TheaterLoc: A Case Study In Building An Information Integration 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}@cs.usc.edu

While there has been much written about various information integration technologies, there has been little said regarding how to combine these technologies together to build an entire 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 cities in the Los Angeles area, including an interactive map depicting their relative locations and video trailers of the movies playing at the various area 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, wrappers for each of the web-based data sources, and a video server to stream the movie trailers. 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 problems we encountered, as well as our plans to deal with them.

1 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[1] [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 agent 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 new information agents and in this paper we describe exactly what is involved in putting together an information integration application. This paper provides a detailed behind-the-scenes look at the pieces that comprise one of the recent applications we built. This application, which we call TheaterLoc, integrates data on movie theaters and restaurants, allowing the user to view the information on a map, look up restaurant reviews, movie showtimes, and view trailers of films.

There has already been substantial work on information integration [Weiderhold 1996], including those projects which focus on applying that technology to the Web, such as the Information Manifold [Levy *et al.* 1996], Occam [Kwok and Weld 1996], Infomaster [Genesereth *et al.* 1997], and InfoSleuth [Bayardo Jr. *et al.* 1997], as well as work done specifically about 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 what our plans are for extending the application.

[1] In Greek mythology, Ariadne gave Theseus the thread with which to find his way out of the Minotaur's labyrinth.

The next section describes what the application does from the user's point of view. Then in Section 3, we describe how it works, including the modeling of the domain, the axioms for integrating the sources, the planning for efficiently processing sources, and the wrappers for extracting the data from the web pages. In section 4, we identify some of the main challenges and how we addressed these challenges or are planning to address them. Finally, we conclude with a discussion of how this application compares to what is available from other web sites, describe another application that we are developing, and identify additional research issues that we will be exploring.

2 The TheaterLoc Application

2.1 Overview

TheaterLoc allows users to retrieve information about restaurants and/or movie theaters in cities in and around Los Angeles county. The application is accessed via the Web using any Web browser.[2] As shown in Figure 2.1, users choose whether they want to gather information about restaurants or theaters (or both) and then choose the city in which they are interested. The system returns information about the theaters and restaurants in that city, as well as a custom, interactive map identifying their relative locations within that city, as illustrated in Figure 2.2.

Users can then click on any of these plotted points to be taken to a web page containing further details about that particular place. For example, when a restaurant is chosen, as shown in Figure 2.3, users are taken to its corresponding CuisineNet web page, which contains reviews, pricing information, and ratings. Figure 2.4 shows that, if a theater is chosen from the map, users are presented with a listing of the current movies playing at that theater, along with their showtimes and links to video previews. Clicking on the preview link initiates a video streaming session in which the trailer for the movie is played.

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 previews come from Hollywood.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 Server.

The TheaterLoc application is effective because it shields the user from having to go to these sites separately, navigate through different user interfaces, and integrate the data manually. Instead, clients are presented with a single, cohesive application which seamlessly integrates the useful data from these sources and automatically correlates them as necessary.

