# Discovering and Learning Semantic Models of Online Sources for Information Integration

**José Luis Ambite**[1], **Bora Gazen**[2], **Craig A. Knoblock**[1], **Kristina Lerman**[1], **Thomas Russ**[1]

[1] University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
{ambite,knoblock,lerman,tar}@isi.edu

[2] Fetch Technologies
841 Apollo Street, Suite 400
El Segundo, CA 90245
gazen@fetch.com

## Abstract

Much work in Information Integration and the Semantic Web assumes that rich semantic models of sources exist. In practice, there is a tremendous amount of data on the Web, but it is typically hard to find, has little or no explicit structure, and there is rarely any semantic description of the data. We describe an integrated end-to-end system that can automatically discover web sources, invoke and extract the data from them, and build their semantic models. We describe the challenges in integrating the component technologies into a unified approach to discovering, extracting and modeling new online sources. We evaluate the integrated system in three different domains and demonstrate that it can automatically discover and model new data sources.

## 1 Introduction

Information integration applications combine data from multiple sources to answer user queries or complete a task. A common assumption is that a person must first find and model the data sources, which an automated system would then use programmatically. This first step can require significant effort and must be repeated for each new data source. While various technologies, most notably the Semantic Web, have been proposed to enable programmatic access to new sources, they are slow to be adopted, and at best will offer only a partial solution because information providers will not always agree on a common schema. We have developed an alternative approach that exploits background knowledge and data semantics to automatically discover and model new data sources.

We assume we start with a set of example sources and their semantic descriptions. These sources could be Web services with well-defined inputs and outputs or even Web forms that take a specific input and generate a result page as the output. The system is then given the task of finding additional sources that are similar, but not necessarily identical, to the known source. For example, the system may already know about several weather services and then be given the task of finding new ones that provide additional coverage for the world and building a semantic description of these new weather services that makes it possible to exploit them for additional analysis.

This problem can be broken down into the following subtasks. First, given an example source, find other similar sources. Second, once we have found such a source, extract data from it. For a web service, this is not an issue, but for a Web site with a form-based interface, the source might simply return an HTML page from which the data has to be extracted. Third, given the syntactic structure of a source (i.e., the inputs and outputs), identify the semantics of the inputs and outputs of that source. Fourth, given the inputs and outputs, find the function that maps the inputs to the outputs.

We have previously developed solutions for each subtask separately. Plangprasopchok & Lerman [2007] showed that social bookmarking sites, such as del.icio.us, can be used to identify sources similar to a given source. For example, given a geocoder that maps street addresses to their geographic coordinates, the system can identify other geocoders by exploiting the keywords used to describe such sources on del.icio.us. Gazen & Minton [2005] developed an approach to automatically structure web sources without any previous knowledge of the source. Lerman et al. [2007] developed an approach to semantically label online data that uses sources with a known semantic model to then learn to label the inputs and outputs of a previously unknown source. Carman & Knoblock [2007] developed an approach to learn a semantic description, i.e., a Datalog rule that precisely describes the relationship between the inputs and outputs of a source, in terms of known sources.

In all previous work, each problem was solved independently. Here, we describe the integration of these four separate components into a single unified approach to discovering, extracting from and semantically modeling new online sources. In the previous work each of these components made assumptions that were not completely consistent with the other components. We had to address these issues to build an end-to-end system. This work provides the first general approach to automatically discover and model new sources of data. Previous work, such as the ShopBot system [Perkowitz *et al.*, 1999], has only done this in a domain-specific way where a significant amount of knowledge was encoded into the problem (e.g., shopping knowledge in the case of Shopbot).

