A Dataflow Approach to Agent-based Information Management

Greg Barish, Daniel DiPasquo, Craig A. Knoblock, and Steven Minton
Information Sciences Institute, Integrated Media Systems Center, and Department of Computer Science
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292

ABSTRACT

Recent research has made it possible to build information agents that retrieve and integrate information from the World Wide Web. Although there now exist solutions for modeling Web sources, query planning, and information extraction, less attention has been given to the problem of optimizing agent execution. In this paper, we describe Theseus, an efficient plan execution system for information agents. Through its pipelined, dataflow-style architecture, Theseus offers a high degree of parallelism and asynchronous information routing during execution. Theseus differs from prior work in reactive planning systems and parallel databases because it gathers information from the Web, a domain where information retrieval is a problem that is network-bound and is often based on interleaved data gathering and navigation. The Theseus plan language and architecture directly address these issues, resulting in an efficient execution system.

Keywords: plan execution, dataflow, data integration

1. INTRODUCTION

Gathering information from the World Wide Web is a research problem that has received substantial attention in recent years. There now exist a number of systems [7, 10, 13] and approaches towards automating this process, including work on data extraction [11, 14], query planning [1, 12], data materialization [2], and methods for handling data inconsistency [4]. Today, it is possible to construct useful agents built on these technologies to perform automatic and intelligent data integration [3].

Although these individual technologies may each be efficient, overall end-to-end agent execution performance is often sub-optimal. This is primarily because Web-based data integration is a process that is network-bound and masks the efficiencies of individual technologies (such as data extraction). Complicating matters is the fact that building useful agents often requires larger, more complex plans. For example, consider how people commonly use the Web to locate houses for sale that meet a particular set of criteria (*e.g.*, price and location). This process means more than simply executing a particular query once and then returning a long list of data. Typically, searching for a house means executing the same or similar queries periodically, perhaps on a daily basis, over the course of a few weeks or months. Furthermore, a "useful" search process means gathering only new or updated listings (meeting the specified criteria) for each query execution. Users are rarely interested in being reminded of houses they were previously notified about. Also, with the explosive growth in mobile networking, there are many users who would prefer to have their query results distributed through different messaging means (*i.e.*, e-mail, cellular phone, fax) and reported using a variety of formats (*i.e.*, XML, HTML, text, voice). Finally, in addition to message notification, it is often desirable to have newly gathered information trigger a variety of other actions. For example, if a search for a house yields a result, a user may want to immediately send an automated e-mail to the corresponding real-estate agent suggesting a meeting time (based on the user's personal schedule, also kept online).

Thus, while gathering data is unquestionably an important task, there are also challenges related to useful processing of this data. We believe that information gathering is a piece of a larger puzzle called *information management*, a problem that involves conditional plan execution, continuous querying, query result accumulation, local persistent storage, and the linking of other actions to the results of queries. This problem encompasses issues that 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. Searching for a new house is merely one type of application. 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. In these scenarios, users want more than to just query and retrieve data once - they want to be able to monitor Web sites. The dynamic nature of the Internet invites this approach.

However, a means for building high-performance agents for this type of information management remains a relatively open issue.

1.1. Contributions of This Paper

The main contribution of this paper is to describe the benefits of combining features found in both parallel databases and general plan execution systems, marrying the efficiency found in former with the generality and flexibility found in the latter, to improve the performance of dynamic information management. Parallel database research [5, 8, 16] has shown that it is possible to build highly efficient query execution systems for local databases. Existing plan execution systems [6, 15] have proven to be more generally applicable to a wide-range of planning problems and often provide more flexibility in terms of plan control flow (i.e., support for loops and conditionals).

We have implemented the combination of these features in Theseus, an efficient plan execution system for information agents. At the heart of Theseus is a parallel, dataflow-style executor. Furthermore, its plan language supports 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 means for addressing practical ways in which users interact with the Web. Most importantly, through properties of its architecture, Theseus reduces the overall effect of network latencies on data integration, providing increased parallelism and asynchrony during execution, so that the overall end-to-end agent execution process is substantially faster.