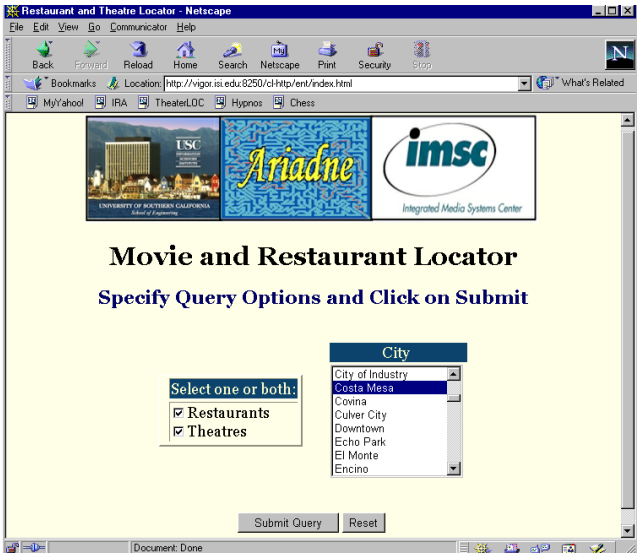


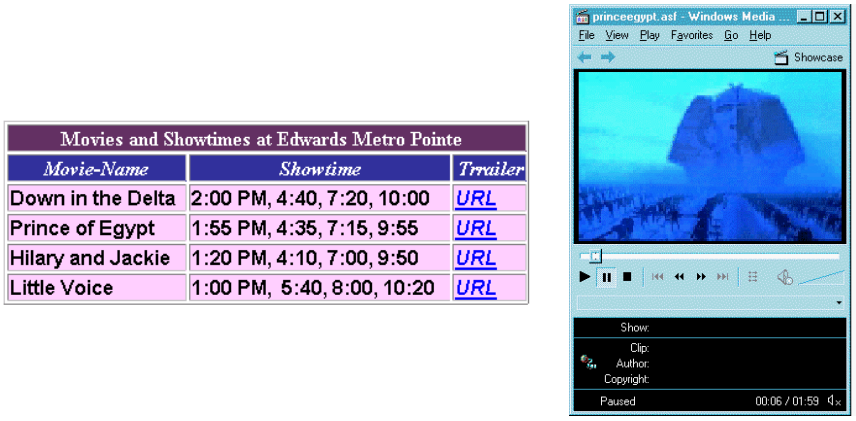
Figure 2.1: TheaterLoc user interface

[2] To watch the video previews, users must also have installed the Microsoft Media Player plug-in

Restaurants and Theatres in Costa Mesa							
Place-Name	Type	Latitude	Longitude	Street	City	State	Url
Edwards Cinema	Theatre	33.6733	-117.92	1534 Adams Ave.	Costa Mesa	CA	URL
Edwards Cinema Center 4	Theatre	33.6704	-117.919	2701 Harbor Blvd	Costa Mesa	CA	URL
Edwards Metro Pointe Stadium 12	Theatre	33.6896	-117.896	901 South Coast Dr.	Costa Mesa	CA	URL
Edwards Triangle Square 8	Theatre	33.6424	-117.918	1870 Harbor Blvd.	Costa Mesa	CA	URL
Skewers	Restaurant	33.6332	-117.914	298 East 17th Street	Costa Mesa	CA	URL
Mi Casa	Restaurant	33.6332	-117.914	296 East 17th Street	Costa Mesa	CA	URL
Henry-n-Harrys' Goat Hill Tavern	Restaurant	33.6395	-117.919	1830 Newport Avenue	Costa Mesa	CA	URL
Garduno's	Restaurant	33.6351	-117.918	209 East 17th Street	Costa Mesa	CA	URL

Figure 2.2: List of movies and restaurants, along with interactive map

Figure 2.3: Detail page for restaurant



Movies and Showtimes at Edwards Metro Pointe		
Movie-Name	*Showtime*	*Trailer*
Down in the Delta	2:00 PM, 4:40, 7:20, 10:00	URL
Prince of Egypt	1:55 PM, 4:35, 7:15, 9:55	URL
Hilary and Jackie	1:20 PM, 4:10, 7:00, 9:50	URL
Little Voice	1:00 PM, 5:40, 8:00, 10:20	URL

Figure 2.4: Theater/Movie detail page, with showtimes and video preview

2.2 System Architecture

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

The system architecture is shown in Figure 2.5. When a client issues a query though the web interface, the HTTP request is processed by the web server, and a corresponding query is sent to Ariadne to be resolved. The mediator, in turn, constructs a plan regarding 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 joins or projections) need to be done for purposes of integration.

