the Theseus operators. Notice that they fall into three groups: *data manipulation, control,* and *communication*. The first type focus on more traditional, relational-style operators for processing data. The second group provides support for conditional execution, loops, and synchronization. The third type enables external input and output of data., including operators for extracting data, interacting with external databases, and notifying users via e-mail or pager. This last group of operators essentially provides the mechanisms for specifying input and output to Theseus plans.

Data Manipulation	**Control**	**Communication**
Select	Iterate	Notify
Project	Compare	Db-Retrieve
Join	Fork	Db-Store
SetDifference	Wait	Db-Insert
Union	Queue	Db-Delete
Aggregate	Null	
Sort		
Concat		

Table 3.1: Theseus operator classifications

3.2. The Execution System

Two key strengths of the Theseus execution environment are its support for parallelism and data pipelining. Each operator is implemented as a thread that begins execution when its enablement criteria are met. Since each operator is independent, it is often the case that multiple operators will execute in parallel. When execution is complete, operators enable other operators through output enablements. Thus, the execution system is highly parallel and event driven.

We can best illustrate the pipelining and parallelism available in Theseus by examining the portion of the HomeSeekers plan in which we navigate the through the listing pages via next buttons and retrieve (*house_id, house_url, next_link*) tuples from the page. Figure 3.1 shows an approximation of the operator scheduling for this portion of the plan. The figure shows how the data retrieved from the house listings page is sent to two parts of the plan which are executing in parallel: one part which explores house details

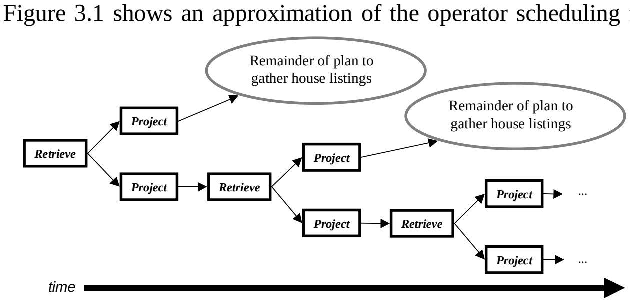


Figure 3.1: Approximation of operator scheduling

further (via the *house_url*) and one part which follows the *next_link* URL. The key advantage shown is that houses can be explored in detail as to whether they meet selection criteria while another part of the plan asynchronously extracts more house references and queues them for the same investigation.

This example demonstrates how the Theseus execution system compliments its plan language. While the latter allows plan writers to focus on the simple declaration of complex integration and management tasks in terms of data flow, the former permits highly concurrent execution and asynchronous data queuing.

4. RELATED WORK

There exist a few notable general plan execution systems, including PRS [10] and RAP [6]. These systems focus on real-time plan execution and interleaved planning and execution. They differ from Theseus in that plan execution is sometimes reactive, as in PRS, whereas Theseus is not concerned with the runtime modification of plans. Also, Theseus focuses purely on problems related to information management while the others are more commonly associated with robot-style plan execution.

An interesting vision for a plan execution system, closer to the design of Theseus, is that of [18], which explores information gathering using a "sensing" approach. The system described supports repeated execution of information gathering plans and, like Theseus, is interested in web site monitoring. Theseus differs from this system in that it defines an efficient architecture for executing these types of plans. Also, Theseus is concerned with other information management challenges, such as external communication with users and query result aggregation.

There has also been recent work describing approaches to the challenge of efficient information gathering from the Web [7]. Friedman and Weld describe a parallel execution system which optimizes sub-optimal plans for low cost execution. A related project, and one which bears similarity to Theseus in terms of its quest for execution efficiency, is the Tukwila system [12]. Tukwila bridges aspects of planning and database research in search of an execution system for data integration.

Theseus is also related to work in parallel databases [4], since it describes an architecture for operator execution in a parallel environment and part of its plan language is devoted to data manipulation operators, the same type that are found in such systems as GAMMA [5] and Volcano [11]. However, parallel database systems typically operate on local data sources and focus on optimized query processing in a parallel environment. In contrast, Theseus focuses on efficiently integrating multiple remote information sources and supporting mechanisms for practical, automatic, information management.