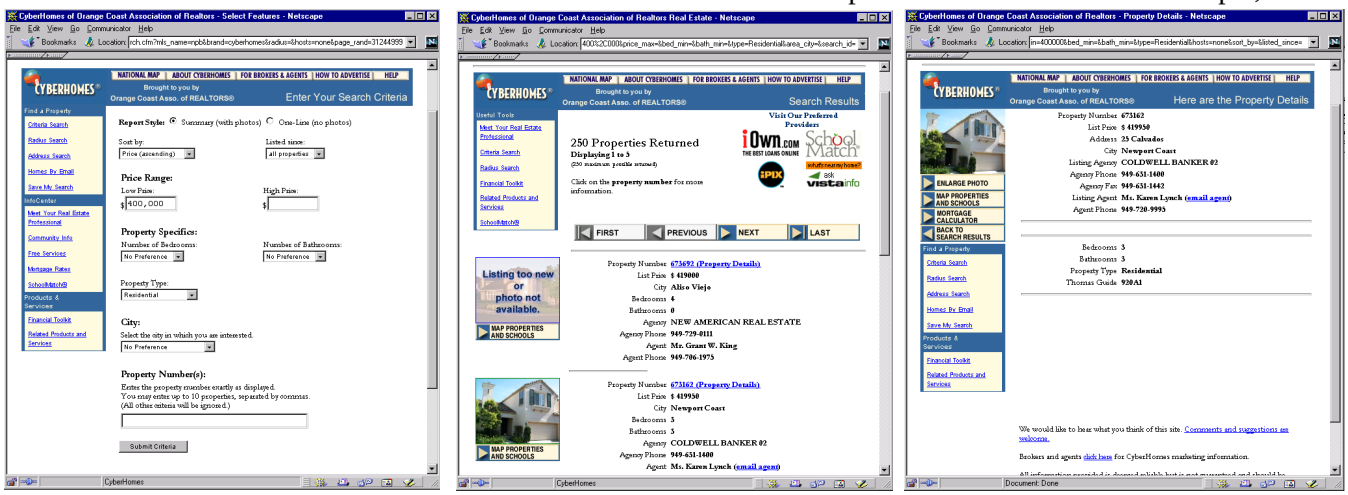
Theseus has evolved from research related to the Ariadne information mediator project [10]. Ariadne facilitates the integration of multiple heterogeneous data sources, including local databases, web sources, and knowledge bases, so that the combined data can be accessed from a single, logical model. To extract data from the Web, Ariadne uses data source *wrappers* to query web sites as if they were SQL-capable 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.

2. MOTIVATING EXAMPLE

To understand the motivations behind the design of Theseus more clearly, it is useful to consider an example of a realistic information integration plan. The example presented here involves CyberHomes, a web site that allows users to locate available houses for sale throughout the United States. The goal of this example is to monitor the CyberHomes site for the ongoing availability of houses that match a particular set of location and price constraints.

The initial CyberHomes web page consists of a form-based query interface, shown in Figure 1a. Submitting this form returns a page containing up to five houses, as in Figure 1b. At the bottom of this page, there may also be a "*Next Listings*" URL that leads to another page of five houses, and so on - until all of the houses that match this query are shown. Notice that a further complication arises because, in order to retrieve detailed information from these listings, a user must follow an additional URL for each house. The detail page is shown in Figure 1c.

By simply examining the layout of the CyberHomes site, it becomes fairly obvious that any automatic monitoring would require a plan that included loops and conditionals. For example, notice



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

Figure 1: The query page (a), results page (b), and detail page (c) from CyberHomes

in Figure 1b, the listings page, that users can only view five houses at a time before needing to click on the *Next Listings* link. At some point, users 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 list of URLs on each 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 five houses or eventually looping through the entire accumulation.