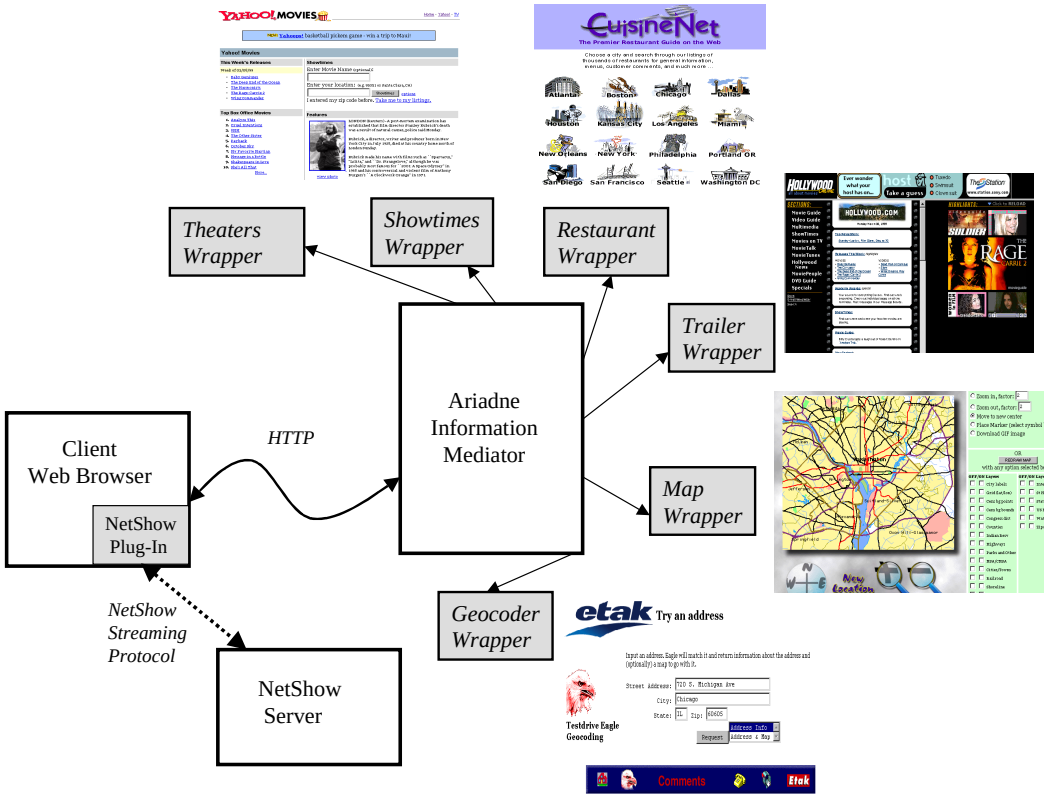


Figure 2.5: TheaterLoc System Architecture

The mediator relies on both *functional sources* and *data sources* in order to get the information it will integrate. Many of the data sources typically accessed by Ariadne applications are web sites. Access to the data on these sites is accomplished through communication with data source *wrappers*, which represent a standard, flexible query interface to a set of logically related web pages. Many web pages are semi-structured data 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 the sites they interface with are listed in Table 2.1

Wrapper Name	Web Source	HTTP Address
Restaurant	CuisineNet	http://www.cuisinenet.com
Theater, Showtimes	Yahoo Movies	http://movies.yahoo.com
Geocoder	E-TAK Geocoder	http://www.geocode.com/eagle.html-ssi
Tiger	USGS Tiger Map Service	http://tiger.census.gov
Trailer	Local Web Site	http://imsc.usc.edu/

Table 2.1: TheaterLoc Wrappers

2.3 Example Query

To illustrate the details of integration within the system, consider the following example. Suppose a user wants to get information about the restaurants and theaters in Costa Mesa. After the user submits the query form for processing, the first thing that happens is that the web server hands off the contents of the request to a CGI/bin-type application, so that a corresponding query, based on the web page form values, can be sent to the information mediator.

Once this query is received, the mediator plans its solution. The resulting plan consists of several subqueries to various functional and data sources that the mediator knows about, so that the desired information can be efficiently retrieved and integrated. Generally, the plans for this initial TheaterLoc query consist of the following two abstract steps: retrieving information about the various restaurants and theaters in Costa Mesa, and constructing a map showing where in that city they are located.

The detailed plan needed to accomplish these two tasks includes a few additional steps, based on the functional and data 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.

2.3.1 Retrieving Theater and Restaurant Demographics

For our example query, since the user has chosen to get information on all theaters and restaurants in Costa Mesa, the planner 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 user had requested only to get information about the theaters in Costa Mesa, the mediator would realize that there would be no need to query the Restaurant wrapper, since the CuisineNet source only provides information about restaurants.

It is interesting to note the URL attribute associated with each restaurant and theater. For restaurants, that URL simply points to a specific CuisineNet page which describes the location in more detail (reviews, menus, etc). However, in the case of theaters, the URL actually represents an HTTP encoded form of

another Ariadne query, to request detailed information about the theater. This query and its associated plan are described later in this example.

2.3.2 Retrieving The Interactive Map

The next step in the plan for our Costa Mesa query involves contacting the Geocoder wrapper to gather geographic coordinate information for each theater and restaurant. This data is needed for the step which follows, which is to construct an interactive map depicting the locations of these places. The source which provides the map, the Tiger wrapper, is actually a functional source which takes, as input, a list of places to be plotted, along corresponding hyperlink references. The Tiger wrapper then contacts the US Census Tiger Map Server to generate a custom map for the points plotted (the map is correctly sized, roughly double the maximum distance between the plotted points).

