Exploiting Data Semantics to Discover, Extract, and Model Web Sources

José Luis Ambite, Craig A. Knoblock, Kristina Lerman, Anon Plangprasopchok, Thomas Russ
USC Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292, USA
{ambite,knoblock,lerman,plangpra,tar}@isi.edu

Cenk Gazen, Steven Minton
Fetch Technologies
2041 Rosecrans Ave, El Segundo, CA 90245
{gazen,minton}@fetch.com

Mark Carman
Faculty of Informatics, University of Lugano
Via Buffi 13, CH-6904 Lugano, Switzerland
mark.carman@lu.unisi.ch

Abstract

We describe DEIMOS*, a system that automatically discovers and models new sources of information. The system exploits four core technologies developed by our group that makes an end-to-end solution to this problem possible. First, given an example source,* DEIMOS *finds other similar sources online. Second, it invokes and extracts data from these sources. Third, given the syntactic structure of a source,* DEIMOS *maps its inputs and outputs to semantic types. Finally, it infers the source's semantic definition, i.e., the function that maps the inputs to the outputs.* DEIMOS *is able to successfully automate these steps by exploiting a combination of background knowledge and data semantics. We describe the challenges in integrating separate components into a unified approach to discovering, extracting and modeling new online sources. We provide an end-to-end validation of the system in two information domains to show that it can successfully discover and model new data sources in those domains.*

1. Introduction

An assumption in much of the work on data mining is that a person must first find and model the information from which an automated system would then perform the data mining. This first step can require significant effort and must be repeated for each new data source. An alternative that we explore in this paper is to exploit a combination of background knowledge and data semantics to automatically discover and model new sources of information.

In this work, we assume that we start with a set of example sources and semantic descriptions of those sources. These sources could be web services with well defined inputs and output 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 have knowledge about several weather services and then be given the task of finding additional weather services 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 four subtasks. First, given an example source, how do we find other similar sources. Second, once we have found such a source, how do we extract the data from that source. 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 needs to be extracted. Third, given the syntactic structure of a source (i.e., the inputs and outputs), what are the semantics of the inputs and outputs of that source. Fourth, given the inputs and outputs, what is the function that maps the inputs to the outputs.

The core components that make an end-to-end solution to this problem possible have been developed in previous work. Lerman and Plangrasopchok [15] 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, which maps street addresses to its latitude and longitude coordinates, the system can identify other geocoders that are available online by exploiting the keywords used to describe such sources on a social bookmarking web site. Gazen and Minton [5] developed an approach to automatically structure web sources without any previous knowledge of the source. Lerman, Plangrasopchok, and Knoblock [9] developed an approach to semantic labeling of the online information. The system uses sources for which

it already has a semantic model to then learn to label the inputs and outputs of a perviously unknown source. Finally, Carman and Knoblock [1] developed an approach to learn a semantic description that precisely describes the relationship between the inputs and outputs of a source. This approach learns these descriptions as datalog descriptions in terms of known sources.

In all of this previous work, each of these problems were solved independently of the other components. In this paper, we describe the integration of these four separate components into a single unified approach to discovering, extracting and modeling new online sources. The challenge in integrating these separate components is that in the previous work each of these systems makes assumptions that are not necessarily consistent with the other components. To build an end-to-end approach, we had to address these issues. This work provides the first general approach to automatically discover and model new sources of data. Previous work, such as the ShopBot system [13], has done this in a domain-specific way where a significant amount of knowledge is encoded into the problem (e.g., shopping knowledge in the case of Shopbot), but there are no general approaches to this problem.

In this paper we describe DEIMOS (Discovery, Extraction, and Inference of Models Of Sources), a system that exploits background knowledge in order to discover, extract from, and model online data sources. In the remainder, we first describe the architecture of DEIMOS and the background knowledge it needs (Section 2). Second, we describe the integrated steps to discover new sources (Section 3), invoke and extract data from the discovered sources (Section 4), semantically type the inputs and outputs of these discovered sources (Section 5), and then semantically modeling the function performed by these sources (Section 6). Third, we present results of an end-to-end validation in two information domains, where the only input to the system is an example of a source in that domain and a semantic description of that source, and the system discovers and builds a semantic model of the discovered sources (Section 7). Finally, we compare with related work (Section 8) and conclude with a discussion and directions for future research (Section 9).

