International Journal of Cooperative Information Systems

THE ARIADNE APPROACH TO WEB-BASED INFORMATION INTEGRATION*

CRAIG A. KNOBLOCK, STEVEN MINTON, JOSE LUIS AMBITE,
NAVEEN ASHISH, ION MUSLEA, ANDREW G. PHILPOT, and SHEILA TEJADA
Information Sciences Institute, Integrated Media Systems Center,
and Department of Computer Science
University of Southern California
4676 Admiralty Way,
Marina del Rey, CA 90292

Received (to be inserted
Revised by Publisher)

The Web is based on a browsing paradigm that makes it difficult to retrieve and integrate data from multiple sites. Today, the only way to do this is to build specialized applications, which are time-consuming to develop and difficult to maintain. We have addressed this problem by creating the technology and tools for rapidly constructing information agents that extract, query, and integrate data from web sources. Our approach is based on a uniform representation that makes it simple and efficient to integrate multiple sources. Instead of building specialized algorithms for handling web sources, we have developed methods for mapping web sources into this uniform representation. This approach builds on work from knowledge representation, databases, machine learning and automated planning. The resulting system, called Ariadne, makes it fast and easy to build new information agents that access existing web sources. Ariadne also makes it easy to maintain these agents and incorporate new sources as they become available.

Keywords: Information integration, information agents, web sources, knowledge representation, machine learning, automated planning, wrappers

1. Introduction

The amount of data accessible via the Web and intranets is staggeringly large and growing rapidly. However, the Web's browsing paradigm does not support many information management tasks. For instance, the only way to integrate data from multiple sites is to build specialized applications by hand. These applications are time-consuming and costly to build, and difficult to maintain.

This paper describes Ariadne,[a] a system for extracting and integrating data from semi-structured web sources. Ariadne enables users to rapidly create *information agents* for the Web. Using Ariadne's modeling tools, an application developer

*This article is an extended version of the article originally published in AAAI'98 [20]
[a]In Greek mythology, Ariadne gave Theseus the thread that let him find his way out of the Minotaur's labyrinth.

starts with a set of web sources – semi-structured HTML pages, which may be located at multiple web sites – and creates a unified view of these sources. Once the modeling process is complete, an end user (who might be the application developer himself) can issue database-like queries as if the information were stored in a single large database. Ariadne's query planner decomposes these queries into a series of simpler queries, each of which can be answered using a single HTML page, and then combines the responses to create an answer to the original query.

The modeling process enables users to integrate information from multiple web sites by providing a clean, well-understood representational foundation. Treating each web page as a relational information source – as if each web page was a little database – gives us a simple, uniform representation that facilitates the data integration. The representation is quite restricted, but we compensate for that by developing intelligent modeling tools that help application developers map complex web sources into this representation.

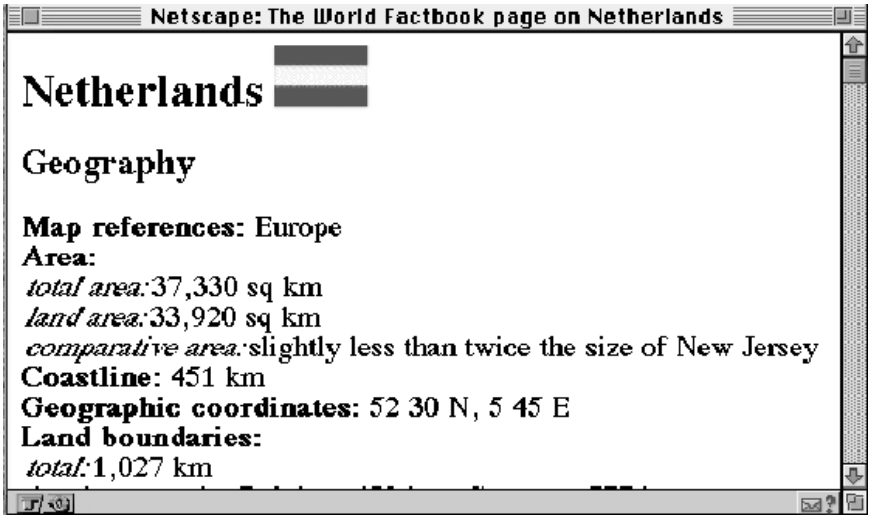

Figure 1: A CIA Factbook Page

We will illustrate Ariadne by considering an example application that involves answering queries about the world's countries. An excellent source of data is the CIA World Factbook, which has an HTML page for each country describing that country's geography, economy, government, etc. The top of the Factbook page for the Netherlands is shown in Figure 1.[b] Some of the many other relevant sites include the NATO site, which lists the NATO member countries (shown in Figure 2), and the World Governments site, which lists the head of state and other government officers for each country (shown in Figure 3). Consider queries such as "What NATO

[b] All the web sources in our examples are based on real sources that Ariadne handles, but we have simplified some of them here for expository purposes.

countries have populations less than 10 million?" and "List the heads of state of all the countries in the Middle East". Since these queries span multiple countries and require combining information from multiple sources, answering them by hand is time consuming. Ariadne allows us to rapidly put together a new application that can answer a wide range of queries by extracting and integrating data from prespecified web sources.

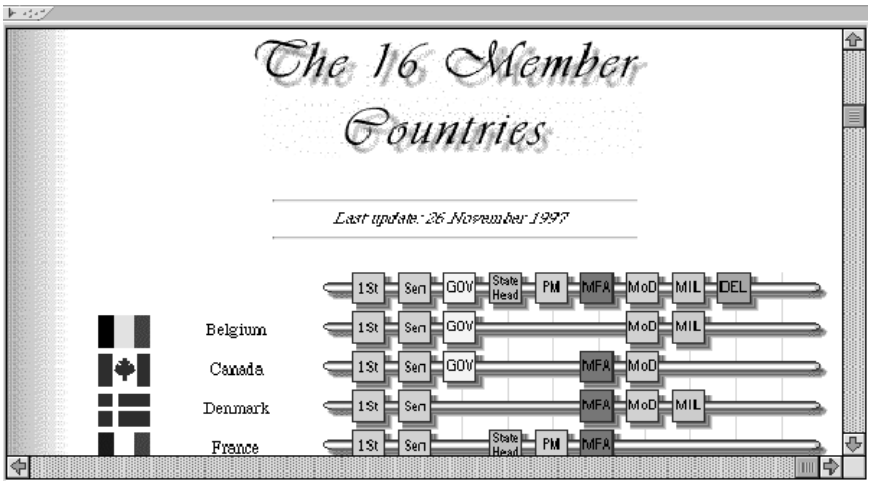

Figure 2: NATO Members Page

In the following section we describe our basic approach to query planning, where a unifying domain model is used to tie together multiple information sources. We then describe the details of our modeling approach: how we represent and query individual web pages, how we represent the relationships among multiple pages in a single site, how we integrate data that spans multiple sites, and how we represent and materialize data locally to optimize an application. In each section, we also describe the methods that are used in modeling and query processing, and how the uniform representational scheme supports these methods.

2. Approach to Information Integration

The Ariadne integration framework consists of the following components:

- A model of the application domain,
- A description of the information sources in terms of this model,
- Wrappers that provide uniform access to the information sources so that they can be queried as if they were relational databases, and
- A query planner that dynamically determines how to efficiently process a user query given the set of available information sources.