Based on the map returned, the Tiger wrapper also calculates how the geographic coordinates should be translated to Cartesian coordinates for each establishment, plots a map point, and associates a hyperlink with that point. This association is done by way of an HTML image-map, which allow hypertext links to be embedded in different regions of a single image. In short, the graphical points on the map shown in Figure 2.4 are linked to other web pages: in particular, to the detailed pages about either the particular restaurant or theater to which the point refers.

Now that the demographic and map information has been gathered, both are returned to the user on an HTML page, as shown in Figure 2.2 Since the URL attribute is included both in the summary table and as an embedded hyperlink in the map, users can choose to explore more detail about either a restaurant or theater. If they choose a restaurant, such as "Henry and Harry's Goat Hill Tavern," they are taken to the CuisineNet page for that restaurant (Figure 2.3). 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 (Figure 2.4). Again, the web server takes the responsibility of translating a web-based request into a domain-level query.

2.3.3 Retrieving Movie Showtime and Previews

Once received by Ariadne, the query for theater details contains the name of the theater which is selected. In our example, we have selected the Edwards Pointe Metro Cinema, listed in Figure 2.4. The 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 on the showtimes of movies playing at that theater 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. This relation is shown as an HTML table, in Figure 2.4.

If the user chooses the link for a movie trailer, a request is made to a video server, which then initiates a video streaming session. At this point, the user's video client is activated and the trailer can be viewed. Figure 2.4 shows the viewing of the Prince Of Egypt trailer.

3 Behind the Scenes at TheaterLoc

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

3.1 The Ariadne Information Mediator

3.1.1 Data Modeling

In Ariadne, relationships between data are expressed through the *domain model* for the application. The model contains information about classes, their attributes, and their relationship to other classes. For example, in TheaterLoc, **Restaurant** is a class. It has several attributes, including cuisine-type. It is also related to other classes, such as **Theater**. Classes can be mapped to zero or more actual data or functional

sources. For example, an abstract class may exist only for purposes of unifying related classes, and not be related to any direct sources. Its instantiation is meaningless, but it is useful for grouping like classes and implementing inheritance at the modeling level. On the other hand, a class may actually be represented by two sources, each of which contributes unique or duplicate attributes to a single class.

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 3.1. This model shows the domain level classes of **Map**, **Place**, **Movie**, **Restaurant**, and **Theater** (all are related to the **Root** class, which unifies the system). 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.

