The TheaterLoc Virtual Application

Greg Barish, Craig A. Knoblock, Yi-Shin Chen, Steven Minton, Andrew Philpot, Cyrus Shahabi

Information Sciences Institute, Integrated Media Systems Center, and Department of Computer Science
University of Southern California
4676 Admiralty Way, Marina del Rey, CA, 90292
{barish, knoblock, minton, philpot}@isi.edu {yishinc, shahabi}@pollux.usc.edu

Abstract

Although much has been written about various information integration technologies, little has been said regarding how to combine these technologies together to build an entire "virtual" application. In this paper, we describe the design and implementation of TheaterLoc, an information integration application that allows users to retrieve information about theaters and restaurants for a variety of cities in the United States, including an interactive map depicting their relative locations and video trailers of the movies playing at the selected theaters. The data retrieved by TheaterLoc comes from five distinct heterogeneous and distributed sources. The enabling technology used to achieve the integration includes the Ariadne information mediator and wrappers for each of the web-based data sources. We focus in detail on the mediator technologies, such as data modeling, source axiom compilation, and query planning. We also describe how the wrappers present an interface for querying data on web sites, aiding in information retrieval used during data integration. Finally, we discuss some of the major integration challenges we encountered and our plans to address them.

Introduction

There is a wealth of interesting data sources and applications available on the World Wide Web, but it is difficult to do much with the information except look at it or build a specific application to process the data available. Writing separate applications each time is a time-consuming and redundant task. We have developed a system called Ariadne (Knoblock et al. 1998) that makes it possible to rapidly construct an information agent that can integrate data sources that were not originally designed to work together. The resulting virtual application dynamically performs the integration in order to minimize the problems associated with storing and maintaining data. Ariadne includes tools for constructing wrappers that make it possible to query web sources as if they were databases and the mediator technology required to dynamically and efficiently answer queries using these sources.

We claim that Ariadne makes it possible to rapidly build virtual applications and in this paper we describe exactly what is involved. Specifically, we provide a detailed, behind-the-scenes look at one of the recent applications we have built. This application, called TheaterLoc (Barish et. al. 1999a), integrates data related to movie theaters and restaurants, allowing users to view their locations on a map, look up restaurant reviews and movie showtimes, and watch trailers of films.

There has already been substantial work on information integration (Weiderhold 1996) and projects that focus on applying this technology to the World Wide Web, including Information Manifold (Levy et. al. 1996), Occam (Kwok and Weld 1996), Infomaster (Genesereth et. al. 1997), and InfoSleuth (Bayardo et. al. 1997), as well as related work specifically on information extraction (Hammer et. al. 1997; Doorenbos et. al. 1997; Kushmerick 1997). But what is noticeably absent from the literature is a study on what it takes to put together an entire application using the various integration technologies. To that end, we describe the details of how TheaterLoc works and how it was developed.

The next section describes what the application does from the user's point of view. Then, we describe how it works, including the domain modeling, the query planning, and the wrappers for extracting the data from web pages. Next, we enumerate TheaterLoc development tasks and their costs. Lastly, we identify some remaining challenges and how we are currently addressing them.

The TheaterLoc Application

TheaterLoc (http://www.isi.edu/ariadne/demo/theaterloc) is a web site that allows users to retrieve information about restaurants and movie theaters for various cities in the United States. Users first choose the city in which to query. The system then returns information about the theaters and restaurants in that city, as well as a custom, interactive map identifying their relative locations within that city, illustrated in Figure 1.

Users can then click on any of the plotted points to be taken to a web page containing further details about that particular place. For example, when a restaurant is chosen, users are taken to its corresponding CuisineNet web page (as shown in Figure 2), which contains reviews,