Another observation that can be made about the CyberHomes example is that multiple data retrievals are necessary. For example, the information to be gathered includes house listings as well as details about each house. The cost of retrieving this information is high: separate retrievals are required for each group of listings as well as for each page of details about a particular house. Since network access for remote data retrievals plays such a large role in the example, it would be preferable to parallelize as much of the remote retrievals as possible. For example, once the listings of the houses are retrieved, it would be optimal to obtain details about houses which meet the criteria specified – but to gather that data in parallel.

A related optimization involves the queuing of data retrieved. As mentioned earlier, each page of house listings may include a *Next Listings* link. To "monitor" the CyberHomes site, it is desirable to have the agent analyze each set of listings to identify houses that are both (a) new and (b) meet the search criteria specified. This analysis process may not take long, but it will not be instantaneous either. In the meantime, it may be possible to follow the *Next Listings* link and gather the next set of houses to be analyzed. In short, it would be optimal to queue the sets of listings for analysis, without having to wait for the completion of analysis on the previous listings. Specifically, a higher degree of asynchrony through queuing is desirable.

Figure 2 shows a Theseus plan for monitoring the CyberHomes site. Essentially, an agent executing this plan can notify a user when it becomes aware of new houses which meet a set of specified criteria. The plan is invoked with a location and price limit from the user. The Retrieve operator (which gathers data from web sites via a wrapper) takes these initial constraints, posts them to the initial CyberHomes 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 has the purpose of following the *Next Listings* links at the bottom of each listings page, and passing these links to a second loop, shown near the bottom of Figure 2. This next loop iterates through the listings sent from the first loop, comparing each with those stored in a local database, to determine which houses are new. Finally, the plan specifies the extraction of more detailed information for each new house.

3. EXECUTION SYSTEM

The design of the Theseus execution system is centered around efficiency. Where possible, opportunities for asynchrony and parallelism have been exploited, both properties significantly improving overall agent performance. These features are realized through the Theseus plan language and its dataflow architecture.

3.1 Plan Language

As shown in Figure 3, plans in Theseus are dataflow graphs, where graph nodes are *operators* and edges