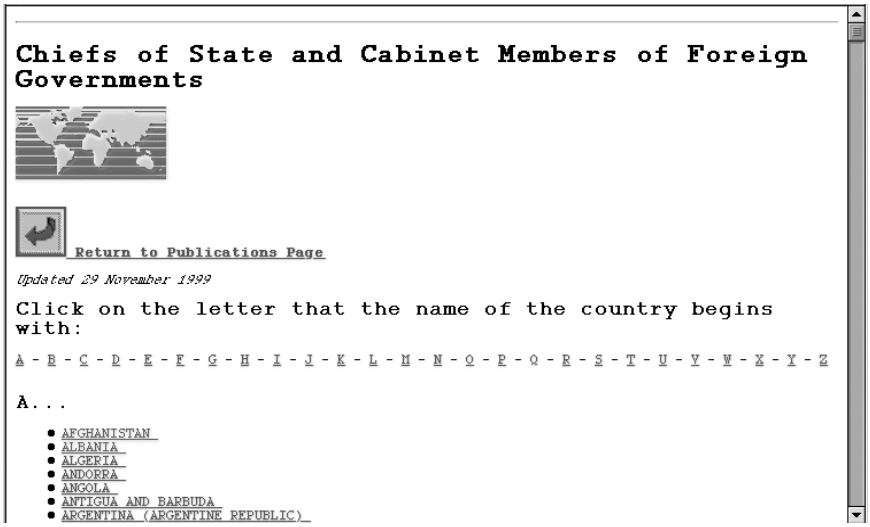

Figure 3: World Governments Page

As we describe in this paper, these components provide the infrastructure to build a complete web-based information integration system. For example, navigating through a web site is simply a matter of creating wrappers for the navigation pages, representing these pages, and letting the query planner generate the appropriate navigation plans. Similarly, resolving naming inconsistencies across sites is addressed by building a new information source that provides the mapping, modeling this information source, and using the query planning to generate plans that use these mappings when needed. Finally, locally storing data to optimize plans can be done simply by creating a new information source with the cached data, modeling this source, and relying on the query planning to use this cached information when needed.

In this section we provide an overview of Ariadne's integration model and how it facilitates efficient query planning. In later sections we will show how this uniform model supports the requirements of a complete information integration system for the web.

Ariadne's approach to information integration is an extension of the SIMS mediator architecture [5, 6, 19]. SIMS was designed for structured information sources such as databases and knowledge bases (and to some extent output from programs). In Ariadne, we extend the SIMS approach to semi-structured sources such as web sources by using wrappers. Also, database applications typically involve only a small number of databases, while web applications can involve accessing many more sources. Since the SIMS planner did not scale well to large numbers of sources, for Ariadne we developed an approach capable of efficiently constructing large query plans by precompiling part of the integration model and using a local search method

for query planning [4].

2.1. Integration Model

In SIMS and Ariadne the mediator designer defines a *domain model*, which is an ontology of the application domain that integrates the information in the sources and provides a single terminology over which the user poses queries. The domain model is represented using the Loom knowledge representation system [24]. Each information source is defined to be equivalent to a class description in the domain model. Thus, Ariadne uses a form of local-as-view source descriptions, cf. [28]. This approach facilitates the addition of new sources to the mediator, since the new source definitions do not interact with the previous ones.

As an example of a domain model consider Figure 4. For simplicity of exposition, this model assumes that the information in the three web sites described earlier, the CIA World Factbook, the World Governments site, and the NATO members page, is available in three separate databases, along with a fourth database containing a map for each country (later we show the modeling when the information is available from web sources, see Figure 11). The model contains four classes with some relations between them. For example, 'NATO Country' is a subclass of 'Country', and 'Country' has a relation called 'Head-of-State' which points to a class with the same name. Each domain class has a set of attributes. For example, some of the attributes of the 'Country' class are total area, latitude, population, etc. We use the domain model to describe the contents of each information source. For example, the figure shows that the CIA Factbook is a source for information about Countries, and the World Governments database is a source for Heads of State. A source may provide only a subset of the attributes for a given domain class, so the system may need to combine several sources to obtain all the desired attributes. For example, if the user requests the total area and a map of a country, the system must retrieve the total area from the CIA World Fact Book and the map from the Map database.

2.2. Query Processing

Queries are presented to the system in terms of the domain model. For example, a query might be "List the heads of state of all the countries whose population is less than ten million."[c] The system then decomposes the query into subqueries on the individual sources, such as the World Governments and Factbook sources, producing a query plan consisting of relational operators (i.e., joins, selects, projects, etc.) and access operations to the sources.

To improve the efficiency of query planning, we used two techniques. First, the source descriptions are compiled off-line into a more convenient form. Second, we implemented a transformational query planner that explores the space of query evaluation plans using local search.

[c]We use English translations of the queries for clarity. In the system the queries can be expressed using either SQL or the Loom query language.

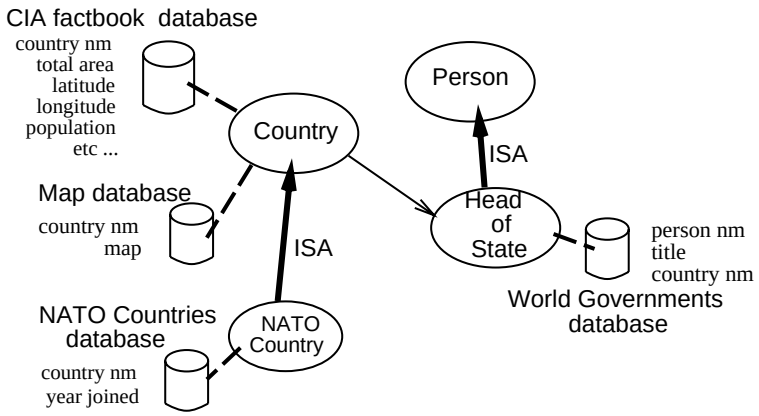

Figure 4: Domain Model with Database Sources

2.2.1. Integration Axiom Precompilation

To facilitate query planning, we developed an algorithm [5] to compile the local-as-view source descriptions into global-as-view integration axioms. Each integration axiom specifies an alternative combination of sources that produces a maximal set of attributes for a domain class. Once the integration axioms are compiled, finding the sources relevant for a user domain query is straightforward. For each domain class in the query the system looks up the integration axioms for that class and its attributes mentioned in the query. The body of each integration axiom is a formula containing only source terms that can be substituted by the corresponding domain class in the query. Compiling axioms *a priori* is a space-for-time optimization, allowing the system to amortize the cost of producing them over all queries, thus avoiding repetition of costly run-time search. This section presents the highlights of the approach, see [5] for details.

Ariadne generates the integration axioms from the source descriptions and the structure of the domain model. The axiom compilation algorithm is based on the iterative application of a set of inference rules. The rules are applied in parallel, constructing a generation of novel integration axioms from existing axioms. The process is repeated until quiescence. The five rules are:

- *Direct Rule*: Inverts the source descriptions.
- *Covering Rule*: Exploits the covering relationships in the domain model (when a class is defined as the union of its subclasses, the subclasses constitute a covering of the class).
- *Definition Rule*: Exploits the constraints in the definition of a domain class.
- *Inherit Rule*: Exploits the inheritance of superclass attributes via shared keys.

- *Compose Rule*: Combines axioms on a given class to provide additional attributes.

The compiled axioms from the domain of Figure 4 are shown in Figure 5. In the first generation the source descriptions are inverted and installed as axioms via the *direct* rule (the axioms *1.1*, *1.2*, *1.3*, and *1.4*). Then, the sources are composed iteratively to produce more attributes for each domain class. In the second generation, the *inherit rule* produces axioms *2.1* and *2.2* by adding the NATO-db source to the body of axioms *1.1* and *1.2* which specilizes them for NATO countries and provides the additional 'map' attribute. Also, the *compose* rule generates axiom *2.3* by joining the CIA Factbook database and the Map database over the common key attribute 'country-nm'. Finally, in the third generation, the inherit rule constructs axiom *3.1* by adding the NATO-db source to axiom *2.3* in order to produce the maximal number of attributes that the available sources provide for the 'NATO Country' class. In this domain, rule application takes three generations until quiescence producing the 8 axioms of Figure 5.