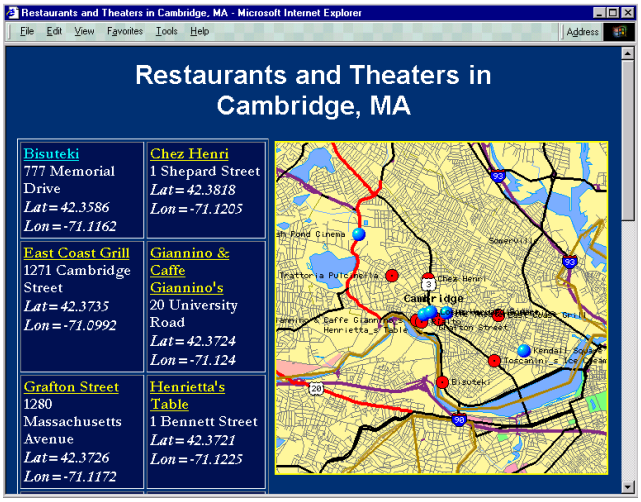

Figure 1: Theaters and restaurants in Cambridge, MA

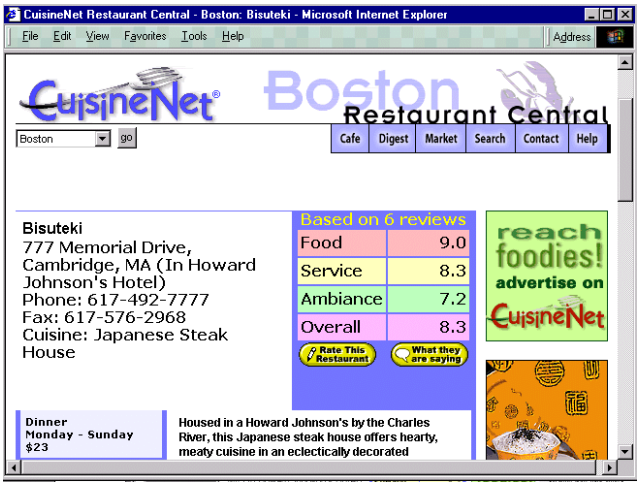

Figure 2: Restaurant detail page

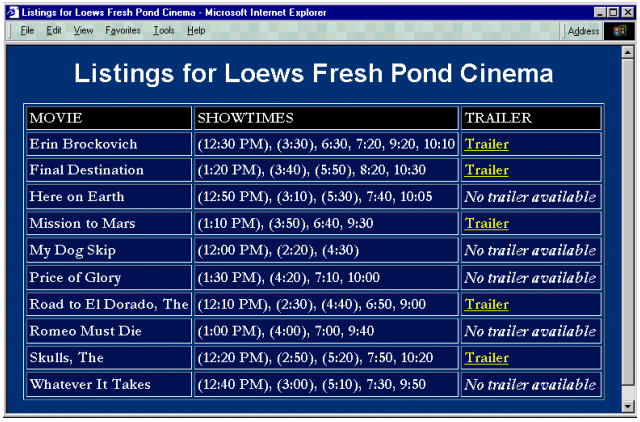

Figure 3: Theater detail page

pricing information, and ratings. Alternatively, choosing a theater returns a listing of the current movies playing there, along with their showtimes, and links to video trailers, as shown in Figure 3.

The information integrated by TheaterLoc comes from five distinct online sources. Restaurant information is gathered from CuisineNet, theater and movie showtime information from Yahoo Movies, and the trailers come from Film.com. Construction of the interactive map is facilitated by two sources: the E-TAK geocoder (to geocode all addresses for plotting) and the US Census Tiger Map Service. The TheaterLoc application is effective because it saves the user from having to go to these sites separately, navigate through different user interfaces, and integrate the data manually. Instead, what is presented is a single, cohesive application that seamlessly integrates the useful data from these sources and automatically correlates them as necessary.

System Architecture

TheaterLoc is a client/server application, where the server side is composed of three major pieces: a web server, an information mediator, and a set of wrappers to access data sources. For the purposes of this paper, we will focus on the details of the mediator and wrappers, since they are the centerpiece of the integration effort.

The system architecture is shown in Figure 4. When a user issues a query though the web interface, the HTTP request is processed by the web server, and a corresponding query is sent to Ariadne for resolution. The mediator, in turn, constructs a plan indicating which sources should be queried and how the data retrieved should be integrated. This plan also contains information about how to order the steps of information retrieval (since there may be dependencies), which steps can be executed in parallel, and what other data manipulation functions (such as relational joins or projections) need to be done to answer the query.

Many of the data sources comprising Ariadne applications are web sites. Access to their data is accomplished through communication with data source *wrappers*, which provide a standard, flexible query interface to a set of logically related web pages. Web pages are considered semi-structured sources, in that they contain useful information, organized in a predictable manner, which can be extracted automatically.

Wrappers are used to parse the data from these web pages, essentially providing a database-like interface to the data contained on those pages. They allow the mediator to interrogate web sites for information in a standard and structured manner, specifically, a subset of the SQL language. The TheaterLoc wrappers and their underlying web site sources are listed in Table 1.