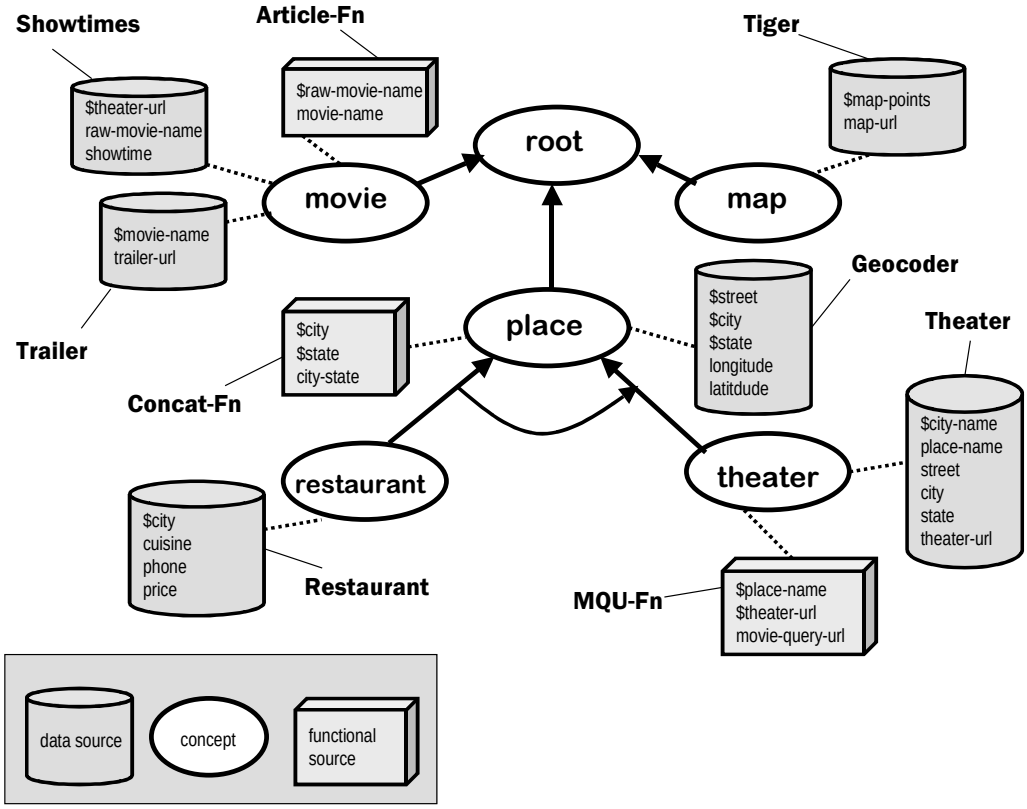


Figure 3.1: TheaterLoc domain model

Also shown for each source is a list of the attributes provided by that source, as well as any binding constraints [Kwok and Weld, 1996] associated with the 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**.

One functional source listed in the model is the Article-Fn source, which takes as input an attribute called raw-movie-name and then 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 (we will describe this issue in more detail in Section 4). 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-name might be "Bug's Life, A" and the movie-name returned would be "A Bug's Life".

As an example of a data source, consider the Geocoder. Although this is a data source, it also has a notion of required input attributes – street, city, and state – in order to retrieve latitude and longitude. This kind of constraint is an example of a source containing a binding constraint.

Since it is the central part of the system which plans and gathers data from the various sources, the mediator also serves as the data integration mechanism. Since the individual sources have no knowledge of each other (or of the domain model), the mediator naturally assumes this role. Thus, filtering tasks, such as the joining data based on various constraints are handled at this level. In the example Costa Mesa query, described in Section 2.2, joins are done by the mediator between theater/restaurant demographics and the corresponding geographic coordinate information for those places.

3.1.2 Query Planning

Planning in Ariadne consists of two major steps: an initial axiom compilation phase and then the runtime planning phase. The first step is executed once, when the application is first initialized. The second step is executed dynamically, on a per query basis.