## 2 End-to-end Discovery, Extraction, Modeling

Our end-to-end approach finds sources, invokes them and extracts the data, determines the semantic types of the out-

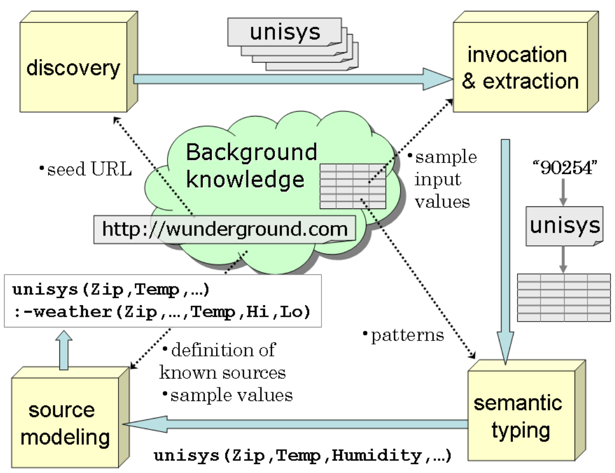

Figure 1: DEIMOS system architecture

puts, and builds the source models. The overall DEIMOS architecture is shown in Figure 1. The system starts with a known source and background knowledge about this source. It then invokes each module to discover and model new related sources. Our techniques are domain-independent, but we will illustrate them with weather domain examples.

The background knowledge required for each domain consists of the semantic types, sample values for each type, a domain input model, the known sources (seeds), and the semantic description of each seed source. For the weather domain, the background knowledge consists of: (1) Semantic types: e.g., TempF, Humidity, Zip; (2) Sample values for each type: e.g., "88 F" for TempF, and "90292" for Zip; (3) Domain input model: a weather source may accept Zip or a combination of City and State as input; (4) Known sources (seeds): e.g., http://wunderground.com; (5) Source descriptions: specifications of the functionality of the source in a formal language of the kind used by data integration systems [Levy, 2000]. The following Local-as-View [Levy, 2000] Datalog rule specifies that wunderground returns current weather conditions and five day forecast for a given zip code:

```
wunderground($Z,CS,T,F0,S0,Hu0,WS0,WD0,P0,V0,
             FL1,FH1,S1,FL2,FH2,S2,FL3,FH3,S3,
             FL4,FH4,S4,FL5,FH5,S5) :-
  weather(0,Z,CS,D,T,F0,_,_,S0,Hu0,P0,WS0,WD0,V0)
  weather(1,Z,CS,D,T,_,FH1,FL1,S1,_,_,_,_,_),
  weather(2,Z,CS,D,T,_,FH2,FL2,S2,_,_,_,_,_),
  weather(3,Z,CS,D,T,_,FH3,FL3,S3,_,_,_,_,_),
  weather(4,Z,CS,D,T,_,FH4,FL4,S4,_,_,_,_,_),
  weather(5,Z,CS,D,T,_,FH5,FL5,S5,_,_,_,_,_).
```

with an input attribute (denoted by "$") Z (Zip) and outputs CS (CityState), T (Time), FLi and FHi low and high temperatures in degF (TempInF) on the $i$th forecast day (0= today, 1= tomorrow, ...), D (Date), S (Sky conditions), Hu (Humidity), WS (Wind speed in MPH), WD (WindDir), P (Pressure in inches), and V (Visibility in miles).

DEIMOS first uses the discovery module to identify services that are likely to provide functionality similar to the seed. Once a promising set of target sources has been identified, DEIMOS uses the invocation and extraction module to determine what inputs are needed on Web forms, and how to extract the returned values. DEIMOS then invokes the semantic typing module to automatically infer the semantic types of the output data. Once DEIMOS constructs a type signature for a new source, it then invokes the source modeling module to learn its source description. We will describe each of these modules in turn and then describe some of the challenges in building an end-to-end solution.

## 2.1 Source Discovery

This module identifies sources likely to provide similar functionality as the seed. To do this, we mine a corpus of tagged documents from the social bookmarking site del.icio.us [Plangprasopchok and Lerman, 2007]. When a user bookmarks a Web source, she selects a keyword, a *tag*, from an uncontrolled personal vocabulary to describe it. Tagging enables the user to organize her bookmarks and efficiently find them later. For example, as of October 2008, http://wunderground.com has been tagged by over 3,200 people. Among the popular tags are useful descriptors such as "weather," "forecast," "meteo," as well as more general terms such as "travel" and "tools."