Example Query

To illustrate the details of integration within the system, consider the following example. Suppose a user wants to map restaurants and theaters in Cambridge, Massachusetts.

As described earlier, this HTTP request is translated by the web server into an query sent to Ariadne, which then plans a solution. The resulting plan consists of several

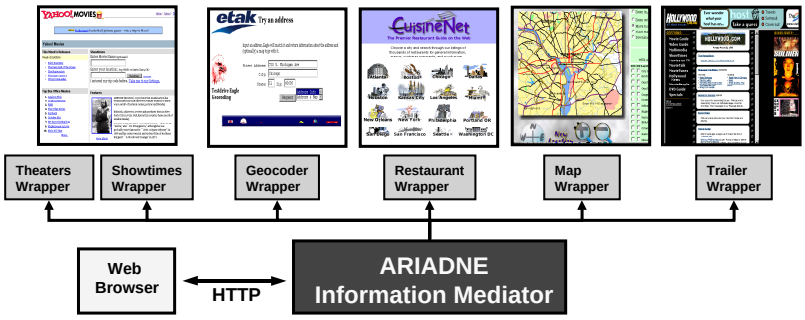

Figure 4: TheaterLoc System Architecture

subqueries to the various data sources that the mediator knows about, so that the desired information can be efficiently retrieved and integrated. For the example query, this plan consists of retrieving information about the various restaurants and theaters in Cambridge, and retrieving a map showing their relative locations.

The detailed plan needed to accomplish these two tasks includes a few additional steps, based on the information sources available. Recall that it is not possible to simply get all of this information from a single source. The mediator must reason about what data the various sources can offer and then construct a plan which retrieves the desired information based on the features, limitations, and dependencies between the sources. These detailed steps are described below.

Retrieving Theaters and Restaurants. For our example query, since the user has chosen to get information on all theaters and restaurants in Cambridge, the mediator will initially determine that it needs to query the Restaurant and Theater wrappers to get demographic information (names, addresses, and URLs) for both types of establishments. In contrast, if the application interface had allowed the user to search only for information about theaters in Cambridge, the mediator would realize that there would be no need to query the Restaurant wrapper, as CuisineNet only provides data about restaurants.

Retrieving the Interactive Map. The next step in the plan for our Cambridge query involves using the Geocoder source to convert theater and restaurant street addresses into latitudes and longitudes. This is necessary in order to construct a query to the Map wrapper, which retrieves a dynamic, interactive map indicating the locations of these places.

The map returned is an HTML "image map", where each plotted point is associated with a hyperlink to a page containing more details about that location. Thus, users can click on a particular point to explore more detail about either a restaurant or theater. If they choose a restaurant they are taken to the CuisineNet page for that restaurant. If, on the other hand, they choose the URL attribute for a theater, another Ariadne query is invoked to collect movie showtime and video trailers for those movies. Again, the web server translates an HTTP-based request into a domain-level query.

Wrapper	Source	URL
Restaurant	*CuisineNet*	www.cuisinenet.com
Theater	*Yahoo*	movies.yahoo.com
Showtimes	*Yahoo*	movies.yahoo.com
Geocoder	*E-TAK*	www.geocode.com
Tiger	*USGS*	tiger.census.gov
Trailer	*Film.com*	www.film.com

Table 1: TheaterLoc wrappers and underlying sources

Retrieving Movie Information. The query to Ariadne for theater details contains the name of the theater chosen. The corresponding plan that the mediator constructs to solve this query consists of two steps: (a) interrogating the Yahoo Movies site via the Showtimes wrapper for information about movie showtimes and (b) for each movie, querying the Trailer wrapper to locate the URL for the video trailer, if any, associated with that movie. The combined information is joined into a single relation and subsequently returned to the user as an HTML table. Users can then view the trailer by clicking on the link provided in this table.

How TheaterLoc Works

We now take a detailed look at the inner workings of TheaterLoc, focusing primarily on the Ariadne mediator and wrapper technologies.

Data Modeling

In Ariadne, relationships between data are expressed in the application *domain model*. This model contains information about classes, their attributes, and their relationship to other classes. The domain model provides a unifying ontology for describing the contents of the sources.

The model supports both *functional sources* and *data sources*. The former essentially has a set of input and output attributes: when given the required input attributes, functional sources perform some computation and produce the output attributes. Data sources, on the other hand, simply contain a relation to be returned. There are often instances when data sources require some input in order to return a relation (for web sites, this is the case when executing an HTTP POST request), so they can be very similar to functional sources. However, in Ariadne,

functional sources are typically local and they always involve computation performed locally. Data sources are either local or remote and, if they do involve computation, that computation is performed remotely.