2. Architecture and Background Knowledge

The architecture of DEIMOS appears in Figure 1, which shows how the individual components of the system interact with each other and with the background knowledge. We illustrate our terminology on the geospatial domain. The background knowledge for the geospatial domain, which contains geocoding and related services, consists of (1) **Semantic types:** e.g., Address, Latitude, Zipcode; (2) **Sample values for each type:** e.g., "123 Elm Street" for Address, and "90292" for Zipcode; (3) **Possible inputs and outputs:** a source may accept a full Address or a combination of Street and Zipcode (4) **Known sources:** e.g., *http://geocoder.us*; (5) **Source descriptions:** specifying the functionality of the source in a formal language of the kind used by data integration systems [10], e.g., the following datalog rule specifies that geocoder.us returns geographic coordinates of an address:

```
geocoder.us(Address, Street, City, StateAbbr, ZIP,
    Latitude, Longitude):-
  Address(Address, Street, City, StateAbbr, State, ZIP,
    CountryAbbr, Country, Latitude, Longitude)
```

DEIMOS components rely on the information contained in the background knowledge for the domain to automate the process of identifying and modeling new data sources for the domain, as shown in Fig. 1. DEIMOS starts by identifying services that are likely to provide functionality similar to that provided by a known source in the domain, that we call the *seed*. A sample seed for the geospatial domain is *geocoder.us*. Once a promising set of Web sources has been identified, DEIMOS has to invoke the sources and extract the returned data. This process involves determining what inputs are needed on Web forms, and how to extract and interpret the returned values. This provides a input/output type signature of the service. DEIMOS exploits background knowledge, both in the use of known semantic types for the input values with which to query the services, and also to automatically infer semantic types of the output data. Once DEIMOS constructs a type signature for a new source, it can programmatically invoke the source and analyze its output to infer a source description of the new source.

3. Source Discovery

The goal of Source Discovery is to identify services that are likely to provide similar functionality to the seed. One approach to solve this problem is to query a Web search engine with terms that appear in the seed to identify all other sources that use the same terms. Then, we could use a traditional Information Retrieval-based similarity measure, e.g., TF/IDF, to find sources similar to the seed. Such approaches are not likely to produce high quality results for Web sources that generate their contents dynamically in response to a query. First, such sources may contain very little text in the input form page, where the user enters the query. In addition, no descriptive terms indicating the output of the source may be included on this page. At best, a search engine may rely on the metadata supplied by the source or the anchor text in the pages that link to the source. Woogle [3] is one of the few search engines to address this problem by indexing Web services based on the syntactic metadata provided in the WSDL files. However, this approach applies only to Web services and not to a host of other dynamic Web sources whose discovery we plan to automate.