We can use tags to categorize Web sources, similar to the way documents are categorized using their text. The usual problems of sparseness (few unique keywords per document), synonymy (different keywords with the same meaning) and polysemy (same keyword with multiple related meanings), will also be present in this domain. Dimensionality reduction techniques, such as document topic modeling [Hofmann, 1999], were developed for text categorization domain to alleviate some of these problems. These techniques project documents from a multi-dimensional word space to a dense topic space. One such technique, Latent Dirichlet Allocation (LDA) [Blei *et al.*, 2003], is a probabilistic generative model in which a distribution of a document over a set of topics is first sampled from a Dirichlet prior. To generate each word in the document, a topic is first sampled from the distribution, then a word is selected from the distribution of topics over words. The parameters are learned using Gibbs sampling.

We use LDA to learn a compressed description, or the 'latent topics', of a collection of tagged Web sources. By analogy to document topic modeling, we view a source on del.icio.us as a document, and treat the tags across all users who bookmarked it as words. The compressed description forms the basis for comparing similarity between sources. If a source's learned topic distribution is similar to the seed's, it is likely to provide similar functionality.

To gather sources possibly related to the seed, we first collect the tags annotating the seed, and then retrieve all other sources which were annotated with those tags. After learning the latent topics, the retrieved sources are ranked according to how similar they are to the seed in the topic space.

## 2.2 Source Invocation and Extraction

To retrieve data from Web sources, DEIMOS has to figure out how to invoke the source. DEIMOS relies on background knowledge to constrain the search for valid inputs. The background knowledge contains information about the possible types of inputs expected by sources in the domain and example values of these types. In the *weather* domain, some sources may expect a single input, e.g., zipcode, while others require the city and state in separate fields.

DEIMOS repeatedly invokes the source with different permutations of possible domain input values, looking for a set of mappings that yields results pages from which it can successfully extract data. This brute force approach works as long as Web services have a small number of fields.

Web sources that generate pages dynamically in response to a query specify the organization of the page through a *page template*, which is then filled with results of a database query. The page template is therefore shared by all pages returned by the source. Given two or more sample pages, we can derive the page template and use it to extract data from the pages.

We define a *template* as a sequence of alternating stripes and slots. Stripes are the common substrings and slots are placeholders for data. We use the Longest Common Subsequence algorithm (LCS) [Gazen and Minton, 2005] to induce a template from sample pages. The common substrings are the template stripes and the gaps between stripes are the slots. Given the following snippets from two pages, "`HI:65<br>LO:50`" and "`HI:73<br>LO:61`", we can induce the template "`HI:*<br>LO:*`" where "$\star$" marks a slot.

The induced template can be used to extract data from new pages that share the same template. This involves locating the stripes of the template on the new page. Substrings that lie between the stripes are extracted as field values. Applying the template above to the snippet "`HI:50<br>LO:33`" results in two values: "`50`" and "`33`".

To extend the template idea to pages that contain lists of values, we assume that items in a list are formatted using an *item template*. Inducing item templates is more difficult than inducing page templates because unlike the boundaries of a page, the boundaries of items in a list are unknown. To reduce the complexity of finding item templates, we take advantage of the HTML structure and limit the search for list items to sibling nodes in the HTML parse tree. The output of the extraction step is a table of data fields.

## 2.3 Semantic Typing of Sources

We use background knowledge to semantically type the data fields extracted from Web sources. We have developed a content-based classification method that learns the structure of data and uses it to recognize new examples of the same semantic type. Our method is based on the domain-independent approach developed by Lerman et al. [2003] to represent the structure of data as a sequence of tokens and token types, called a *pattern*. Since tokens are strings that contain different character types: alphabetic, numeric, punctuation, *etc*, we use the token's character types to assign it to one or more general types, such as alphabetic, all-capitalized, numeric, one-digit, etc., which have regular expression-like recognizers.