The TheaterLoc domain model is shown in Figure 5. The model shows the domain level classes of Map, Place, Movie, Restaurant, and Theater. Restaurant, for example, is a class that has several attributes (such as cuisine) and is related to other classes (such as Theater). Classes in the domain model are mapped to zero or more actual information sources. For example, in TheaterLoc, the Restaurant class is mapped to the CuisineNet source.

The directed arc edge from Restaurant to Theater indicates a *covering*, referring to the fact that the only types of Place in TheaterLoc are either restaurants or theaters. Since the Restaurant and Theater classes are sub-classes of Place, they naturally inherit attributes of their parent class, namely: street, city, state, city-state, latitude, and longitude. The sources associated with each class are shown as gray cylinders or cubes, near the class. The cylinders indicate data sources, the cubes indicate functional sources. Also shown for each source is a list of the attributes it provides along with any binding constraints (Kwok and Weld, 1996), the latter prepended with a "$" character. Binding constraints are simply input requirements that a source has before it can provide data.

The Geocoder is an example of a data source that has a binding constraint. It requires street, city, and state attributes in order to provide latitude and longitude information about an address. Intuitively, this type of constraint makes sense: one cannot geocode an address without knowing the address first.

Functional sources can also have binding constraints. The Article-Fn source, for example, takes as input an attribute called raw-movie-nm and returns an attribute called movie-name. This purpose of this source is to normalize the ordering of the words of a movie title, so that semantic equivalence can be detected with another source. Specifically, this source is used to move any grammatical article which might appear at the end of a movie title to the front of the title. For example, the raw-movie-nm might be "Bug's Life, A" and the movie-name returned would be "A Bug's Life".

Although data sources with binding constraints appear similar to functional sources at the modeling level, they are usually different at the implementation level. Functional sources typically perform local computation based on input and derive original data based on that input. In contrast, data sources with binding constraints typically use the input information as a means to either perform remote computation or as a means to filter out a logical subset of data from a much larger set.

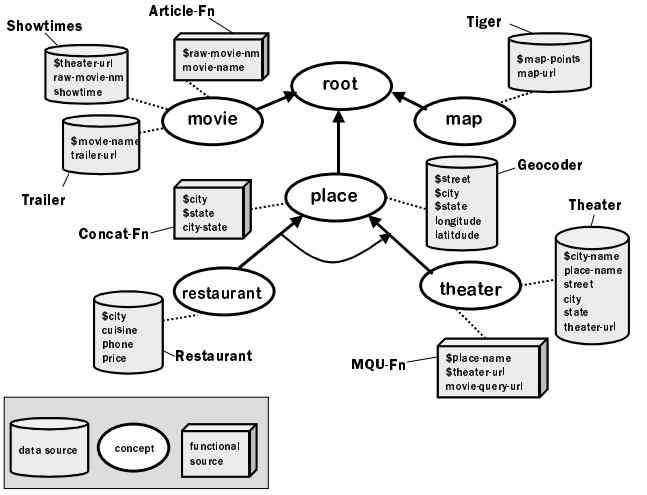

Figure 5: TheaterLoc domain model

Query Planning

Planning in Ariadne consists of two major steps: an initial axiom compilation phase and then a run-time planning phase. The first step is executed once, when the application is first initialized. The second step is executed each time the mediator receives a query.

Axiom Compilation. The reasoning done by the mediator about the domain model leverages the results of an initial domain axiom compilation step (Ambite et. al. 1998) that generates rules about what source combinations can be used to solve various domain queries. Specifically, axiom compilation is based on applying a set of inference rules to construct a lattice describing how various combinations of data modeled at the domain level can be retrieved given the available functional and data sources.

For example, consider the axioms for the TheaterLoc Restaurant domain class, shown in Figure 6. Notice the second axiom, which has a *head* declaring that various attributes of restaurant (such as cuisine and latitude) can be retrieved by combining the CuisineNet and Geocoder sources, as shown in the *body* of the axiom. Essentially, axioms represent how to map domain level terms onto one or more source level terms.