Country(cn ta lat long pop)	$\Leftrightarrow$	CIA-db(cn ta lat long pop)	*1.1*
Country(cn map)	$\Leftrightarrow$	Map-db(cn map)	*1.2*
Country(cn ta lat long pop map)	$\Leftrightarrow$	CIA-db(cn ta lat long pop) $\wedge$ Map-db(cn map)	*2.3*
NATO-Country(cn year)	$\Leftrightarrow$	NATO-db(cn year)	*1.3*
NATO-Country(cn ta lat long pop year)	$\Leftrightarrow$	CIA-db(cn ta lat long pop) $\wedge$ NATO-db(cn year)	*2.1*
NATO-Country(cn map year)	$\Leftrightarrow$	Map-db(cn map) $\wedge$ NATO-db(cn year)	*2.2*
NATO-Country(cn ta lat long pop map year)	$\Leftrightarrow$	CIA-db(cn ta lat long pop) $\wedge$ Map-db(cn map) $\wedge$ NATO-db(cn year)	*3.1*
Head-of-State(pn cn title)	$\Leftrightarrow$	WorldGov-db(pn cn title)	*1.4*

Figure 5: Compiled Integration Axioms

2.2.2. Query Planning

Once the integration axioms have been compiled, Ariadne is ready to accept user queries expressed over terms of the domain model. Query planning for each user query follows two steps. First, the query is parsed, simplified, and rewritten so that each class mentioned in the query is the most specific according to the definitions in the domain model. This ensures that the appropriate integration axioms are used during query planning. For example, consider an extension to Figure 4 which includes the class 'Military Leader', defined as the subclass of 'Head of State' where the title is equal to "general", "colonel", etc. An input query that asks for information about a 'Head of State' whose title is "general" will find all the relevant axioms associated with the class 'Military Leader' as opposed to the class

'Head of State' that is mentioned in the query.

Second, the query is optimized using a transformational query planner [4, 3] based on the Planning by Rewriting paradigm [2, 1]. The query planner first constructs an initial query evaluation plan based on a depth-first parse of the query. This initial plan is possibly suboptimal, but it is generated very efficiently. Then, the plan is iteratively transformed using a set of rewriting rules in order to optimize the plan cost. The rewriting rules are derived from properties of the relational algebra, the distributed environment, and the integration axioms in the application domain. The space of plan rewritings is explored efficiently using local search methods. During the rewriting process, the planner considers the different sources, operators, and orders of the operators that can be used to answer the query.

The query planner has a modular, declarative, and extensible architecture. The initial plan generator, the cost metric, the set of rewriting rules, and the strategy used during the rewriting search, are all modules that can be extended or replaced independently. Since our query planner is based on a domain-independent approach to planning, it is extensible in a principled way and very flexible. The specification of both the plan operators and the plan rewriting rules is declarative.

As an example, consider the processing required to retrieve the names of all NATO countries and the year they joined NATO for those countries that have a population of less than 10 million. In this case, the relevant axiom that provides the population of NATO countries and the year of incorporation (using a minimal set of sources) is axiom *2.1* in Figure 5 (if there were several alternative axioms for that information, the planner would consider them during the search). Based on this axiom, assume that the planner constructs the initial plan of Figure 6. This plan is suboptimal since it retrieves the names and population for all countries from the Factbook source, the names and year from the NATO source, and then joins both relations locally, which is very costly since the Factbook source is quite large. Moreover, the selection on population is done after the join, instead of being used to reduce the number of tuples that participate in the join. The query planner rewrites this initial plan producing the optimized plan of Figure 7. The optimized plan first retrieves the name and year of the NATO countries, projects the country names, and passes NATO country names so that only the population of NATO countries is retrieved from the CIA World Factbook source. In addition the selection on population has been placed immediately after the retrieval from the Factbook source to further reduce the number of tuples participating in the join.

Ariadne's uniform integration model is expressive enough to encompass a wide variety of web sources but simple enough to allow for efficent query planning. In the following sections, we discuss how, based on this simple model, we provide a coherent set of solutions for each level of the problem of integrating information on the web. First, we discuss how we model and automatically learn wrappers for individual web pages. Second, we describe the modeling that allows our query planner to generate plans that navigate among pages. Third, we present techniques to identify entities across sites so that they can be integrated. Fourth, we describe further

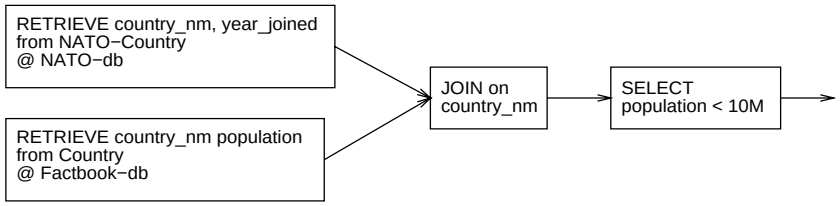

Figure 6: A Suboptimal Initial Query Plan

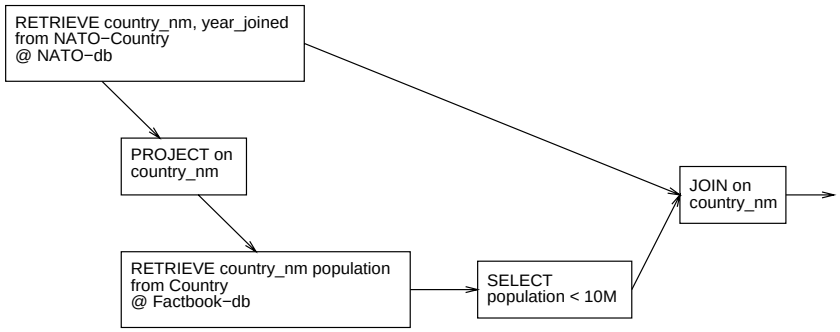

Figure 7: An Optimized Query Evaluation Plan

performance optimization by selectively materializing data. Finally, we describe some of the applications that have been developed using Ariadne and assess the approach.

3. Modeling the Information on a Page

The previous section described how the planner decomposes a complex query into simple queries on individual information sources. To treat a web page as an information source so that it can be queried, Ariadne needs a wrapper that can extract and return the requested information from that type of page. While we cannot currently create such wrappers for unrestricted natural language texts, many information sources on the Web are *semistructured*. A web page is semistructured if information on the page can be located using a concise formal grammar, such as a context-free grammar. Given such a grammar, the information can be extracted from the source without recourse to sophisticated natural language understanding techniques. For example, a wrapper for pages in the CIA Factbook would be able to extract fields such as the Total Area, Population, etc. based on a simple grammar describing the structure of Factbook pages.

Our goal is to enable application developers to easily create their own wrappers for web-based information sources. To construct a wrapper, we need both a semantic model of the source that describes the fields available on that type of page and a syntactic model, or grammar, that describes the page format, so the fields can be

extracted. Requiring developers to describe the syntactic structure of a web page by writing a grammar by hand is too demanding, since we want to make it easy for relatively unsophisticated users to develop applications. Instead, Ariadne has a "demonstration-oriented user interface" (DoUI) where users show the system what information to extract from example pages. Underlying the interface is a machine learning system for inducing grammar rules.

Figure 8 shows how an application developer uses the interface to teach the system about CIA Factbook pages, producing both a semantic model and a syntactic model of the source. The screen is divided into two parts. The upper half shows an example document, in this case the Netherlands page. The lower half shows a semantic model, which the user is in the midst of constructing for this page. The semantic model in the figure indicates that the class Country has attributes such as Total Area, Coastline, Latitude, Longitude, etc. The user constructs the semantic model incrementally, by typing in each attribute name and then filling in the appropriate value by cutting and pasting the information from the document. In doing so, the user actually accomplishes two functions. First, he provides a name for each attribute. Notice that he can choose the same names as used in the document (e.g., "Total area") or he can choose new/different names (e.g., "Latitude"). As we will explain later, the attribute names have significance, since they are the basis for integrating data across sources.