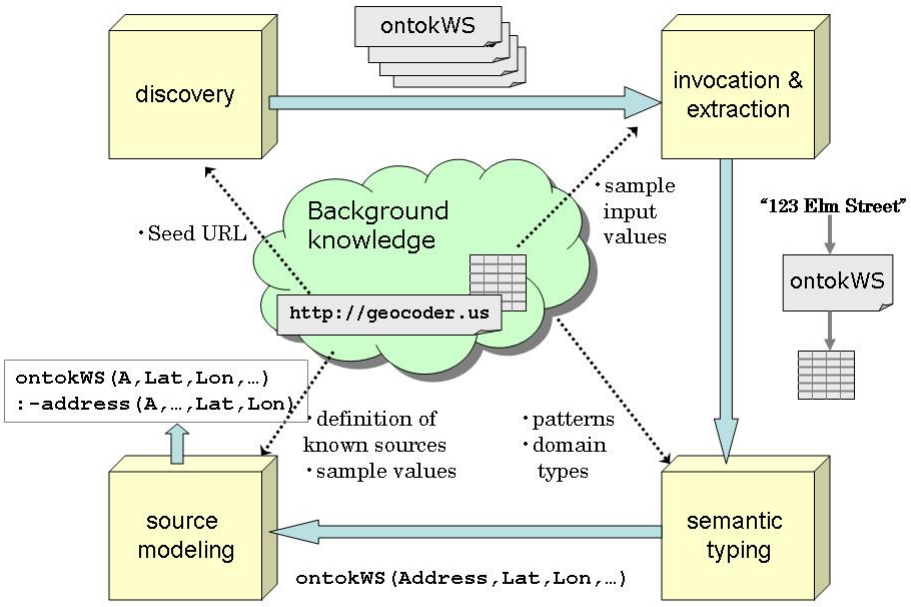

Figure 1. DEIMOS **system architecture**

Instead, in order to discover sources similar to the seed, we mine a corpus of tagged documents from the social bookmarking site *del.icio.us*. Tagging has become a popular method for annotating content. When a user tags an object, *e.g.*, a Web document on *del.icio.us*, a scientific paper on *CiteULike*, or an image on *flickr*, she labels it with a *tag* freely chosen from an uncontrolled personal vocabulary. Tagging enables the user to organize objects in order to efficiently find them later. We assume that users annotate Web sources according to their functionality. As of August 4, 2008, *geocoder.us* has been tagged by over 1,600 people on *del.icio.us*. Among popular tags are useful descriptors of the service such as "geocoding," "gps," "mapping," "tool," and "geocode."

We have developed a probabilistic model which groups 'similar' tags to learn a compressed description of a collection of tagged Web sources [15]. The compressed description, or 'latent topics', forms the basis for comparing similarity between sources. In other words, if a source's topic distribution is similar to the seed's, the source is likely to provide the same functionality. The retrieved sources are, therefore, ranked according to how similar they are to the seed in the learned topic space.

For the geospatial domain, among the results are URLs of sources that provide information about geocoding products and services, e.g., *http://www.nacgeo.com/geocode.asp* and *http://www.pushpin.com/*; a Canadian geocoder *http://geocoder.ca*; multi-country geocoders such as *http://www.travelgis.com/geocode/*, as well as other US-specific geocoders, such as *http://www.ontok.com/geocode/* and *http://www.lat-long.com/*.

4. Source Invocation and Extraction

In order to retrieve data from Web sources, DEIMOS has to identify input forms on Web pages and parse them to determine how to invoke the source. The returned pages are then processed to extract data from it. The page is divided into virtual "columns" which contain strings. The algorithm used by DEIMOS often produces a large number of columns (sometimes more than 1,000 per page). It is the task of the semantic typing component of DEIMOS to choose a subset of those columns to form the type signature for that particular source.

4.1. Source invocation

DEIMOS relies on the 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. In the *geospatial* domain, some sources may expect a single input in the form of an address, while others require the street and city to be entered separately. Table 1 shows possible inputs

Inputs		Outputs
Type	**Value**	
Address	123 Elm Street, El Segundo, CA 90294	Field of type Address
Street	123 Elm Street	Field of type Street
City	El Segundo	Field with one of City, StateName, StateAbbr,
StateAbbr	CA	ZipCode5Digit, Zipcode5Plus4Digit
CityState	El Segundo, CA	Field with one of Latitude,
ZipCode5Digit	90294	LatitudeDMS, LatitudeDMS-D
Latitude	34.898748	Field with one of Longitude,
Longitude	-118.037684	LongitudeDMS, LongitudeDMS-D

Table 1. Possible input and output types for a geocoding service, along with some sample values.

for a *geospatial* service, along with sample values for each type. DEIMOS repeatedly invokes the source with different permutations of possible domain input values until a set of mappings is found that yields results pages from which it can successfully extract data. DEIMOS stops as soon as one successful mapping is found. The brute force approach works because in many domains most Web forms have a fairly small number of fields.

4.2. Data extraction

We are interested in modeling Web sources that generate pages dynamically in response to a query. These sources specify the organization of the page through a *page template*, which is then filled with results of a database query. The page template is, thus, 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.

The page template can be thought of as the grammar that generates the pages. DEIMOS induces a grammar for the Document Object Model (DOM) trees of the pages [5]. The kinds of grammars it induces allow us to extract single data items as well as lists, where the rows of a list contain data items or nested lists. It finds the grammar by comparing pages to identify repeating sub-structures and merges grammars into a more general one. Both stages use ideas based on templates. A template is a sequence of alternating slots and stripes where the stripes are the common sub-structures among all the pages and slots are the placeholders for pieces of data. One way to find the template of a set of pages is to find the longest common subsequence (LCS) of all the pages. The LCS immediately gives the stripes of the template and with a little bookkeeping, the slots can also be found.

We can extend the template idea to DOM trees. Given a set of sequences of DOM elements, we find the LCS and then for each element in the LCS, we recursively apply the algorithm to the set of child elements.