The initial axiom compilation step significantly reduces the run-time execution of the system. Instead of performing a costly search to locate those sources

```
restaurant(?$city _ ?cuisine _ _ _ _ ?phone ?place-name ?place-type ?price ?state ?street ?url _)
  <-> cuisinenet(?street ?$city ?place_type ?cuisine ?url ?place-name ?phone ?price ?state)

restaurant(?$city _ ?cuisine _ ?latitude _ ?longitude ?phone ?place-name ?place-type ?price ?state ?street ?url _)
  <-> cuisinenet(?street ?$city ?place_type ?cuisine ?url ?place-name ?phone ?price ?state) and
      geocoder(?$city ?latitude ?longitude ?$state ?$street)

restaurant(?$city ?city_state ?cuisine _ ?latitude _ ?longitude ?phone ?place-name ?place-type ?state ?street ?url _)
  <-> concatfn(?$city ?city_state ?$state) and
      cuisinenet(?street ?$city ?place-type ?cuisine ?url ?place-name ?phone ?state) and
      geocoder(?$city ?latitude ?longitude ?$state ?$street)
```

Figure 6: Partial list of TheaterLoc axioms

required to answer a given query, the planner can instead quickly consult the pre-compiled axiom lattice.

Planning By Rewriting. When queries are posed to the system, Ariadne reasons about the domain model and source descriptions in order to develop an efficient plan for retrieving and integrating the data. The method used to accomplish this is called Planning-by-Rewriting (PBR) (Ambite and Knoblock 1997). Under PBR, an initial, sub-optimal plan is quickly generated and then iteratively improved by applying a series of rewriting rules. Rewriting relies on local search algorithms that can alter both the sources used to resolve portions of a query as well as the ordering of operations performed by the mediator during information integration.

The resulting plans produced by PBR can significantly optimize and simplify the linear portions of the plan, as well as exploiting opportunities for parallelism between tasks, where possible. For example, the planning for the query about restaurants and theaters in Cambridge would discover that the collection of demographic information and the geocoding of that information was a necessarily serial sequence, whereas the collection of the demographic data from the Theater wrapper and the collection of data from the Restaurant Wrapper were independent plan steps that could be parallelized.

An example plan to locate the theaters in Cambridge, which represents a sub-plan of the original example presented earlier, is shown in Figure 5. Generally, what is illustrated here is that a list of theaters is being retrieved from the Theaters wrapper, geocoded and the relevant attributes returned as output. In addition, for each theater, a *movie-query-URL* (which is the basis for the movie-showtimes query) is derived. In looking at the figure, we can identify a series of plan operators associated with these general tasks. For example, notice that there is a retrieval done for Theaters, the result used as the basis for geocoding (Geocoder retrieve step), and the subsequent results are joined along the *street*, *city*, and *state* attributes. Later, there is a join done between this information and the *movie-query-URL* information, based on *place-name*. Finally, the output contains the attributes of the class, provided by the integrated sources.

Wrapper-based Information Extraction

Wrappers, as described previously, provide a generic mechanism by which a web site can be queried as a traditional database, in a subset of the SQL syntax. In TheaterLoc, for example, when querying the list of restaurants from CuisineNet for Cambridge, the SQL query:

```
select name, address, url from CuisineNet
where city='Cambridge'
```

is issued to the Restaurant wrapper by the mediator. The wrappers work by using a *page model* to describe the location and type of web page(s), an *embedded catalog* to define the hierarchical relationship between data on a page, and a set of *extraction rules* describing how to parse data from that page (Muslea et. al.. 1999).

The page model describes how the pages should be contacted in order to prepare for data extraction. For example, the E-TAK Geocoding site consists of an HTML form that requires address information as input to return the geographic coordinates for that address. Thus, the extraction of those coordinates is contingent on submitting the form (an HTTP POST request). The automatic entering of data onto the form and subsequent POST request are described in the page model.

An embedded catalog is used to model the hierarchical relationships between the attributes on a page. For example, the Showtimes wrapper contains a two-level embedded catalog which describes the fact that each theater page contains a list of one or more movies. The embedded catalog is used as a basis for how to parse a given web page. Multiple levels in the catalog typically indicate list-like structures on pages, so that nested lists of information can be extracted in a structured manner.