3.1.2.1 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], which generates rules about what source combinations can be used to solve various domain queries (and subqueries). 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 3.2. Notice the second axiom, which has a *head* declaring that various attributes of restaurant (such as cuisine type and latitude) can be retrieved by combining the source level data sources of CuisineNet and Geocoder (the *body* of the axiom). Essentially, axioms represent how to map domain level terms onto one or more source level terms.

```
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 3.2: Axioms for the Restaurant class

As shown, the axioms depict both domain and source level classes. The "$" symbol in the source level description for the *Geocoder* indicate that the **latitude** and **longitude** attributes can be returned by providing **city**, **state** and **street**. Intuitively, this makes sense: one cannot geocode something in geographic coordinates without knowing its physical address. This dependency is referred to as an axiom

binding pattern, and it is one of the source-level dependencies that the planner must negotiate when resolving a domain level query.

Another interesting aspect of the **Restaurant** axioms involves a relationship to one of the functional sources (*Concat-Fn*). The third axiom declares that the **city-state** attribute can be generated by the *Concat-Fn* source, as long as the **city** and **state** attributes are supplied. Data sources with binding patterns appear similar to functional sources, at the interface level, but they are usually different at the implementation level. Functional sources usually perform local computation based on input and derive original data based on that input. In contrast, data sources with binding patterns 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.

Axiom compilation significantly reduces the run time execution of the system. The planner no longer needs to perform a costly search to identify the combinations of sources which can fulfill the needs of a particular domain query or subquery.

3.1.2.2 Plan Generation and Optimization

As mentioned earlier, Ariadne reasons about the domain model and source descriptions in order to develop an efficient plan for retrieving and integrating the data. The planning 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 associated with resolving the query about restaurants and theaters in Costa Mesa would identify 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 Costa Mesa, which represents a sub-plan of the original example presented in Section 2, is shown in Figure 3.3. 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 <street, city, state>. Later, there is a join done between this information and the movie-query-URL information, along the condition <place-name>. Finally, the output contains the attributes of the class, provided by the integrated sources.

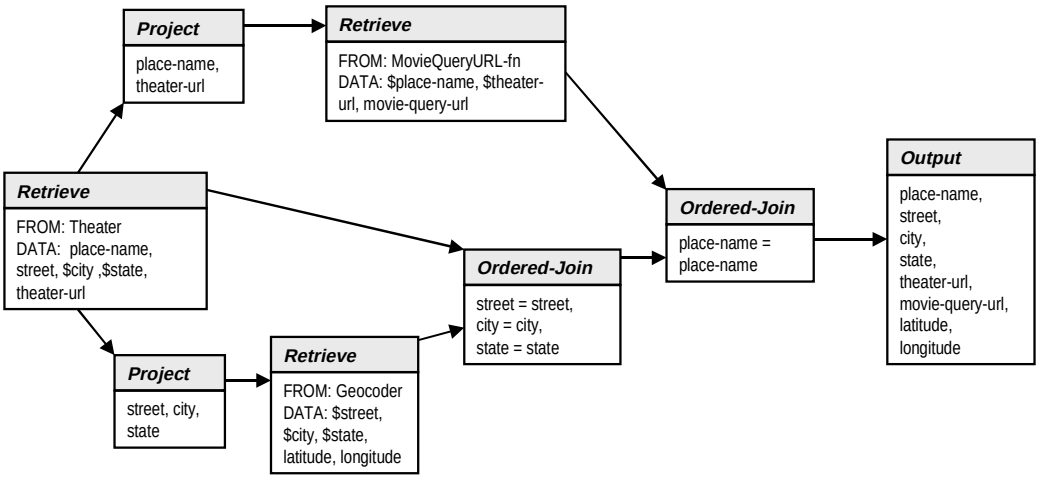


Figure 3.3: Fragment of plan to extract restaurant demographics and geographic coordinates

3.2 Automatic Wrapper-based Information Extraction

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

select name, link, address from CuisineNet where city='Westwood'

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 which takes address information as input and returns 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 components, and where the data is located within those components. 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 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. Application developers use the 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 induction 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 semi-structured source, such as a web page, consider the TheaterLoc Showtimes wrapper. As shown in Figure 3.4, the Yahoo Movies web page for the "AMC Santa Monica 7" 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*.

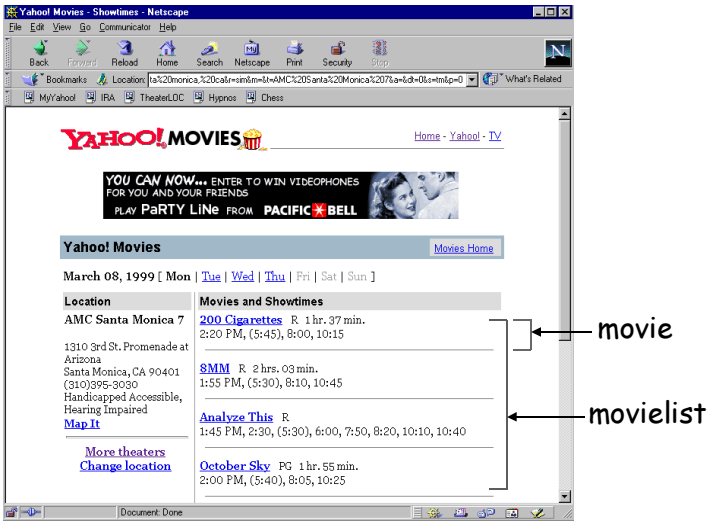


Figure 3.4: the Yahoo! Movies page showing theater name and showtimes

Figure 3.5 shows the actual page model file for the Showtimes wrapper. Notice that a binding pattern relationship exists: a URL for a theater must be supplied in order to receive information about movies and showtimes. Figure 3.6 shows the hierarchical embedded catalog for the same wrapper. The movielist/movie relationship, as described above, is captured here. Finally, Figure 3.7 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.

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

Figure 3.5: Page model file for the Showtimes wrapper

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

Figure 3.6: Embedded catalog for the Showtimes wrapper

It is also interesting to note the two-level embedded catalog which mirrors the list-like structure of the actual web page (Figure 3.7), 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 enumeration would not be useful at the application level (we do not need to extract the individual showtimes), so we used two levels.