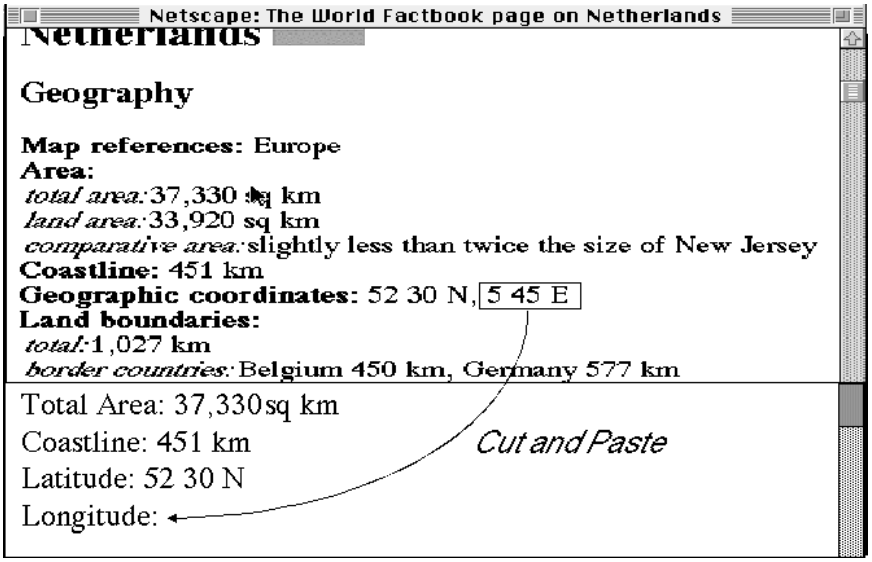

Figure 8: Creating a Wrapper by Demonstration

The second function achieved by the user's demonstration is to provide examples so that the system can induce the syntactic structure of the page. Ideally, after the user has picked out a few examples for each field, the system will induce a

grammar sufficient for extracting the required information for all pages of this type. Unfortunately, grammar induction methods may require many examples, depending on the class of grammars being learned. However, we have observed that web pages have common characteristics that we can take advantage of, so that a class of grammars sufficient for extraction purposes can be rapidly learned in practice.

More specifically, we can describe most semistructured web pages as *embedded catalogs*. A *catalog* is either a homogeneous list, such as a list of numbers, (1,3,5,7,8), or a heterogeneous tuple, such as a 3-tuple consisting of a number, a letter, and a string, (1,A,"test"). An *embedded catalog* (or, for short, $\mathcal{EC}$) is a catalog where the items themselves can be catalogs. As an example, consider the fragment of the CIA Factbook page shown in Figure 8. At the top level, it can be seen as a 9-tuple that consists of Map References, Total Area, Land Area, Comparative Area, Coastline, Latitude, Longitude, Total Land Boundaries, and Border Countries. Furthermore, the Border Countries represent an embedded list of 2-tuples that contain a Country Name and a Border Length. Note that the illustrative page description above contains *all* the fields in the document, while in practice, the user can identify only the items of interest for a particular application. For instance, a user can model the CIA Factbook page as a 4-tuple that consists of the Total Area, Latitude, Longitude, and Border Countries, where the last field is an embedded list of 2-tuples that contain a Country Name and a Border Length.

An embedded catalog is a structured description of the information on a page, and can be (trivially) converted into an XML view of the document, as illustrated in Figure 9.

```
<!DOCTYPE document [
 <!ELEMENT document ( TotalArea, Latitude, Longitude,
                      Neighbors_LIST )>
 <!ELEMENT TotalArea (#PCDATA)>
 <!ELEMENT Latitude (#PCDATA)>
 <!ELEMENT Longitude (#PCDATA)>
 <!ELEMENT Neighbors_LIST ( Neighbor* )>
 <!ELEMENT Neighbor (Name, BorderLength)>
 <!ELEMENT Name (#PCDATA)>
 <!ELEMENT BorderLength (#PCDATA)> ] >
```

Figure 9: **Sample XML description of an embedded catalog.**

Besides being used as *data schema* in the integration process, the $\mathcal{EC}$ description of a document plays another important role: in Ariadne, we use a document's $\mathcal{EC}$ to extract the data in a hierarchical manner. For instance, in order to extract all the neighboring countries from a document, we begin by extracting the Border Countries from the whole document; then we iterate through this list, and break it down to individual 2-tuples; finally, from such tuples, we extract each individual Country Name. In other words, if we look at $\mathcal{EC}$ as a tree-like structure describing the embedded data, in order to extract a relevant item we must successively extract

each of its ancestors from their respective parents in the tree. Our approach has a major advantages: it transforms a potentially hard problem (i.e., extracting *all* items from an arbitrarily complex document) into a set of simpler ones (i.e., extracting *one* individual item from its parent in the $\mathcal{EC}$). This is particularly appealing when a document contains a large number of items and multiple levels of embedded data (e.g., list within lists).

Because web pages are intended to be human readable, special markers often play a role identifying the beginning or ending of an item in an embedded catalog, separating items in a homogeneous list, and so on. These distinguishing markers can be used as landmarks for locating information on a page. A *landmark grammar* describes the position of a field via a sequence of landmarks, where each landmark is a sequence of tokens and wildcards (e.g., $Number$, $CapitalizedWord$, $AllCaps$, etc.). For example, to find the beginning of the longitude, we can use the rule

R1 = $SkipTo($`Geographic coordinates`$)$ $SkipTo($`,`$)$

which has the following meaning: start from the beginning of the document and skip everything until you find the landmark `Geographic coordinates`; then, again, ignore all tokens until you encounter the first comma. Similarly, we can use the rule

R2 = $SkipTo($ `<br> Land boundaries`$)$

to identify the end of the longitude field (`<br>` is the HTML tag that forces the line break, and, consequently, it is not displayed by the browser in Figure 8).

In order to fully define a wrapper, one needs to provide the embedded catalog, together with one extraction rule for each field and one additional iteration rule for each list in the $\mathcal{EC}$ (iteration rules are applied *repeatedly* to the content of the list in order to extract all individual tuples). Our recent work [26] shows that in practice, a subclass of landmark grammars (i.e., linear landmark grammars) can be learned rapidly for a variety of web pages using a greedy covering algorithm. There are several reasons for this. First, because web pages are intended to be human readable, there is often a *single* landmark that distinguishes or separates each field from its neighbors. Therefore, the length of the grammar rules to be learned will usually be very small, and learning will be easy in practice. Second, during the demonstration process, users traverse a page from top-to-bottom, picking out the positive examples of each field. Any position on the page that is not marked as a positive example is implicitly a negative example. Thus, for every positive example identified by the user, we obtain a huge number of negative examples that the covering algorithm can use to focus its search.

The empirical evaluation of STALKER [26], our wrapper induction system, shows that in most of the cases our system learns perfect extraction rules (i.e., 100% accuracy) based on just a handful of examples. We tested STALKER on 30 information sources, which required the induction of 206 different rules. In 182 cases STALKER generated a perfect rule (in most of the cases based on just a couple of labeled examples), and 18 other rules had an accuracy above 90% based on as few as 10 training examples. On the same 30 information sources, WIEN [21], which was

the first wrapper induction system, requires one to two orders of magnitude more labeled examples in order obtain a similar or worse performance.