The patterns associated with a semantic type can be efficiently learned from example values of the type. We use learned patterns to recognize new instances of a semantic type by evaluating how well the patterns describe the new data. We developed a set of heuristics to evaluate the quality of the match. These heuristics include how many of the learned learned patterns match data, how specific they are, and how many tokens in the examples are matched [Lerman *et al.*, 2007]. For example, a subset of the type signature-learned for source weather.unisys.com is:

```
unisys($Zip,TempF,TempC,Sky,Humidity, ...)
```

## 2.4 Source Modeling

Up to this stage, DEIMOS has learned a typed input/output signature for a novel source. However, a typed signature is only a partial description of the source's behavior. What we need is a semantic characterization of its functionality—the relationship between its input and output parameters. Such functionality can be declaratively described as a logical rule in a relational query language such as Datalog. Mediators can use Datalog *source descriptions* to access and integrate the data provided by the sources [Levy, 2000]. Specifically, DEIMOS infers a Local-as-View (LAV) description. The inference algorithm is described in detail in Carman & Knoblock [2007]. Here, we illustrate the main ideas using our running example. Consider the following *conjunctive* source description for weather.unisys.com:

```
unisys($Z,CS,T,F0,C0,S0,Hu0,WS0,WD0,P0,V0,
       FL1,FH1,S1,FL2,FH2,S2,FL3,FH3,S3,
       FL4,FH4,S4,FL5,FH5,S5):-
  weather(0,Z,CS,D,T,F0,_,_,S0,Hu0,P0,WS0,WD0,V0)
  weather(1,Z,CS,D,T,_,FH1,FL1,S1,_,_,_,_,_),
  weather(2,Z,CS,D,T,_,FH2,FL2,S2,_,_,_,_,_),
  weather(3,Z,CS,D,T,_,FH3,FL3,S3,_,_,_,_,_),
  weather(4,Z,CS,D,T,_,FH4,FL4,S4,_,_,_,_,_),
  weather(5,Z,CS,D,T,_,FH5,FL5,S5,_,_,_,_,_),
  centigrade2farenheit(C0,F0).
```

A *domain model*, consisting of predicates such as weather and centigrade2farenheit, assigns precise semantics to sources in an application domain.

The Source Modeling module of DEIMOS learns these definitions by combining *known* sources to emulate the input/output values of a new *unknown* source. For example, assume that from previous learning or from human input, the system already knows the description of wunderground (cf. Section 2) and the following temperature conversion service:

```
convertC2F(C,F) :- centigrade2farenheit(C,F)
```

Intuitively, the following join of the known sources should yield a good approximation for the input/output values of our unknown source. Replacing the known sources by their definitions yields the LAV source description shown above.

```
unisys($Z,CS,T,F0,C0,S0,Hu0,WS0,WD0,P0,V0,
       FL1,FH1,S1,FL2,FH2,S2,FL3,FH3,S3,
       FL4,FH4,S4,FL5,FH5,S5) :-
  wunderground(Z,CS,T,F0,S0,Hu0,WS0,WD0,P0,V0,
       FL1,FH1,S1,FL2,FH2,S2,FL3,FH3,S3,
       FL4,FH4,S4,FL5,FH5,S5),
  convertC2F(C0,F0)
```

Learning this definition involves searching the space of possible hypotheses (Datalog conjunctive rules) that could explain the observed inputs and outputs. DEIMOS uses Inductive Logic Programming to enumerate the search space in an efficient, best-first manner and finds the most specific rule that best explains the observed data. During this search the system uses the learned semantic types (for the unknown source) and the already known types of the background sources to prune candidate hypotheses. The system considers only conjunctive queries that join on variables of compatible types.

DEIMOS evaluates each candidate hypothesis (conjunctive query) over a set of sample input tuples, generating a set of *predicted* output tuples. It then compares the generated output tuples with those actually produced by the source being modeled to see if the predicted and actual outputs are similar. As part of its background knowledge, DEIMOS associates a similarity function with each semantic type. For numbers, the similarity is an absolute or a relative (percent) difference. For text fields, it uses string similarity metrics (*i.e.,* Levenshtein distance). DEIMOS uses the Jaccard similarity to rank different hypotheses according to the amount of overlap between the predicted output tuples and the observed ones.