Finally, the extraction rules describe how a page should be split into a hierarchy of regions, and where the data is located within each of these regions. Whereas the embedded catalog describes the general tree-like structure of a page, the extraction rules define how to locate the nodes and leaves of that tree, the latter being the actual data to be extracted. Rules are expressed in a regular-expression like syntax, and are based on identifying

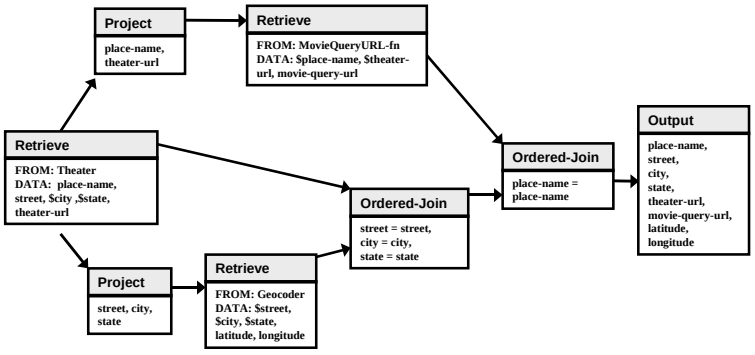

Figure 7: Part of the TheaterLoc query plan

landmarks near where the matching expression will appear.

Generation of the page model, embedded catalog, and extraction rules is accomplished through training the system via a graphical user interface (GUI). Application developers use the wrapper GUI to choose web pages they want to extract data from, as well as where the various parts of data on that page are located – they actually point and click to indicate this information. Using inductive learning, a system called STALKER (Muslea et. al. 1998) generates the rules associated with the user-defined catalog and model.

As an example of how wrappers extract data from a web page, consider the TheaterLoc Showtimes wrapper. As shown in Figure 8, the Yahoo Movies web page for a theater shows a list of movies and their showtimes. Obviously, there are some natural structures and patterns associated with the data for that page. Wrappers take advantage of this semi-structure to perform information extraction. For example, the figure shows that on each page there is a notion of a *movielist*, which is composed of a list of *movies*.

Figure 9a shows the actual page model file for the Showtimes wrapper. Notice that a binding pattern relationship (indicated by the "?" symbol) exists: a URL for a theater must be supplied in order to receive information about movies and showtimes. Figure 9b shows the hierarchical embedded catalog for the same wrapper. The movielist/movie relationship, as described above, is captured here. Finally, Figure 9c presents the extraction rules. These rules describe how to locate relevant data on a web page. Notice that they are somewhat related to the embedded catalog, in the sense that hierarchical relationships must have special rules which show how to locate multiple child instances. For example, the notion that *movielist* contains one or more *movies* requires that the extraction rules specify not only where the *movielist* can be found on the page, but also how to iterate through it, so that multiple instances of its children can be identified.

It is also interesting to note the two-level embedded catalog which mirrors the list-like structure of the actual web page (Figure 8), where each theater contains a list of movies, and each movie has a set of showtimes. Notice that we also could have extended the catalog to a third-level, to capture the *list* of showtimes, instead of just the showtimes as one large string. But, that sort of

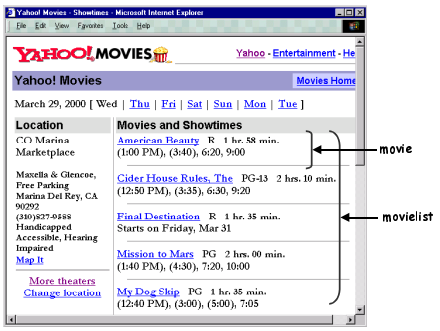

Figure 8: Web page for Yahoo! Movies

(a)

```
sourcetype PAGE
sourceURL ?URL
attributes:
    MOVIEstring,
    SHOWTIMES:string,
    URL:url
```

(b)

```
g ROOT MOVIELIST
e MOVIELIST MOVIE.SHOWTIMES
```

(c)

```
MOVIELIST item
Begin_Rule
  SkipTo(Movies) SkipTo(and)
  SkipTo(Showtimes)SkipTo(<A HRef=)
End_Rule _BE_
  SkipTo(module movies) SkipTo(</table>)
  SkipTo( ) SkipTo( )
_EOI_

MOVIELIST iter
Begin_Rule
  SkipTo(<A HRef=)
End_Rule _FE_
  SkipTo(<tr>) SkipTo(<td>) SkipTo(<hr)
_EOI_

MOVIE item
Begin_Rule
  SkipTo(<b>)
End_Rule _FE_
  SkipTo(</b>)
_EOI_
```

Figure 9 (a) page model, (b) emedded catalog, (c) extraction rules

enumeration would not be useful at the application level (we do not need to extract the individual showtimes), so we used just two levels.