There are several differences between the approaches taken by STALKER and WIEN. First of all, WIEN's approach to handling documents with multiple levels of embedded data turned out to be impractical: even for the domains in which such a wrapper exists, the learning algorithm failed to find it. Second, WIEN uses a very simple extraction language. By assuming that all the fields are *always* present and in exactly the same order, WIEN is capable of learning the rules extremely fast, provided that they exist. On the other hand, these assumptions make it impossible to wrap more complicated sources. Last but not least, the same simplicity of the extraction language, together with the fact that WIEN does not extract the sibling fields independently of each other, leads to failure to wrap sources from which STALKER finds perfect rules for most of the items, and slightly imperfect ones for the remaining fields.

A quite different approach to wrapper induction is the one used in SoftMealy [16]. This system induces extraction rules expressed as finite transducers, and it addresses most of the WIEN's shortcomings. However, its empirical evaluation is quite sketchy, which makes it hard to compare with WIEN and STALKER. There are three other recent systems that are focusing on learning extraction rules from online documents: SRV [13], RAPIER [11], and WHISK [27]. Even though these approaches are mostly concerned with extracting data from natural language text, they could be also applied to some simple wrapper induction problems.

The modeling tool we have described enables unsophisticated users to turn web pages into relational information sources. But it has a second advantage as well. If the format of a web source changes in minor respects, the system could induce a new grammar by reusing examples from the original learning episode, without any human intervention (assuming the underlying content has not changed significantly). This is a capability we are currently exploring.

4. Modeling the Information in a Site: Connections between Pages

The previous section showed how Ariadne extracts information from a web page to answer a query. However, before extracting information from a page, Ariadne must first locate the page in question. Our approach, described in this section, is to model the information required to "navigate" through a web site, so that the planner can automatically determine how to locate a page.

For example, consider a query to our example information agent asking for the population of the Netherlands. To extract the population from the Factbook's page on the Netherlands, the system must first find the URL for that page. A person faced with the same task would look at the index page for the Factbook, shown in Figure 10, which lists each country by name together with a hypertext link to the page in question. In our approach, Ariadne does essentially the same thing. The index page serves as an information source that provides a URL for each country

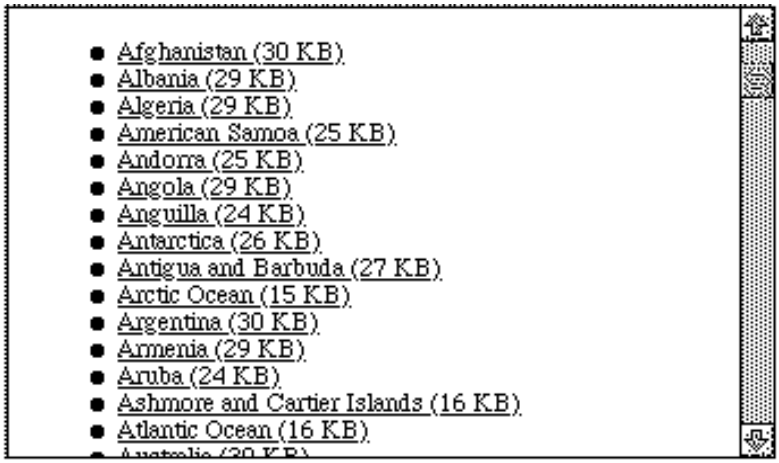

Figure 10: CIA Factbook Index

page. These pages in turn serve as a source for country-specific information.

To create a wrapper for the index page, the developer uses the approach described in the last section, where we illustrated how a wrapper for the Factbook's country pages is created. There is only one difference: this wrapper only wraps a single page, the index page. The developer creates a semantic model indicating that the index page contains a list of countries, each with two attributes, country-nm and country-URL.[d] The learning system induces a grammar for the entire page after the developer shows how the first few lines in the file should be parsed.

As the wrappers for each source are developed, they are integrated into the unifying domain model. Figure 11 shows the domain model for the completed geopolitical agent. (Notice that we have substituted web source wrappers for the hypothetical databases used previously.) To create the domain model, the developer specifies the relationship between the wrappers and the domain concepts. For instance, the developer specifies that the Factbook country wrapper and the Factbook index wrapper are both information sources for "country" information, and he identifies which attributes are keys (i.e., unique identifiers). In the example, "country-nm" and "country-URL" are both keys. Binding constraints specify the input and output of each wrapper (shown by the small directional arrows in Figure 11). The country page wrapper takes a country-URL, and acts as a source for "total area", "population", "latitude", etc. The index wrapper takes a country name[e] and acts as a source for "country-URL". Given the domain model and the binding constraints, the system can now construct query plans. For instance, to

[d] During the demonstration, a check box is used to extract a URL from a hyperlink, as opposed to grabbing text.

[e] No URL is needed as input to the index page wrapper since the URL of the index page is a constant.

obtain the population of a country given its name, the planner determines that the system must first use the country name to retrieve the country-URL from the index page wrapper, and then use the country-URL to retrieve the population data from the country page wrapper.

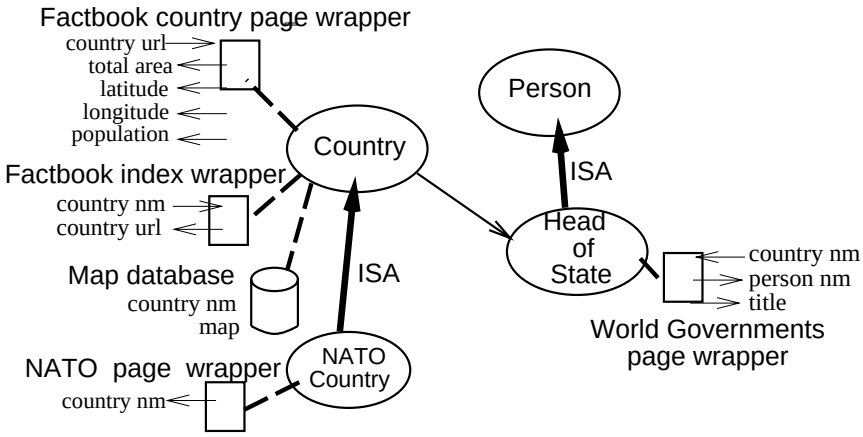

Figure 11: Domain Model with Web Sources

Explicitly modeling 'navigation' pages, such as the Factbook index, as information sources enables us to reuse the same modeling tools and planning methodology underlying the rest of the system. The approach works well in part because there are only two common types of navigation strategies used on the Web – direct indexing and form-based retrieval. We have already seen how index pages are handled; form-based navigation is also straightforward. A wrapper for an HTML form simply mimics the action of the form, taking as input a set of attributes, each associated with a form parameter name, and communicating with the server specified in the form's HTML source.

When the resulting page is returned, the wrapper extracts the relevant attributes in the resulting page. Imagine, for instance, a form-based front end to the Factbook, where the user types in a country name and the form returns the requested country page. To create a wrapper for this front end, the developer would first specify that the parameter associated with the type-in box would be filled by a "country-nm". He would then specify how the system should extract information from the page returned by the form using the approach described in the last section.

The Factbook example described in this section illustrates our basic approach to modeling navigation pages. Many web sites are more complex than the Factbook. The approach still works, but the models become more involved. For instance, indexes can be hierarchical, in which case each level of the hierarchy must be modeled as an information source. Imagine the top-level Factbook index was a list of letters, so that clicking on a letter "C" would produce an index page for countries start-

ing with "C" (a "subindex"). We would model this top level index as a relation between letters and subindex-URL's. To traverse this index, we also need an information source that takes a country name and returns the first letter of the name (e.g., a string manipulation program). Thus, altogether four wrappers would be involved in the navigation process, as shown in Figure 12. Given a query asking for the Netherlands' population, the first wrapper would take the name "Netherlands", call the string manipulation program, and return the first letter of the name, "N". The second wrapper would take the letter "N", access the top level index page, and return the subindex-URL. The third wrapper would take the subindex-URL and the country name, access the subindex page for countries starting with "N", and return the country-URL. Finally, the last wrapper would take the country-URL and access the Netherlands page. The advantage of our approach is that all these wrappers are treated uniformly as information sources, so the query planner can automatically determine how to compose the query plan. Furthermore, the wrappers can be semi-automatically created via the learning approach described earlier, except for the string manipulation wrapper, which is a common utility.