Once we have the template, we can use it to extract the slots by finding the nodes in the original DOM structures that are not in the stripes of the template. Data in the same slot is added to the same column. The output of this step is a (sometimes large) table of columns of data.

5. Semantic Typing of Sources

We leverage background knowledge in order to semantically type the data returned by Web sources. Data usually has some structure: phone numbers, prices, addresses, *etc.* follow some format. We have developed a content-based classification method that learns this structure and uses it to recognize new examples of the same semantic type. Below is a brief explanation of the approach.

5.1. Semantic labeling

We developed a domain-independent language [7] that represents 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: e.g., alphabetic, all-capitalized, numeric, one-digit, etc. The general types have regular expression-like recognizers.

The patterns associated with a semantic data type can be efficiently learned from example values of the type. We can later 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, how many tokens in the examples are matched, and so on. We found that our system can accurately recognize new instances of known semantic types from a variety of domains, such as weather and flight information, yellow pages and personal directories, financial and consumer product sites, etc [8, 9].

Input:	field "addr" of type Address
Outputs:	column6 of type Longitude, 115.62
	column7 of type Latitude, 92.4
	column8 of type Street, 3.94
	column16 of type StateAbbr,14

Table 2. Mapping information for a geospatial source http://ontok.com

5.2. Identifying Column Types

To validate a results page, the column view of the pages provided by the Source Extraction component is processed to produce a mapping of a subset of columns to semantic types. The background knowledge is used in two ways here. First, it provides sample values of each semantic type to build recognizers, or 'patterns', which DEIMOS uses to automatically infer semantic types of the outputs. Second, the background knowledge also contains information about what types of outputs can be expected from a source. The output types are divided into required and optional return values, as shown in Table 1 for a *geospatial* source. All required and a specified fraction of the optional values must be identified in the output for a source to be deemed a proper service for the domain. For a *geospatial* service 80% of the optional types have to be identified.

Once the mapping of columns to types is completed, we have a type signature of the Web source. This provides the input to the next stage of our processing, the derivation of the source model. As an example, the type signature of *http://ontok.com*, which was identified by the Discovery component as a possible geospatial service, is shown in Table 2.

There are several reasons why a type signature can fail to be found. First, we may not be able to invoke the form with a set of inputs that provides a consistently identifiable output. This is usually because the form does not really represent an example of the type of web source that we are trying to learn. Recall that the source discovery phase is not a very precise procedure. The precision is supplied by determining whether we can invoke the source successfully. Second, we may not get reasonable output because incomplete modeling of potential input value combinations. Third, the system may not be able to parse the result pages consistently. If the returned pages are highly variable, perhaps because of dynamically inserted advertisements, this can throw off the grammar inference stage of the column splitting algorithm. Finally, a source may not return enough of the fields that are needed by the domain. This is usually legitimate grounds for rejecting the source, but in some cases it may indicate that the page source is just a more minimalistic web source. For example, a minimalist return page for a geocoder, which just returns latitude and longitude but doesn't echo the input address, would not be identified by the criteria that we used for the domain.

6. Source Modeling

At this stage, DEIMOS has learned a typed input/output signature for a novel source. For example, the system represents the *ontok.com* geocoding service of Table 2 by the signature: `ontok.com($Address, Address, Longitude, Latitude, Street, StateAbbr)`. However, a typed signature is only a partial description of the behavior of a source. What we want is a characterization of the functionality of the service, that is, the relationship between the input and output parameters of the service. Such functionality can be declaratively described as a logical rule in a relational query language such as Datalog. Once the functionality of a source is specified in such declarative form, the data provided by the resource can be accessed and integrated using data integration techniques [19, 10, 17]. Specifically, DEIMOS infers a Local-as-View (LAV) description [10, 18]. The inference algorithm for source definitions is described in detail in Carman & Knoblock [1]. Here, we illustrate the main ideas through an example.

Consider a simple Web Service called `CalculateDistance`, which takes as input two zipcodes and returns the distance between the centroids of the zipcodes in miles. The LAV rule describing the functionality of the source is:

```
CalculateDistance($zip1, $zip2, dist):-
  centroid(zip1, lat1, long1),
  centroid(zip2, lat2, long2),
  greatCircleDist(lat1, long1, lat2, long2, dist2),
  convertKm2Mi(dist1, dist2).
```

The predicates `centroid`, `greatCircleDist`, and `convertKm2Mi` are part of the domain model and are used to assign precise commonly-understood semantics to sources from a given domain.