## 2.5 Challenges in an End-to-End Solution

We briefly describe the primary challenges in building the end-to-end solution and how we addressed them. The first step is taking the URLs from the source discovery module and attempting to automatically invoke these sources and run the extraction module. One issue is that in many cases the system could not find the correct form to invoke on a page with multiple forms. Many sources now have a search form as the first form on the page and this is not the one that typically produces the relevant data. This was easily fixed by attempting to invoke all forms in the underlying HTML page. This allows us to find many more sources, especially in the weather domain.

Once the system invoked a source, several problems could arise in the extraction step. The extraction system did well on non-numeric fields, but it did not have a good model of numbers and often failed to extract them in a way that allowed the later components to recognize the types of the data. For example, a longitude value of "-38.524953" was divided into four tokens: "-", "38", ".", "524953". The system would then find that all of the pages had both the "-" and "." in them, and it built a template that only extracted the integer and decimal portions of the data. The next step would then fail to recognize that these numbers comprised a longitude value. We addressed this problem by rewriting the page tokenizer so that it correctly extracts numbers as "-38.524953".

In the semantic typing module, the most significant challenge relates to numeric values. The problem is that a temperature such as "10C" looks a lot like a wind speed of "10 mph" once you remove the units. Recall that the extraction module finds the values that *change* across pages. In structured sources such as these the units rarely change and are thus not extracted. Since units are typically a single token that comes immediately following the value, we built a post-processor that generated additional candidates for semantic typing that included that tokens that were most likely to capture unit information or other context. This is done by checking the document object model (DOM) of the page and appending tokens immediately following a value if it occurs at the same level in the DOM tree, which means that it likely occurs immediately after the value on the page. For "10 mph", the system would generate both "10" and "10 mph" and attempt to determine the semantic type of each of them.

Once the semantic types are assigned, the source modeling module attempts to learn a semantic description. A critical challenge is that there are often synonyms for values that are critical to invoking sources and comparing results. For example, in the flight domain some sources are invoked by the airline name and others are invoked using the corresponding 3-letter airline code. Likewise, some sources might report "Arrived" and others use "Landed". For such synonyms, we provided synonym tables as additional sources that can be used in the source modeling step.

## 3 Experimental Evaluation

We performed an end-to-end evaluation of DEIMOS on the *geospatial*, *weather*, and *flight* domains. The seeds for these domains are: geocoder.us, which returns geographic coordinates of a specified address, wunderground.com, which returns weather conditions for a specified location, and flytecomm.com, which returns the status of a specified flight.

DEIMOS starts by crawling del.icio.us to gather sources possibly related to each seed according to the following strategy. For each seed we (i) retrieve the 20 most popular tags users applied to this resource; (ii) for each of the tags, retrieve other sources that have been annotated with that tag; and (iii) collect all tags for each source. We removed low ($< 10$) and high ($> 10,000$) frequency tags, which left us with (a) 5,572 unique sources with 16,887 unique tags for *geospatial*; (b) 7,176 unique sources with 77,056 tags for *weather*; and (c) 3,562 unique sources and 14,297 tags for *flight*. Next, for each domain, we apply LDA, with the number of topics fixed at 80, to learn the hidden topics of the gathered sources. We then rank them according to how similar their topic distributions are to the seed's topic distribution, using Jensen-Shannon Divergence [Lin, 1991].

The 100 top-ranked URLs from the discovery module are passed to the invocation & extraction module, which tries to (1) recognize the form input parameters and calling method on each URL, and (2) extract the resulting output data. This produces an input type signature for these web sites and allows DEIMOS to treat them as web services. DEIMOS invoked each target source with 10 sample inputs.

Figure 2 shows the number of target sources returned by each phase. Filtering occurs at those points in the process where we apply semantic information. The filtering is largely effective. There is a sharp drop off at the Source Typing stage. While the Invocation & Extraction module is able to build a template, in many cases there is no useful data to extract. This happens beneficially when it turns out the site really isn't a good domain source. It can also be an error if sample pages from the site have some differences in the DOM structure that cannot be handled with our current heuristics (for example, a weather source which dynamically inserts a severe weather alert into results for some queries). In these cases, the extracted data often contains chunks of HTML page, which the the Source Typing module cannot recognize.