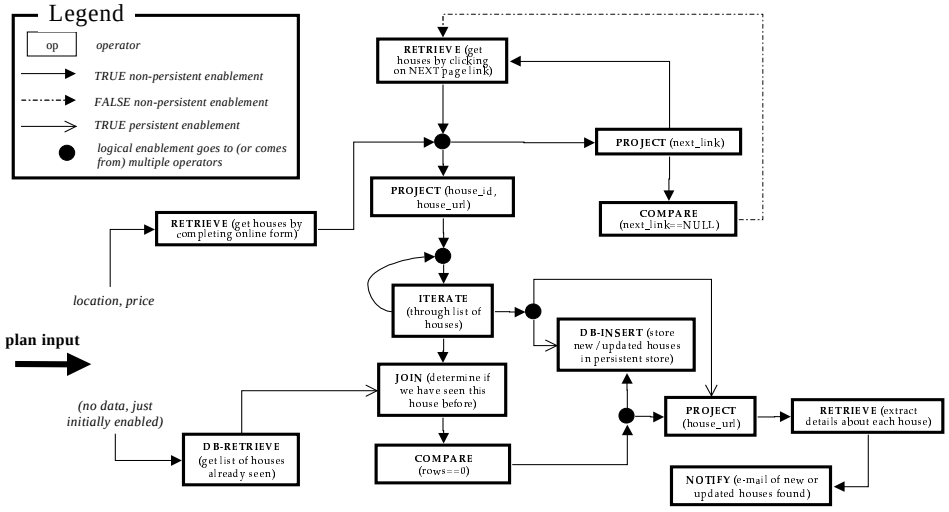


Figure 2: The Theseus CyberHomes plan

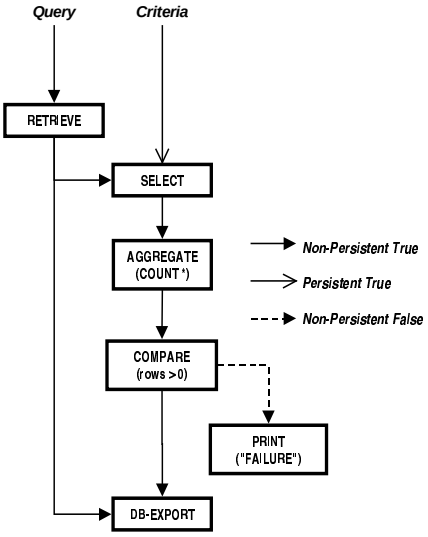


Figure 3: Operators and enablements

are *enablements*. Operators in Theseus act just as they do in many other planning systems: they perform some action when activated. Enablements are similar to pre-conditions and post-conditions in other planning systems. Just as the fulfillment of pre-conditions triggers an operator to execute, sets of enablements in Theseus activate its operators to execute. However, enablements perform this activation at run-time, whereas pre/post-conditions in other planning systems affect operators at the time of planning. Furthermore, the assertion of post-conditions and fulfillment of pre-conditions in most planning systems is a synchronous process. In contrast, enablements in Theseus can be communicated asynchronously because operators have input queues.

For example, Figure 3 shows a Retrieve operator that provides an enablement for the Select operator (performing the same function as the relational algebra operator of the same name). The purpose of combining these operators is to filter out data that has been retrieved from the Web based on some criteria. So, in some sense, the enablement that Retrieve provides Select is a means for asserting a post-condition indicating "Has-Data," as well as a means for routing that data. Notice also that Select is only *partially enabled* by Retrieve. There is another enablement which is associated with the data about the filtering criteria itself.

Plans in Theseus are associated with a set of initial enablements, just as plans in other systems specify a set of initial conditions. When a Theseus plan first starts executing, these initial enablements activate one or more operators in the plan, which execute in parallel. After those operators perform their action, they can each produce a set of resulting enablements. Those enablements, in turn, may activate other operators, which can generate new enablements, and so on. Processing continues until no more operators are enabled.

3.1.1 Operators

An operator can be thought of simply as a processing engine. Usually, operators take some input data, perform some computation, and produce a set of output data. For example, the Theseus Select operator takes a set of selection criteria and a relation on which to apply that criteria, and then produces a resulting relation which obeys that criteria. Since Theseus is an execution system for information agents, the most common type of data routed is a relation.

As a result of execution, operators return either true, false, or an error. This differs from operators in parallel database systems [7, 13, 24], which usually return true or (it is assumed) generate some form of exception. However, in Theseus, the ability for an operator to return a boolean result allows for an important property: *conditional execution*. The result of an operator can thus determine which output enablements are produced; in effect, which path(s) remaining plan execution follows.

For example, consider the behavior of the Compare operator in Figure 3. The prior operator, Aggregate, determines how many tuples have been selected out of the retrieved data. If `Compare` finds that there is at least one tuple, it returns true, otherwise it returns false. It is this result that will determine which enablement is asserted, thus whether DB-Export or Print is executed. This is a simple example of how operators and enablements can be combined to provide conditional execution.

Table 1 summarizes most of the Theseus operators. Notice that they fall into three groups: *data manipulation*, *control*, and *communication*. The first

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

Table 1: Theseus operator classifications

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.

3.1.2 Enablements

As described earlier, enablements can simply be thought of as pre/post-conditions which are required/established during plan execution. They may or may not carry data. For example, the Aggregate operator in Figure 3 asserts an enablement that routes a scalar value (the number of tuples) to Compare. In contrast, Compare does not route any data when it generates its true or false enablement.

Enablements can either be *persistent* or *non-persistent*. These two types permit more flexible and useful plan declarations as well as making plan execution more efficient. Persistent enablements improve the declarative power of Theseus plans, giving the plan author a mechanism for specifying a constant or static data, such as a query string or selection criteria.

In terms of efficiency, persistent enablements reduce the amount of signaling and data routing required during execution. Consider Figure 3, where the Select operator has two enablements: a non-persistent relation (from Retrieve) and a persistent set of criteria (enabled at the start of plan execution). Now suppose that this plan fragment was executed multiple times, as if it were part of a loop. It would be wasteful to continually route the same criteria to the Select operator because it would never change during the loop. In contrast, the relation obtained from Retrieve could very well change, since it represents data from a remote, dynamic web site.