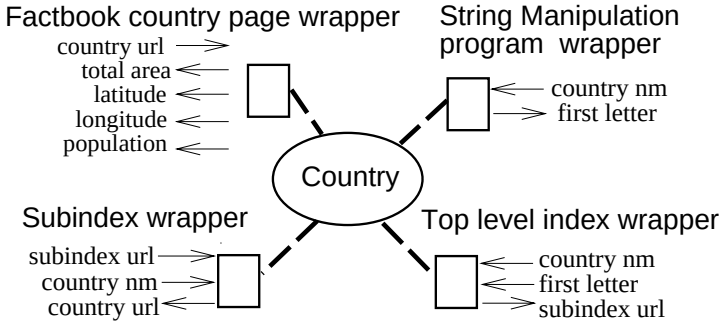

Figure 12: Domain Model with Hierarchical Index

5. Modeling Information Across Sites

Within a single site, entities (e.g., people, places, countries, companies, etc.) are usually named in a consistent fashion. However, across sites, the same entities may be referred to with different names. For example, the CIA Factbook refers to "Burma" while the World Governments site refers to "Myanmar". Sometimes formatting conventions are responsible for differences, such as "Denmark" vs. "Denmark, Kingdom of". To make sense of data that spans multiple sites, we need to be able to recognize and resolve these differences.

Our approach is to select a primary source for an entity's name and then provide a mapping from that source to each of the other sources where a different naming scheme is used. An advantage of the Ariadne architecture is that the mapping itself can be represented as simply another wrapped information source. One way to do

this is to create a *mapping table*, which specifies for each entry in one data source what the equivalent entity is called in another data source. Alternatively, if the mapping is computable, it can be represented by a *mapping function*, which is a program that converts one form into another form.

Figure 13 illustrates the role of mapping tables in our geopolitical information agent. The Factbook is the primary source for a country's name. A mapping table maps each Factbook country name into the name used in the World Governments source (i.e., WG-country-nm). The mapping source contains only two attributes, the (Factbook) country name and the WG-country-nm. The NATO source is treated similarly. So, for example, if someone wanted to find the Heads of State of the NATO countries, the query planner would retrieve the NATO country names from the NATO wrapper, map them into (Factbook) country names using the NATO mapping table, then into the World Government country names using the World Governments mapping table, and finally retrieve the appropriate heads of state from the World Governments wrapper.

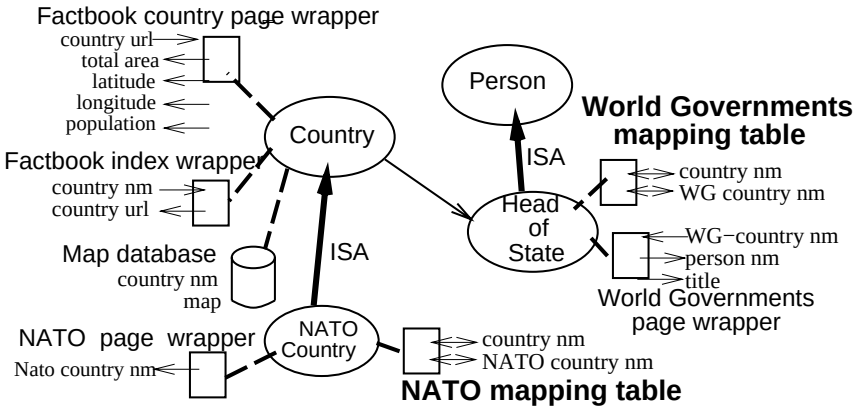

Figure 13: Domain Model with Mapping Tables

We have developed a semi-automated method for building mapping tables and functions by analyzing the underlying data in advance. The method attempts to pair each entity in one source with a corresponding entity (or entities, in some cases) in another source. The basic idea is to use information retrieval techniques to provide an initial mapping (following [12]), and then to apply machine learning techniques to improve the mapping. The initial mapping matches entities from two sources based on their textual similarity. For example, "Denmark" and "Denmark, Kingdom of" are assumed to refer to the same entity because both names include "Denmark", an infrequently-occurring word. In the subsequent learning phase, the system learns two types of rules to help improve/verify the initial mapping. *Transformation rules* identify textual transformations like acronyms, abbreviations, and phrase orderings that are common in the domain. For instance, the system can learn that "Rep" is a

commonly used abbreviation for "Republic", or that one source commonly employs acronyms, or that one source represents person names as "LastName, Firstname", while the other uses "FirstName LastName". The system also learns *Mapping rules* which are used when we can compare entities along multiple attributes.

For example, consider a multi-year Factbook application which includes yearly versions of the Factbook (each year a new version of the CIA Factbook is released), as shown in Figure 14. Sometimes countries in the new Factbook have new names, or countries merge or split. These name confusions can often be resolved by using the attributes containing geographical information, e.g., land area, latitude and longitude. These attributes stay constant over time and are unique to a country, so they are good indicators of a mapping. Mapping rules specify the importance of each attribute when computing a mapping.

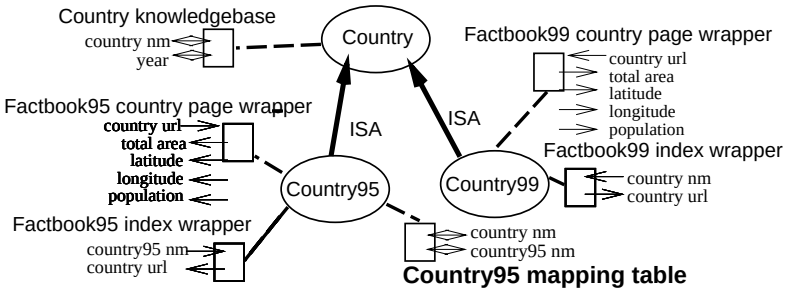

Figure 14: Multi-Year Factbook Model

We are prototyping an active learning method for learning transformation rules and mapping rules. In our approach, a human is asked to verify some of the pairs in the initial mapping, i.e., to indicate whether the pairs are correctly or incorrectly matched. Then the system attempts to learn new rules, and selects additional pairs for the human to verify. The system selects the pairs that will be most valuable for confirming/disconfirming the hypotheses explored by the learning algorithm. The goal is to obtain a mapping for which the system is highly confident, while minimizing the time the human must spend.

6. Performance Optimization by Selective Materialization

An issue with building applications using Ariadne is that the speed of the resulting application is heavily dependent on the individual web sources. The response time may be high even when the query planner generates high-quality information integration plans. This is because for some queries a large number of pages may have to be fetched from remote Web sources and some sources may be very slow. In this section we describe our approach to optimizing agent performance by locally materializing selected portions of the data. The materialized data is simply a locally stored relation that is handled using the same uniform representation and

query planning techniques used throughout the system.

The brute force approach would be to simply materialize all the data in all the Web sources being integrated. However this is impractical for several reasons. First the sheer amount of space needed to store all this data locally could be very large. Second, the data might get updated at the original sources and the maintenance cost for keeping all the materialized data consistent could be very high. Our approach is thus to materialize data *selectively*. In this approach to optimizing performance by selectively materializing data there are two key issues that must be addressed.