```
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_

SHOWTIMES item
 Begin_Rule
   SkipTo(<br>)
 End_Rule  _FE_
   SkipTo( )
_EOI_
```

Figure 3.7: STALKER extraction rules for the Showtimes wrapper

4 TheaterLoc Challenges and Extensions

Like most information integration efforts, building the TheaterLoc application was not without challenges. Bringing together information from distributed, heterogeneous sources often requires some extra "glue" to address outstanding integration problems. One of our ongoing goals with Ariadne and its related technologies is to reduce the amount of glue necessary to rapidly build information integration applications.

In this section, we describe the various problems we encountered while building TheaterLoc. These dilemmas are characteristic of the problems frequently associated with designing and implementing information integration applications. We have active areas of research focusing on addressing many of these issues.

4.1 Resolving Data Inconsistencies

One ongoing issue related to data source integration has to do with finding a way to normalize semantically identical instances of data between two or more sources. In TheaterLoc, we encountered a problem where the Yahoo Movies site contained spelling and punctuation for movies which differed from that at Hollywood.com, where we located corresponding trailers.

For example, Yahoo Movies listed the movie "Bug's Life, A" whereas Hollywood.com listed that movie as "A Bugs Life". At first glance, this does not seem like a difficult problem. One could write a function, such as *move-the-article-to-the-front*, to do the conversion – and we did, in this case. However, different types of data require different conversion rules or functions, in order to aid in the detection of semantic equivalence. In short, the same article-moving trick would not be sufficient when trying to detect, for example, that "Ten Things I Hate About You" and "10 Things I Hate About You" refer to the same data item. Detecting semantic equivalence is a common requirement in information integration, since integrated sources often express the same data differently.

Closely related to this issue is the idea that, when looking at domain classes, certain attributes are more useful as a basis for distinction than others. For example, consider two Restaurant wrappers (one uses the Zagats survey, the other uses Fodors). The first might list a restaurant in Hollywood (neighborhood of Los

Angeles) while the other might simply list the same restaurant as being in Los Angeles. Intuitively, we know that a better way to resolve whether these are two different restaurants is to look at whether their name and phone numbers are the same. This type of semantic knowledge is useful when reconciling the data from two different sources.

We are actively exploring a source-independent solution for intelligently comparing data between multiple sources in which the data may not appear exactly the same, but can be detected as semantically similar [Tejada *et al.* 1998]. Our approach generally involves constructing mapping tables and functions to denote equivalencies between variant representations of the same data. Analysis of the types of data being normalized can be used to guide what parts of the data can be ignored when attempting to detect semantic equality. Intelligent data comparisons for purposes of detecting semantic equality must be done based on the domain of the data being compared. We are exploring how to learn such domain-dependent transformations in a domain-independent manner.

4.2 Optimizing the Performance of the Mediator

One typical trait of information integration applications is that they are slow. This should not be surprising, given the nature of the implementation: at their core, integrated applications are only as fast as their most latent sources. Thus, one slow website can substantially affect overall application performance. In TheaterLoc, we encountered a speed issue with the Tiger source, which occasionally took several seconds to render a map. Another related problem is that some data sources, especially those which are web-based, are not always reliable. In TheaterLoc, we found that the Geocoding site was not always working or was temporarily out-of-service.

As a local remedy for these issues, we are investigating the optimization of data access by selective materialization of data sources. Since information integration applications are frequently associated with very large databases, we must be careful to locally materialize [Ashish *et al*. 1998] only those instances of data which will bring the greatest improvement to the overall application performance. Our approach to selective materialization is based on the frequency of queries to that data, as well as other source-specific metadata, such as the metric of source (web site) responsiveness. We are also exploring the approach of resolving highly fragmented classes, where a single class may be associated with many sources. We would like to collapse the materialization of this data, to a minimal set of classes.

4.3 Automatic Maintenance via Autonomous Data Gathering

One of the difficulties in integrating the Hollywood.Com source was that the video previews it offered were in multiple formats. Often, they were in Microsoft Windows Media format (suitable for Microsoft Media Player clients), but sometimes only an Apple Quicktime version of the trailer existed. To address this problem, we periodically converted the trailers from the Hollywood.com site to be in Windows Media format, and then stored them locally. We needed to do this on a regular basis, since it is typical for a few new movies to be released each week. In general, this sort of data maintenance was actually representative of a larger issue: how to automatically gather data and the perform various operations based on that data. More specifically, in TheaterLoc, some sort of function was needed to automatically scan for new movies and then update the local trailers site, as necessary.