Persistent enablements remain active for the duration of plan execution. Furthermore, if they carry data, that data remains 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.

Non-persistent enablements, on the other hand, are simply those that exist until they are consumed. Thus, once an operator executes based on some 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 (and their data) can be queued. Since Theseus is a parallel execution environment, it is often possible to have data queuing at multiple operators that accept non-persistent enablements. This creates the effect of asynchronous data pipelining during plan execution.

3.1.2.1 Uses of Enablements

Enablements have three functions in the Theseus plan language: to act as a mechanism for control flow, to provide synchronization during execution, and to provide a way to pass data between operators.

In terms of control flow, enablements are the basis for how a plan is executed. Operators execute when they receive their input enablements, and this execution may trigger other operators, and so on, defining program flow. Enablements can also be used to implement plan loops. For example, notice that the Iterate operator in the CyberHomes 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.

A third and final purpose of enablements is to route data. By definition, dataflow systems are defined by *arcs* between *actors* that route *tokens*. In Theseus, enablements serve as the arcs, operators are the actors, and the data are the tokens. The notion of persistent enablement, wherein an enablement and its

data are continually available once activated, is a special modification of traditional dataflow to capture a declarative need in plan specification, as well as for efficiency purposes in Theseus.

3.2 Efficiency of Execution

The most important aspect of Theseus is its execution efficiency. By implementing Theseus as an dataflow-style system, we were able to realize high degrees of parallelism and asynchrony, well-known attributes of high performance systems.

3.2.1 Operator Parallelism

Maximizing concurrency is a key requirement for Theseus. Historically, information integration for heterogeneous remote data sources has been a problem that is network-bound. Such systems can only execute as fast as their most latent sources. Motivated by the desire for concurrency found in both parallel database systems and reactive plan execution systems, we built Theseus as a dataflow-style execution machine, to support a high degree of operator parallelism.

In Theseus, operators are implemented as threads (lightweight processes). The system is efficient because the scheduling for these threads is handled by the operating system. Care was taken to limit the amount of synchronization between threads required at run-time. For example, operators are not required to engage in a request-reply style of communication in order to exchange data. Rather, operators can accomplish this process by asynchronously writing to the input queue of another operator. The only synchronization required is that needed when posting the actual enablements corresponding to that data. However, since this posting is a very quick process, the time necessary to lock any global data structures is minimal.

In the CyberHomes plan, operator parallelism is exploited when gathering the details for a particular house. For example, suppose ten houses in a given set of house listings successfully met the search criteria. Since no data dependencies exist between them and because each operator instance is implemented as separate thread of execution, retrieval of details for each can be executed in parallel.

3.2.2 Pipelining

When analyzing a typical data integration plan, such as the one described in Section 2, we can make two important observations. One is that these plans often require loops. For example, in the CyberHomes plan, there is a need to iteratively gather sets of houses until the page that does not have a *Next Listings* link is encountered. The nature of gathering data on the Web is such that iteratively retrieving groups of data is a fairly common occurrence.

Another observation is that the execution time of the various plan operators can vary widely. Specifically, for Web-based data integration, the execution time for a Retrieve operator can be several orders of magnitude slower than the rest of the operators in the plan (such as Project or even Join). This is mainly due to the fact that Retrieve suffers from the latency involved in the access of a remote data source over the network.

Next, consider what is involved in actually collecting the entire set of house listings from CyberHomes. Typically, a plan to do this would involve gathering the current set of houses and waiting for that set to be processed before gathering the next set. The processing of each set is not trivial: a join against those houses already seen (to determine the truly new houses) is required, as well as the network-based retrievals of details for each new house. Even if the retrieval of details for multiple houses can be parallelized, a plan having a loop containing slow operators is something which can substantially affect overall performance.