The Development Process

We constructed TheaterLoc in a very short amount of time. Table 2 enumerates the time spent on each part of the development process. In total, building TheaterLoc required 3-4 days. One of the more important things to note from this table is that, with the exception of the Tiger Map wrapper, using our wrapper GUI tool to automate the construction of all TheaterLoc wrappers cost about 3 hours of total project time. The Tiger wrapper required special integration (coordinate translation) and ended up taking us an entire day to complete.

Task	Time Required
Application design	*3-4 hours*
Domain modeling	*2-3 hours*
Wrapping web sources (w/GUI)	*2-3 hours*
Tiger Map wrapper	*1 day*
Building functional sources	*1-2 hours*
Interface/HTML	*2-3 hours*
Web server integration	*3-4 hours*
Application testing	*2-3 hours*
Total	**3-4 days**

Table 2: Project development Time

The tools and approach that we used to build TheaterLoc also make it an application that is easy to maintain and extend. For example, adding a new source into the application is simply a matter adding this source to the domain model and using the wrapper GUI tool to automate its construction. Similarly, dealing with changes to the sources is also a straightforward process that does not require re-engineering the entire system.

It is worthwhile to note the amount of reusability inherent in virtual applications like TheaterLoc. The wrappers, in particular, could be integrated into another virtual application without any modifications. For example, if some future application required plotting hospitals and schools on an interactive map based on their street addresses, the geocoder and map wrappers from TheaterLoc could simply be reused for this purpose.

Challenges and Extensions

Using the Ariadne technologies we have described allows us to rapidly build information integration applications. However, there are still some challenges that remain. In this section, we list two of these challenges and briefly describe our ongoing research at addressing them.

Resolving Data Inconsistencies

One problem frequently encountered when integrating data from multiple sources involves semantically equivalent objects that exist in inconsistent text formats across these sources. With TheaterLoc, for example, although the movie "A Bug's Life" was listed by Film.Com as such, Yahoo Movies listed it as "Bug's Life, A". This made it difficult for us to correlate the two objects, something we needed to do when associating a movie and its showtimes with a video trailer. As described earlier, we solved this problem for TheaterLoc specifically by building a functional source that normalized the textual format of this data. However, it is clear that a more general solution is necessary.

Towards that end, we have been designing an approach that identifies semantic equivalence by matching all of an objects' shared attributes (Tejada et. al. 1998). With our technique, certain attributes have more importance (weight) in deciding a match than others. For example, in TheaterLoc, we could resolve the similarity between "A Bug's Life" and "Bug's Life, A" by computing a similarity metric between all of the shared attributes (such as movie title, director, and actors) and then judge equivalence based on this metric. Since the manual encoding of attribute weighting is time consuming and error-prone, we are also developing an active learning approach for tailoring attribute weighting rules, through limited user input, for specific application domains.

Improving Performance and Scalability

Data Materialization. At their core, information integration applications are only as fast as their most latent sources. One slow website can substantially affect overall application performance. For TheaterLoc, a major bottleneck was the Tiger map source, which occasionally took several seconds to render a map. In addition, web-based data sources are not always reliable. For example, there were times when CuisineNet was temporarily unavailable or simply overloaded with requests.

As a remedy for these issues, we are investigating the optimization of data access by selective pre-fetching and caching of source data (Ashish et. al. 1998). Since information integration applications are frequently associated with very large databases, we must be careful to cache only the subset of data that returns the greatest improvement to overall application performance. Our selective approach is based on the frequency of queries, as well as other source-specific metadata, such as source responsiveness. We are also exploring a solution for highly fragmented classes, where a single class may be associated with many sources. In this case, we would like to collapse the cache into a minimal set of classes.

Dataflow Execution. Web-based information integration usually involves retrieving data from multiple web sites at once and applying a series of relational algebra operations (i.e., Select, Join) to achieve a final result. Complicating this are instances where retrieving a logical set of data (a logical relation) involves extracting data from a series of linked web pages. In general, execution could be optimized by (a) parallelizing as much of the data retrieval as possible and (b) streaming retrieved data back to the plan so that it can be processed as soon as possible.

To accomplish this, we are developing the Theseus plan execution system (Barish et. al. 1999b). Based on a hybrid dataflow architecture, Theseus naturally supports the high degree of parallelism and data pipelining that web-based information integration demands. Eventually, we intend to combine both Ariadne and Theseus, such that former is responsible for plan generation and latter is responsible for execution.