Another drop off occurs in the Semantic Modeling module. The primary reasons for failing to find a source model are one of following: (a) the source was not actually a domain source, (b) the semantic typing module learned an incorrect type signature, (c) the source extraction module extracted extraneous text following the extracted data value, or (d) there was a mismatch in the attribute values (e.g., "Landed" vs. "Arrived").

We use two check-points, at the first and last module's output, to evaluate the system by manually checking the retained

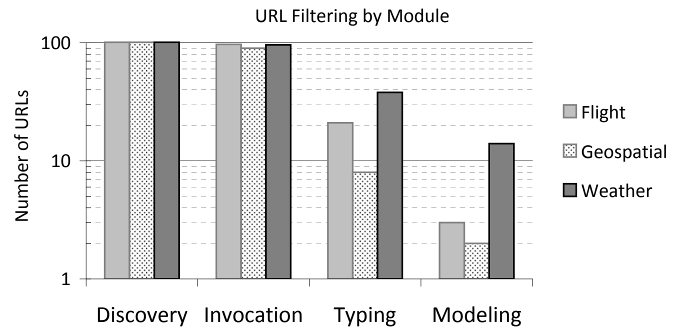

Figure 2: URL filtering by module

**Geospatial**

| | PT | PF |
|---|---|---|
| AT | 8 | 8 |
| AF | 8 | 76 |

**Weather**

| | PT | PF |
|---|---|---|
| AT | 46 | 15 |
| AF | 15 | 24 |

**Flight**

| | PT | PF |
|---|---|---|
| AT | 4 | 10 |
| AF | 10 | 76 |

(a) Source Discovery

**Geospatial**

| | PT | PF |
|---|---|---|
| AT | 2 | 0 |
| AF | 0 | 6 |

**Weather**

| | PT | PF |
|---|---|---|
| AT | 15 | 4 |
| AF | 8 | 14 |

**Flight**

| | PT | PF |
|---|---|---|
| AT | 2 | 0 |
| AF | 5 | 6 |

(b) Source Modeling

Table 1: Confusion matrices (A= Actual, P= Predicted) for each domain associated with (a) the top-ranked 100 URLs produced by the discovery module, and (b) for the descriptions learned by the semantic modeling module.

URLs. We judge the top-ranked 100 URLs produced by the discovery module to be relevant if they provide an input form that takes semantically-similar inputs as the seed and returns domain-relevant outputs. The *weather* domain had 61 such sources, *geospatial* had 16, and *flight* had 14.

Table 1(a) shows the confusion matrices associated with the top-ranked 100 sources in each domain. The precision for each domain is $50\%$, $75\%$ and $29\%$ (with the same recall values). Although there is a similar number of *geospatial* sources among the top-100 results as the *flights* sources, there were twice as many relevant *geospatial* sources among the top-ranked results compared to the *flight* sources. We suspect that the reason for this may be less consistency in the vocabulary of users tagging the flights sources.

At the second check-point, we count the services where DEIMOS learned a semantic description. Table 1(b) presents confusion matrices for this test. In the *geospatial* domain DEIMOS learned partial source descriptions for 2 out of the 8 semantically-typed sources, namely geocoder.ca and the seed. We manually checked the remaining 6 sources and found out that although some were related to geospatial topics, they were not geocoders. Similarly, in the *weather* domain DEIMOS correctly identified 15 true positives and 14 true negatives, it failed to recognize 4 weather sources and proposed descriptions for 8 sources that were not actual weather sources. The false positives (where the system found a description for a non-weather source) consisted of very short descriptions, with only a few attributes modeled. These were the result of invoking a search form, which returned the input, and one of the numeric values on the page randomly matched a seed attribute. We could possibly reduce the number of false learned descriptions with a larger number of queries.