5. DISCUSSION

In this paper, we have presented the Theseus information management system. We have demonstrated that Theseus is a useful tool for building efficient agents which can gather information from the web and put that data to practical use. Because our planning language allows complex plans for managing information to be easily expressed, users can build powerful agents. Furthermore, since the execution system described is based on a dataflow paradigm, and supports data pipelining and a high degree of parallelism, these agents can obtain a high level of performance. While our system is currently very useful, we are working towards improving its user interface and optimization capabilities, for improved performance and scalability.

6. ACKNOWLEDGEMENTS

This work was supported in part by the United States Air Force under contract number F49620-98-1-0046, by the Rome Laboratory of the Air Force Systems Command and the Defense Advanced Research Projects Agency (DARPA) under contract number F30602-98-2-0109, and by the Integrated Media Systems Center (an NSF Engineering Research Center). The views and conclusions contained in this article are the authors' and should not be interpreted as representing the official opinion or policy of any of the above organizations or any person connected with them.

7. REFERENCES

[1] Ambite, J.L. and Knoblock, C.A. 1997. Planning by Rewriting: Efficiently Generating High-Quality Plans. Proceedings of the Fourteenth National Conference on Artificial Intelligence.

[2] Ashish, N.; Knoblock, C.A.; and Shahabi, C. 1999. Selective materializing data in mediators by analyzing user queries. Submitted, *Fourth IFCIS Conference on Cooperative Information Systems*.

[3] Cohen, W. W. 1998. Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity. *SIGMOD Conference 1998: 201-212*

[4] DeWitt D.J. and Gray, J. 1992. Parallel Database Systems: The Future of High Performance Database Systems. *Comm of the ACM 35(6).*

[5] DeWitt, D.J.; Ghandeharizadeh, S.; Schneider, D.A.; Bricker, A.; Hsiao, H.; and Rasmussen, R. 1990. The Gamma Database Machine Project. *IEEE Transactions on Knowledge and Data Engineering 2(1).*

[6] Firby, R.J. 1994. Task Networks for Controlling Continuous Processes. *Proceedings of the 2nd Intl Conference on AI Planning Systems*.

[7] Friedman, M. and Weld, D.S. Efficiently Executing Information-Gathering Plans, *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, Nagoya, Japan, August 1997.

[8] Friedman, M.; Levy, A.; and Millstein, T. 1999. Navigational Plans for Data Integration, *Proceedings of 16th Natl Conf on Artificial Intelligence*.

[9] Genesereth, M.R.; Keller, A.M.; and Duschka, O.M. 1997. Infomaster: An information integration system. *Proceedings of ACM SIGMOD-97.*

[10] Georgeff, M.P. and Lansky, A.L. 1987. Reactive reasoning and planning. *AAAI Proceedings 1987*.

[11] Graefe, G. 1994. Volcano - An Extensible and Parallel Query Evaluation System. *IEEE Transactions on Knowledge and Data Engineering 6(1).*

[12] Ives, Z; Florescu , D.; Friedman , M.; Levy, A.; Weld , D. 1999. An Adaptive Query Execution Engine for Data Integration. *Proc of ACM SIGMOD-99*.

[13] Levy, A.Y.; Rajaraman, A; Ordille, J.J. 1996. Querying Heterogeneous Information Sources Using Source Descriptions. *Proceedings of the 22nd VLDB Conference*.

[14] Knoblock, C.A.; Minton, S; Ambite, J.L.; Ashish, N.; Modi, J.; Muslea, I.; Philpot, A. and Tejada, S. 1998. Modeling Web Sources for Information Integration. *Proceedings of the 15th Natl Conf on Artificial Intelligence*.

[15] Kushmerick, N. 1997. *Wrapper Induction for Information Extraction*. PhD Thesis, Computer Science Dept. University of Washington.

[16] Kwok, C.T and Weld, D.S. 1996. Planning to gather information. In *Proceedings of AAAI-96.*

[17] Muslea, I.; Minton, S.; and Knoblock, C.A. 1998. STALKER: Learning Extraction Rules for Semistructured, Web-based Information Sources. *AAAI-98 Workshop on AI & Information Integration.*

[18] Williamson, M.; Sycara, K., and Williamson, M.. 1996. Unified Information and Control Flow in Hierarchical Task Networks. *Notes of the AAAI-96 Workshop, "Theories of Action, Planning, and Control."*