1. What is the overall framework for materializing data, i.e., how do we represent and use the materialized data?

2. How do we automatically identify the portion of data that is most useful to materialize?

In [7] we presented a framework for representing and using materialized data in an information agent. The basic idea is to locally materialize useful data and define it as another information source for the agent. The agent then considers using the materialized data instead of retrieving data from remote sources to answer user queries. For instance, in the countries application suppose we determined that the population and national product of all European countries was queried frequently and thus useful to materialize. The system would retrieve this data and define it as an additional information source as shown in Figure 15. Given a query the planner would use the materialized data instead of the original Web source(s) to answer the query when it reduced the cost of a query. The system selects the data to materialize by considering a combination of several factors which are described below.

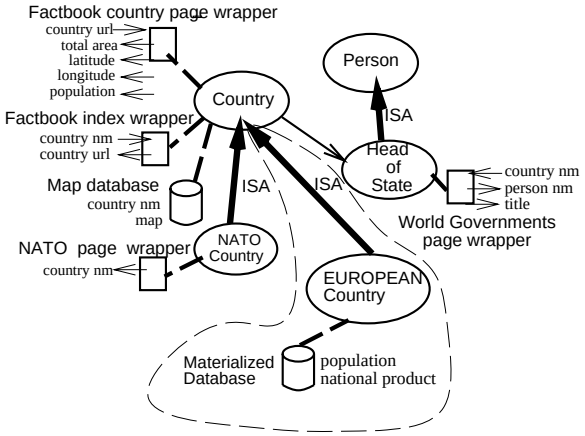

Figure 15: Materialized Data as Another Source

6.1. The Distribution of User Queries

From the user query distribution we determine what classes of data are queried most frequently by users since it is often useful to materialize such classes. We have developed an algorithm known as the CM (Cluster and Merge) algorithm [8], which identifies useful classes of information to materialize by extracting patterns in user queries. A key feature of this algorithm is that it finds *compact* patterns to be extracted. Compact patterns are important from a query planning perspective since we define a new information source in the agent for each class of data materialized. A set of fragmented patterns will result in a large number of new sources. The problem of query planning in an information agent is combinatorially hard and having a very large number of sources could create performance problems for the planner. With compact patterns we need to materialize fewer classes of data and thus define fewer new information sources. For instance in the countries application suppose the class of information *"the population and ethnic divisions of all European countries"* was a frequently queried class of data, the CM algorithm would extract this class as a pattern from the query distribution. Even if some European countries were never queried, it might still materialize all of the data for the European countries in order to minimize the number of patterns stored.

6.2. The Structure of Queries and Sources

As described earlier, we provide database-like query access to otherwise only browsable Web sources by building wrappers around the sources. As a result certain kinds of queries can be very expensive as a lot of the functionality for structured querying of these sources needs to be provided by the wrapper or the mediator. For instance in the countries application, the CIA Factbook source is structured such that selection queries, such as say determining which countries are *European* countries requires the wrapper to retrieve pages of all countries in the Factbook source, which takes a long time.

In many kinds of queries we can materialize a portion of data which if stored locally greatly improves the response time of the expensive queries. In the above example if we materialize locally the names and continents of all countries, determining which countries are *European* countries (the step that requires otherwise fetching pages of all countries from the source) can be done locally and much faster. We have generalized this strategy for various kinds of queries that may be expensive such as selection queries, joins, ordered joins etc. The system analyzes the user query interface to determine what queries can be asked in an application and uses the mediator query cost estimator and also specification of the structure of Web sources to determine which of these queries can be expensive. It then prefetches and materializes data based on heuristics that use information about the kind of query (selection, join etc.) and the structure of the source (whether it supports selections etc.) to speed up the processing of the expensive queries.

6.3. Updates at Web Sources

Finally we must address the issue of updates at Web sources. First the materialized data must be kept consistent with that in the Web sources. It may also be that the user is willing to accept data that is not the most recent in exchange for a fast response to his query (using data that is materialized). Thus we need to determine the frequency with which each class of data materialized needs to be refreshed from the original sources. Next the total maintenance cost for all the classes of data materialized must be kept within a limit that can be handled by the system. We provide a language for describing the update characteristics of each source and also user requirements for freshness. The system then takes update frequency and maintenance cost into account when selecting data to materialize. For instance, considering the countries application again, the Factbook is a source that does not change so we materialize whatever data we want to prefetch or is frequently queried from that source as there is no maintenance cost for the materialized data. However we may not want to materialize data from the World Governments source as that is a source where data can change anytime.

We have implemented a materialization system for Ariadne based on the above ideas. We present experimental results in Table 1 demonstrating the effectiveness of our approach. Specifically we show the improvement in performance obtained by materializing data locally in Ariadne using our system.

Query Set	Response Time (No opt.)	Response Time (Ariadne)	Response Time (Page Level)	Improvement (Ariadne)	Improvement (Page Level)
Q1	41233 sec	1515 sec	36974 sec	96%	10%
Q2	47935 sec	2198 sec	42502 sec	95%	13%

Table 1: Experimental Results

Table 1 shows the experimental results for the countries application. We provide results showing the improvement in performance due to our system and compare it with an alternative scheme of page level caching with the same local space. We measure query response times with and without materialization for two query sets - Q1, which is a set of queries we generated and in which we introduced some distinct query patterns, and Q2, which is a set of actual user queries that we logged from a version of the countries application that was made available online for several months. There is significant performance improvement due to our materialization system. Our system also significantly outperforms the page level caching scheme for both query sets Q1 and Q2. The primary reason is that with our system we are much more flexible in selecting the portion of data that we want to materialize in terms of the attributes and tuples we want to specify. In the page level caching scheme, we can only materialize either all or none of an entire web page which causes local space to be wasted for storing data that may not be frequently queried. Also

in this particular application source structure analysis proves to be very effective as the system locally materializes attributes of countries to speed the processing of certain very expensive kinds of selection queries.

7. Applications

We have used Ariadne to build a variety of applications. We list a few of them below to illustrate the generality of our approach.

As shown in the examples in this paper, we built a country information agent with Ariadne that combines data about the countries in the world from a variety of information sources. The agent extracts and integrates data from from the CIA World Factbook. The Factbook is published each year, so the agent can support queries that span multiple years. The agent also extracts data from other related sources such as the World Governments site, which provides up-to-date information about world leaders. This application provides a good example where significant added value was added to the data sources by providing the additional structure where only the relevant data is extracted and returned for specific countries.

We have built a system using Ariadne that integrates data about restaurants and movie theaters and places the information on a map [9]. The application extracts restaurant names and addresses from CuisineNet, theater names and addresses from Yahoo Movies, movies and showtimes for each theater also come from Yahoo Movies, and trailers for movies from Hollywood.com. The addresses are run through the Etak geocoder to convert street addresses into the corresponding latitude and longitude. Finally, Ariadne uses the US Census Bureau's Tiger Map Server to produce a map with all of the restaurants and theaters for a requested city. Clicking on one of the points of the map will bring up the restaurant review or list of movies as appropriate. If the user selects a specific movie, Ariadne will then play the appropriate trailer. This application provides a compelling integration example that combines a variety of multimedia data sources.

We have applied Ariadne to integrate online electronic catalogs to provide a more comprehensive virtual catalog. In this application, Ariadne combines data on pricing, availability, manufacturer, and so on from four major electronic part suppliers. Each site requires several wrappers in order to navigate to the page that provides the required data. In several cases, the data on a given part is spread over multiple pages at a site. This application provides an example of a virtual data source where up-to-date information on electronic parts is made available from multiple suppliers without maintaining any local data.

8. Discussion

There is a large body of literature on information integration [30], and more recently, several projects focusing specifically on information integration on the Web, such as Tukwila [29], Araneus [25], the Information Manifold [23], Occam [22], Infomaster [15], and InfoSleuth [17]. These systems focus on a variety of issues, including the

problems of representing and selecting a relevant set of sources to answer a query, handling binding patterns, and resolving discrepancies among sources. All of this work is directly relevant to Ariadne, but no other system handles the broad range of practical issues that arise in modeling information within a single page, across pages at a site, and across sites to support web-based information integration.