We are ultimately interested in learning source descriptions, not just identifying sources; therefore, next we evaluate the quality of the learned semantic source descriptions. We do this by comparing the learned description to the model a user would write for the source. We report precision (how many of the learned attributes were correct), and recall (how many of the actual attributes were learned).

Consider the description learned for geocoder.ca:

```
geocoder.ca(A,_,SA,_,Z,S,_,La,Lo) :-
   geocoder.us(A,S,C,SA,Z,La,Lo).
```

with attributes A (Address), S (Street), C (City), SA (State), Z (ZIP), La (Latitude), and Lo (Longitude). Manually verifying the *attributes* of geocoder.ca yields a precision of 100% (6 correct attributes out of 6 learned) and recall of 86% (6 correct out of 7 present in the *actual* source). Similarly, the conjunctive source description learned for unisys.com has a precision of 64% (7/11) and a recall of 29% (7/24):

```
unisys($Z,_,_,_,_,_,_,_,F9,_,C,_,F13,F14,Hu,_,
        F17,_,_,_,_,S22,_,S24,_,_,_,_,_,_,_,_,
        _,S35,S36,_,_,_,_,_,_,_,_,_) :-
   wunderground(Z,_,_,F9,_,Hu,_,_,_,_,F14,F17,
        S24,_,_,S22,_,_,S35,_,_,S36,F13,_,_),
   convertC2F(C,F9)
```

The average precision ($Pr$), recall ($Re$), and $F_1$-measure ($F$) of the attributes in the source descriptions learned by DEIMOS for actual services in each domain were: $Re = 86\%$, $Pr = 100\%$ ($F = 92\%$) for the *geospatial* domain; $Re = 29\%$, $Pr = 64\%$ ($F = 39\%$) for the *weather* domain; and $Re = 35\%$, $Pr = 69\%$ ($F = 46\%$) for the *flight* domain.

We used strict criteria to judge whether a learned attribute was correct. In one case, for example, the semantic typing component mistakenly identified the field containing flight identifiers such as "United 1174" as Airline, which led to a description containing the Airline attribute. We labeled this attribute as not correct, even though the first component was the airline name. In the *weather* domain, DEIMOS incorrectly labeled the 3rd-day forecast as a 5th-day forecast, because the values of these attributes were sufficiently close. Learning using more sample inputs would reduce the chance of a fortuitous value match.

Overall, we consider these results promising. DEIMOS was able to discover Web sources, convert them into programmatically accessible services and learn semantic descriptions of these services in a completely automated fashion.

## 4 Related Work

Early work on using Inductive Logic Programming to learn semantic definitions for Internet sources was the *category translation problem* of Perkowitz *et al.* [1999]. That problem can be seen as a simplification of the source induction problem, where the known sources have no binding constraints or definitions, and provide data that does not change over time. Furthermore, it is assumed that the new source takes a single value as input and returns a single tuple as output.

More recently, there has been work on classifying web services into different domains [Heß and Kushmerick, 2003] and on clustering similar services [Dong *et al.*, 2004]. These techniques can indicate that a new service is probably a *weather* service based on similarity to other weather services. This knowledge is very useful for service discovery, but too abstract for automating service integration. We learn more expressive descriptions of web services—view definitions that describe how the attributes of a service relate to one another.

The schema integration system CLIO [Yan *et al.*, 2001] helps users build queries that map data from a source to a target schema. If we view this source schema as the set of known sources, and the target schema as a new source, then our problems are similar. In CLIO, the integration rules are generated *semi-automatically* with some help from the user.

The iMAP system [Dhamanka *et al.*, 2004] tries to discover complex (many-to-one) mappings between attributes of a source and target schema. It uses a set of *special purpose searchers* to find different types of mappings. Our system uses a general ILP-based framework to search for many-to-many mappings.

## 5 Conclusion and Future Work

We presented a completely automatic approach to discover new online sources, invoke and extract the data from those sources, learn the semantic types of their inputs and outputs, and learn a semantic description of the function performed by the source. We also presented empirical results showing that the system can learn semantic models for novel sources. Our approach is general and only requires a small amount of background knowledge for each domain. This work makes it possible to automatically find new sources for information integration tasks.