To improve the efficiency of execution in such cases, Theseus implements data *pipelining*. This feature allows sets of houses to be staged in a pipeline, immediately following retrieval. After the staging of a particular set, the next set can be retrieved, staged, and so on – independent of how much progress has been made in the processing/analysis phase. Eventually, all houses will be retrieved and processed, and the final accumulated set can be returned.

Pipelining in Theseus is realized through enablement queuing. When one operator produces an output enablement, that enablement can be either (a) immediately consumed or (b) queued, along with any data it carries. Operators continue execution independent of this process, de-queuing accumulated input enablements as soon as possible, per their rate of execution. It is both the buffering and asynchrony aspects provided by the queuing process that facilitates pipelining during execution.

4. RELATED WORK

As described earlier, Theseus can be viewed as a cross between general plan executors and parallel

database systems. The key differences are that (a) unlike general plan executors, Theseus is optimized for the information processing domain and that (b) unlike parallel databases, standard techniques for achieving high-performance (such as the shared-nothing approach) are simply not applicable to information management on the Internet, which consists of heterogeneous and distributed data sources, beyond the administrative domain of the execution engine.

Theseus can also be compared with Tukwila [9], which supports efficient query execution on remote, heterogeneous data sources. Like Theseus, Tukwila is interested in data integration, especially the ability to query web sites as if they were databases. The main difference between Theseus and Tukwila is that Theseus uses a hybrid dataflow model of execution while Tukwila uses standard (von-Neumann) control flow model. Key in understanding this difference is in the tradeoffs between dataflow and control flow systems. The former enables automatic, on-demand parallelism with minimal synchronization, while the latter requires manual management of parallelism, often with more frequent points of synchronization.

5. DISCUSSION

In this paper, we have presented the Theseus execution system. We have demonstrated that Theseus is a useful tool for building efficient information management agents. Because our planning language allows complex information management plans to be easily specified, users can build powerful agents. Furthermore, a dataflow-style execution architecture enables these agents to reach a high level of performance. While our system is very useful in its current state, we are exploring additional plan optimization and efficient execution strategies to further improve performance and scalability.

One such effort is related improving the performance of individual plans. Any single plan (user defined or automatically generated by the user-interface) may be sub-optimal. In future work, we would like to explore the improvement of these sub-optimal plans. In previous work on Ariadne, a highly efficient and scalable approach to plan optimization was demonstrated using Planning-By-Rewriting (PBR) [1], a local-search approach to anytime plan refinement. We are investigating the applicability of PBR to our optimization needs.

We are also exploring ways to improve the scalability and throughput of Theseus in a global context. Specifically, we would like to be able to deploy Theseus as an application service and allow multiple users to periodically execute information gathering plans for a particular domain. This may allow users to share the results of a popular query to a dynamic source and may also present other profitable plan merging opportunities.

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 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.

REFERENCES

[1] Ambite, J.L. and Knoblock, C.A. 1997. Planning by Rewriting: Efficiently Generating High-Quality Plans. *AAAI-97*.

[2] Ashish, N.; Knoblock, C.A.; and Shahabi, C. 1999. Selective materializing data in mediators by analyzing user queries.*COOPS-99*.

[3] Barish, G.; Knoblock, C.A.; Chen, Y-S.; Minton, S.; Philpot, A.; Shahabi, C. 1999. TheaterLoc: A Case Study in Information Integration. *IJCAI-99 Workshop on Information Integration*.

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

[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 Eng 2(1).*

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

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

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

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

[10] 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. *AAAI-1998*.

[11] Kushmerick, N. 1997. *Wrapper Induction for Information Extraction.* PhD Thesis, CS Dept. University of Washington.

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

[13] Levy, A.Y.; Rajaraman, A; Ordille, J.J. 1996. Querying Heterogeneous Information Sources Using Source Descriptions. *VLDB-1996*.

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

[15] Myers, K. 1996. A Procedural Knowledge Approach to Task-Level Control. In *Procs of the Third Intl Conf on AI Planning Systems*.

[16] Wilschut, A.N. and Alpers, P.M.G. 1991. Dataflow query execution in a main memory environment. In *Proc. Conf. On Parallel and Distributed Information Systems*