We believe that Ariadne is successful, in terms of the broad applicability of the approach, because it combines a simple representation scheme with sophisticated modeling tools that map web information sources into this simple representation. There are many examples of impressive AI systems based on relatively simple representational schemes. In the realm of planning, recent examples include SATplan [18] and Graphplan [10]; the former employs a propositional CSP approach, the latter, a graph-based search. In machine learning, propositional learning schemes (e.g., decision trees) have been dominant. Though it is often difficult to understand exactly what a simple representational scheme buys you computationally, one thing seems clear: systems with simple representations are often easier to design and understand.

Ariadne's representation scheme was adopted from database systems, where the world consists of a set of relations (or tables) over objects, and simple relational operators (retrieve, join, etc.) are composed to answer queries. This representation makes it straightforward to integrate multiple databases using an AI planner. Ariadne's planner can efficiently search for a sequence of joins, selections, etc. that will produce the desired result without needing to do any sophisticated reasoning about the information sources themselves.

The Web environment is much more than a set of relational tables, of course. What makes Ariadne possible are the modeling tools that enable a user to create a database-like view of the Web. Other competing approaches to information integration on the web (such as the ones mentioned above) have also adopted database-oriented representational schemes, but they do not include the tools that enable developers to create these models for real web sites. Where our approach becomes challenging (and could break down) is in situations where the "natural" way to represent a web site is not possible due to limitations of the underlying representation.

One such limitation is that Ariadne cannot reason about recursive relations. (To do this properly would require query plans to contain loops.) This has many practical ramifications. For example, consider web pages that have a 'more' button at the bottom, such as Alta Vista's response pages. It would be natural to represent each 'more' button as a pointer to the next page in a list, but there is no way to do this without a recursive relation. Instead, we can build knowledge about 'more' buttons in our wrapper generation tools, so the process of following a 'more' buttons is done completely within a wrapper, hiding the complexity from the query planner.

Another ramification of the planner's inability to reason about recursive relations shows up with hierarchical indexes like Yahoo, where there is no fixed depth to the hierarchy. The natural way to model such pages is with a parent-child relation. Instead, the alternative is to build a more sophisticated wrapper that computes the

transitive closure of the parent-child relationship, so that we can obtain all of a node's descendants in one step.

There is an obvious tension between the expressiveness of the representation and the burden we place on the modeling tools. Some researchers, such as Friedman et al. [14], have introduced more expressive representations for modeling web pages and their interconnections. Our approach has been to keep the representation and planning process simple, compensating for their limitations by relying on smarter modeling tools. As we have described, the advantage is that we can incrementally build a suite of modeling tools that use machine learning, statistical inference, and other AI techniques, producing a system that can handle a surprisingly wide range of tasks.

9. Acknowledgements

The research reported here was supported 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, in part by the United States Air Force under contract number F49620-98-1-0046, and in part by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, Cooperative Agreement No. EEC-9529152. 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.

1. José Luis Ambite. *Planning by Rewriting.* PhD thesis, Department of Computer Science, University of Southern California, 1998.
2. José Luis Ambite and Craig A. Knoblock. Planning by rewriting: Efficiently generating high-quality plans. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Providence, RI, 1997.
3. José Luis Ambite and Craig A. Knoblock. Flexible and scalable query planning in distributed and heterogeneous environments. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, Pittsburgh, PA, 1998.
4. José Luis Ambite and Craig A. Knoblock. Flexible and scalable cost-based query planning in mediators: A transformational approach. *Artificial Intelligence Journal*, 118(1-2):115–161, April 2000.
5. José Luis Ambite, Craig A. Knoblock, Ion Muslea, and Andrew Philpot. Compiling source descriptions for efficient and flexible information integration. To appear in Journal of Intelligent Information Systems, 2000.
6. Yigal Arens, Craig A. Knoblock, and Wei-Min Shen. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems, Special Issue on Intelligent Information Integration*, 6(2/3):99–130, 1996.
7. Naveen Ashish, Craig A. Knoblock, and Cyrus Shahabi. Intelligent caching for information mediators: A kr based approach. In *Knowledge Representation meets Databases (KRDB)*, Seattle, WA, 1998.
8. Naveen Ashish, Craig A. Knoblock, and Cyrus Shahabi. Selectively materializing data in mediators by analyzing user queries. In *Fourth International Conference on Cooperative Information Systems (CoopIS)*, Edinburgh, Scotland, September 1999.

9. Greg Barish, Craig A. Knoblock, Yi-Shin Chen, Steven Minton, Andrew Philpot, and Cyrus Shahabi. The TheaterLoc virtual application. In *Proceedings of Twelfth Annual Conference on Innovative Applications of Artificial Intelligence (IAAI-2000)*, Austin, Texas, 2000.
10. Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 1636–1642, August 1995.
11. M. Califf and R. Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 328–334, 1999.
12. William W. Cohen. Knowledge integration for structured information sources containing text (extended abstract). In *SIGIR-97 Workshop on Networked Information Retrieval*, 1997.
13. D. Freitag. Information extraction from HTML: Application of a general learning approach. In *Proceedings of the 15th Conference on Artificial Intelligence (AAAI-98)*, pages 517–523, 1998.
14. Marc Friedman, Alon Levy, and Todd Millstein. Navigational plans for data integration. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 67–73, Orlando, Florida, USA, August 1999. AAAI Press / The MIT Press.
15. Michael R. Genesereth, Arthur M. Keller, and Oliver M. Duschka. Infomaster: An information integration system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Tucson, AZ, 1997.
16. C. Hsu and M. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Journal of Information Systems*, 23(8):521–538, 1998.
17. R.J. Bayardo Jr., W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. Infosleuth: Agent-based semantic integration of information in open and dynamic environments. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Tucson, AZ, 1997.
18. Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1202–1207, Portland, Oregon, USA, August 1996. AAAI Press / The MIT Press.
19. Craig A. Knoblock. Planning, executing, sensing, and replanning for information gathering. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995.
20. Craig A. Knoblock, Steven Minton, José Luis Ambite, Naveen Ashish, Pragnesh Jay Modi, Ion Muslea, Andrew G. Philpot, and Sheila Tejada. Modeling web sources for information integration. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, WI, 1998.
21. Nicholas Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, Department of Computer Science and Engineering, University of Washington, 1997.
22. Chung T. Kwok and Daniel S. Weld. Planning to gather information. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, OR, 1996.
23. Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query-answering algorithms for information agents. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, OR, 1996.
24. Robert MacGregor. A deductive pattern matcher. In *Proceedings of the Seventh*

National Conference on Artificial Intelligence, Saint Paul, Minnesota, 1988.
25. G. Mecca, P. Atzeni, A. Masci, P. Merialdo, and G. Sindoni. From databases to web-bases: The araneus experience. Technical Report 34-1998, Dipartimento di Informatica e Automazione, Universita' di Roma Tre, May 1998.
26. Ion Muslea, Steven Minton, and Craig Knoblock. Hierarchical wrapper induction for semistructured information sources. *Journal of Autonomous Agents and Multi-Agent Systems*, Forthcoming.
27. S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1/2/3):233–272, 1999.
28. Jeffrey D. Ullman. Information integration using logical views. In *Proceedings of the Sixth International Conference on Database Theory*, Delphi, Greece, January 1997.
29. Zachary G. Ives Daniela Florescu Marc Friedman Alon Levy Daniel S. Weld. An adaptive query execution system for data integration. In *Proceedings of 1999 ACM SIGMOD*, Philadelphia, PA, 1999.
30. Gio Wiederhold. *Intelligent Integration of Information*. Kluwer, 1996.