Initially, we approached this problem by building a script which would automatically download the new trailers from Hollywood.com and then convert them to the Windows Media format, if needed. Of course, one weak aspect of this process is that someone still needed to run the script. Obviously, an improvement to this it would be preferable to have some automatic means for checking for new movie trailers on Hollywood.Com and then do the conversion. We felt that, since the automatic gathering of data from web based sources was something we already knew how to do, it would be interesting to extend that to the class of checking for new information on a web site and then potentially using that information to update a local site (which would be update-friendly, not the typical read-only scenario wrappers face) via the Web.

This goal fits in well with Theseus, a new project we are working on which is closely related to Ariadne. Theseus builds on Ariadne technology by extending the web-based data gathering and integration paradigm to the goals of supporting more complex operations on that data (checking when new data is available) as well as more powerful constructs, such as looping and conditional data gathering. The looping aspect of

our data gathering language in Theseus will allow us to implement the periodic, automatic updating of movie trailers from Hollywood.Com, as they are released.

5 Conclusion and Future Work

The tools and approach that we used to build TheaterLoc not only make it possible to rapidly assemble the application, they also provide an application that is easy to maintain and extend. For example, adding a new source into the application is simply a matter of building the wrapper for the source and then describing the contents of this source using the terms from the domain model. Similarly, dealing with changes to the sources is also a straightforward process that does not require reengineering the system.

We were able to construct TheaterLoc in a very short amount of time. It is difficult to provide a precise estimate on the time required since much of the effort went into building, extending, and refining the tools. If we were to build all of the domain specific components from scratch today (i.e., the model and wrappers), it could probably be done in a few days. It would also be useful to compare the time required to build this application without using Ariadne. We have not done that, but the next best thing is to compare this application to the information available from commercial web sites that provide dinner and/or film information on line, such as movielink.com (777-Film), hollywood.com and dinnerandamovie.com. If you look at these other sites, they lack the map based integration, the links to other related data such as restaurant reviews, and even the ability to quickly navigate from the theater to the trailers of the movies showing at a theater. Clearly these could all be done without Ariadne, but would require significantly more effort to program.

The next step with TheaterLoc is to use it as a foundation for constructing MovieLoc, an application which will help film production companies do things like advanced marketing/forecasting analysis and identify filming locations. In the case of the former, we want to be able to allow film marketing teams to locate potential theaters to distribute their movies to, based on area demographics and proximities from other theaters playing that same movie. In the case of location management, we would like to integrate data from various location providers (those who are paid to provide film producers a location containing, say, “a small farmhouse near a swamp”), so that we can support advanced location searches across multiple providers.

Related to strategic movie deployment, we also intend to support spatial queries on the integrated sources. With MovieLoc, for example, one can envision a movie distributor who wants to distribute a movie for young adults to the theaters that are located in the cities with high population of teenagers between the age of 12 and 18. Spatial queries would also be relevant with the existing TheaterLoc application, in order to support marketing queries such as *find all the zip-code areas that contain less than three Chinese restaurants*. There are many sources on the Web that can be utilized to support these sorts of queries, such as the US Census (http://www.census.gov) for obtaining demographic data and the US Postal Service (http://www.usps.gov) for obtaining geographical coordinates of zip-code areas and cities.

6 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 (DARPA) under contract number F30602-98-2-0109, and in part by the United States Air Force under contract number F49620-98-1-0046. 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.

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. Furthermore, we would like to thank others who contributed to the integration effort, including Anupam Bordia and Yi-Jing (Jessie) Chen. Finally, we are also grateful for the help of Chris Stuber, at the USGS Tiger Mapping Service, for his help regarding the translation of coordinates during map generation.

7 References

Ambite, J.L.; Arens, Y.; Ashish, N.; Knoblock, C.A.; Minton, S; Modi, J; Muslea, M.; Philpot, A.; Shen,W.M.; and Tejada, S. 1998. *The SIMS Manual: Version 2.0*

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*, Providence, RI.

Ambite, J.L. and Knoblock, C.A. 1998. Flexible and Scalable Query Planning in Distributed and Heterogeneous Environments. *Proceedings of the Fourth International Conference 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. Information Sciences Institute, University of Southern California.

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*.

Arens, Y.; Knoblock, C.A.; and Shen, W.M. 1996. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems*, 6(2/3):99-130.

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. *Proceedings of First International Conference on Autonomous Agents*.

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. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 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. *Third Conference on Autonomous Agents*, 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.