Discussion

In this paper, we have described TheaterLoc, an example of a virtual application that integrates data from a set of independent online data sources. We have also described the details of the information integration technology that was used to build TheaterLoc. In particular, we have focused on how state-of-the-art artificial intelligence techniques (planning, knowledge representation, information extraction, and machine learning) were combined to produce the final result. In addition, we have shown that using Ariadne technology makes virtual application development both simple and quick.

It is important to note that TheaterLoc is merely one example of the type of virtual application that can be built using Ariadne. The enabling technologies described in this paper are generic enough to be readily applied to any information domain. In addition, future applications can easily reuse parts of existing ones, further expediting the development process.

New applications are being deployed on the Internet at a rapid rate. While many all offer some measure of independent usefulness, they lack integration with each other. The technology of information integration, such as that embodied by Ariadne, promises a future in which developers can have the power to mix-and-match useful

data from any number of these sources to create endless types of novel virtual applications.

Acknowledgements. This work was supported in part by the Integrated Media Systems Center, a NSF Engineering Research Center, in part by research grants from NCR and General Dynamics Information Systems, in part by NASA/JPL under contract number 961518, in part by the Rome Laboratory of the Air Force Systems Command and the Defense Advanced Research Projects Agency under contract number F30602-98-2-0109, and in part by the United States Air Force under contract number F49620-98-1-0046. Views and conclusions contained in this article are the authors' and should not be interpreted as representing the official opinion or policy of the above organizations or any person connected with them.

We would like to also thank the rest of the Ariadne team: José Luis Ambite, Yigal Arens, Naveen Ashish, Dan DiPasquo, Kristina Lerman, Ion Muslea, Maria Muslea, Jean Oh, and Sheila Tejada. We are also grateful for the help of Chris Stuber, at the USGS Tiger Mapping Service, for his coordinate system translation assistance.

References

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

Ambite, J.L. and Knoblock, C.A. 1998. Flexible and Scalable Query Planning in Distributed and Heterogeneous Environments. *Proc of 4th Intl Conf on Artificial Intelligence Planning Systems*, Pittsburgh, PA.

Ambite, J.L.; Knoblock, C.A.; Muslea, I.; and Philpot, A. 1998. Compiling Source Descriptions for Efficient and Flexible Information Integration. USC/ISI Technical Report.

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

Barish, G.; Knoblock, C.A.; Chen, Y-S.; Minton, S.; Philpot, A; Shahabi, C. 1999a. TheaterLoc: A Case Study in Information Integration. *IJCAI-99 Information Integration Wkshp*.

Barish, G.; DiPasquo, D.; Knoblock, C.A.; Minton, S. Efficient Execution for Information Management Agents. 1999b. *ACM CIKM Workshop on Web Information and Data Management.* Kansas City, MO, USA.

Bayardo Jr., R.J.; Bohrer, W.; Brice, R.; Cichocki, A.; Fowler, J.; Helal, A.; Kashyap, V.; Ksiezyk, T.; Martin, G.; Nodine, M.; Rashid, M.; Rusinkiewicz, M.; Shea, R.; Unnikrishnan, C.; Unruh, A.; and Woelk, D. 1997. InfoSleuth: Agent-based semantic integration in open and dynamic environments. *Proceedings of ACM SIGMOD-97*.

Doorenbos, R.B.; Etzioni, O.; and Weld, D.S. 1997. A scalable comparison shopping agent for the world-wide-web. *Agents-97*.

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

Hammer, J.; Garcia-Molina, H.; Nestorov, S.; Yerneni, R.; Breunig, M.; and Vassalos, V. 1997. Template-based wrappers in the TSIMMIS system. *Proceedings of ACM SIGMOD-97*.

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-98*, Madison, WI.

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

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

Levy, A.Y; Rajaraman, A.; and Ordille, J.J. 1996. Query-answering algorithms for information agents. *Proceedings of AAAI-96*.

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"*, Madison, WI.

Muslea, I.; Minton, S.; and Knoblock, C.A. 1999. A Hierarchical Approach to Wrapper Induction. *Agents-99*, Seattle, WA.

Tejada, S.; Knoblock, C.A.; and Minton, S. 1998. Handling inconsistency for multi-source integration. Technical Report, *AAAI-98 Workshop on "AI & Information Integration"*, Madison, WI.

Weiderhold, G. 1996. *Intelligent Integration of Information*. Kluwer.