In future work, we plan to learn models of sources that cover information for which the system has no previous knowledge. In particular, we will focus on learning models of sources for which the current system can already learn partial models. For example, the system might only learn a small subset of the attributes of a particular source. We will develop an approach that can learn new semantic types (e.g., barometric pressure), new attributes (e.g., 2-day forecasted high temperature), new relations that convert between new semantic types and known types (e.g., converting Fahrenheit to Celsius; converting state names to two-letter abbreviations), and learning more accurate descriptions of the domain and ranges of sources (e.g., distinguishing between a weather source that provides information for the US versus one that provides information for the world). The ability to learn models of sources that go beyond the current knowledge within a system will greatly expand the range of sources that the system can discover and model automatically.

## Acknowledgments

We thank Sorin Ticrea for providing code for processing Web forms, and Dipsy Kapoor for architecting the source invocation and extraction system. This research is based upon work supported in part by the National Science Foundation under award numbers IIS-0324955 and IIS-0535182, in part by the Air Force Office of Scientific Research under grant number FA9550-07-1-0416, and in part by the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-07-D-0185/0004.

## References

[Blei *et al.*, 2003] D.M. Blei, A.Y. Ng, and M.I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[Carman and Knoblock, 2007] M.J. Carman and C.A. Knoblock. Learning semantic definitions of online information sources. *Journal of Artificial Intelligence Research (JAIR)*, 30:1–50, 2007.

[Dhamanka *et al.*, 2004] R. Dhamanka, Y. Lee, A. Doan, A. Halevy, and P. Domingos. IMAP: Discovering complex semantic matches between database schemas. In *Proceedings of SIGMOD*, 2004.

[Dong *et al.*, 2004] X. Dong, A.Y. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In *Proceedings of VLDB*, 2004.

[Gazen and Minton, 2005] B. Gazen and S. Minton. Autofeed: an unsupervised learning system for generating webfeeds. In *K-CAP '05: Proceedings of the 3rd international conference on Knowledge capture*, 2005.

[Heß and Kushmerick, 2003] A. Heß and N. Kushmerick. Learning to attach semantic metadata to Web Services. In *Proc. Int'l Semantic Web Conference*, 2003.

[Hofmann, 1999] T. Hofmann. Probabilistic latent semantic analysis. In *Proc. of UAI*, 1999.

[Lerman *et al.*, 2003] K. Lerman, S. Minton, and C.A. Knoblock. Wrapper maintenance: A machine learning approach. *Journal of Artificial Intelligence Research*, 18:149–181, 2003.

[Lerman *et al.*, 2007] K. Lerman, A. Plangprasopchok, and C.A. Knoblock. Semantic labeling of online information sources. *International Journal on Semantic Web and Information Systems, Special Issue on Ontology Matching*, 3(3):36–56, 2007.

[Levy, 2000] A.Y. Levy. Logic-based techniques in data integration. In Jack Minker, editor, *Logic-Based Artificial Intelligence*. Kluwer Publishers, 2000.

[Lin, 1991] J. Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991.

[Martin *et al.*, 2004] D. Martin, M. Paolucci, et al. Bringing semantics to web services: The OWL-S approach. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition*, 2004.

[Perkowitz *et al.*, 1999] M. Perkowitz, R.B. Doorenbos, O. Etzioni, and D.S. Weld. Learning to understand information on the internet: An example-based approach. *Journal of Intelligent Information Systems*, 8:133–153, 1999.

[Plangprasopchok and Lerman, 2007] A. Plangprasopchok and K. Lerman. Exploiting social annotation for resource discovery. In *AAAI workshop on Information Integration on the Web (IIWeb07)*, 2007.

[Yan *et al.*, 2001] L.L. Yan, R.J. Miller, L.M. Haas, and R. Fagin. Data-driven understanding and refinement of schema mappings. In *Proceedings of SIGMOD*, 2001.