The task of the source modeling module of DEIMOS is to learn this type of definitions. Our approach is to combine *known* sources to try to emulate the input/out values of the new unknown source. For example, assume that the system already contained the following source descriptions (obtained from a human or from the output of previous learning):

```
source1($zip, lat, long):- centroid(zip, lat, long).
source2($lat1, $long1, $lat2, $long2, dist):-
  greatCircleDist(lat1, long1, lat2, long2, dist).
source3($dist1, dist2):- convertKm2Mi(dist1, dist2).
```

In this case, it should be fairly obvious that the following join of the known sources should yield a good approximation for the input/output values of our unknown source `CalculateDistance`.

```
CalculateDistance($zip1, $zip2, dist):-
  source1(zip1, lat1, long1),
  source1(zip2, lat2, long2),
  source2(lat1, long1, lat2, long2, dist2),
  source3(dist2, dist).
```

Replacing the known sources by their definitions yields the LAV source description shown above.[1] Learning this definition involves searching the space of possible hypotheses, that is, conjunctive rules in Datalog that could explain the inputs and outputs observed. DEIMOS employs Inductive Logic Programming techniques to enumerate the search space in an efficient best-first manner. During this search the system takes advantage of the semantic types that have been learned for the unknown source, and that are already known for the background sources, in order to prune the number of possible candidate hypothesis. The system considers only conjunctive queries that join on variables of compatible types.

The system evaluates each candidate hypothesis (a conjunctive query) over a set of sample input tuples, thus generating a set of *predicted* output tuples. The induction system 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. An example of such a test is shown in the table below:

$zip1	$zip2	dist *(actual)*	dist *(predicted)*
80210	90266	842.37	843.65
60601	15201	410.31	410.83
00000	90210	Invalid!	NULL
10005	35555	899.50	899.21

As part of its background knowledge DEIMOS associates a similarity function with each of its semantic types. For numeric values, a similarity can be expressed as an absolute or a relative (percent) difference. For text fields, it uses string distance metrics such as Levenshtein distance. Using these similarity metrics the system compares each pair of predicted versus observed values. Then, DEIMOS uses the Jaccard similarity to rank different hypotheses according to the amount of overlap between the predicted output tuples and the observed ones.

The system continues its search for the best hypothesis until it discovers the most specific rule that best explains the observed data. Thus if the resource provides only distances between zipcodes in California, the system will learn that the inputs to the source must indeed be Californian zipcodes.

The induction system has been tested on a number of different domains including geospatial data sources, financial data sources as well as weather, hotel and automobile related Web Services. In all cases it was able to learn high quality declarative descriptions for the functionality of the different information sources, that could be used by an information integration system to integrate and access the source's data [1].

7. Experimental Evaluation

We present results of an end-to-end evaluation of the system on the *geospatial* and *weather* domains. The *geospatial* domain consists of sources, such as *http://geocoder.us*, that returns geographic coordinates for a specified address. The *weather* domain contains sources, such as *http://wunderground.com*, that return weather conditions at a specified location.

Source Discovery. DEIMOS crawls *del.icio.us* to collect sources potentially similar to the seed (*http://geocoder.us* or *http://wunderground.com* respectively for the two domains). For each seed it retrieves the 20 most popular tags users applied to this source. Then, for each of the tags, retrieves other sources that have been annotated with that tag. Finally, for each source, it collects all bookmarks. This process results in a set of over 15 million (source-user-tag triples) for the domains.

We trained the model on the data and used the learned topic distributions to compute the similarity of resources in each data set to the seed. We evaluated the performance of the model by manually checking the top-ranked 100 resources produced by the model. The resource is judged to have the same functionality, if it provides an input form that takes semantically the same inputs as the seed and returns semantically the same data. Among the 100 highest ranked URLs for each domain, Source Discovery was able to identify 20 relevant *geospatial* sources, 70 relevant *weather* sources.

Source Invocation & Extraction and Semantic Typing. Source Invocation & Extraction attempts to (1) recognize the form input parameters and calling method, and (2) learn an extractor for the resulting output. If successful, DEIMOS can then call the websites programmatically as web services. Then, Semantic Typing generates type signatures for these services.

Given the top 100 URLs identified by Source Discovery, these modules generated two semantically-typed *geospatial* sources and six semantically-typed *weather* sources.

[1] In general, after unfolding the source definitions, the system needs to simplify the resulting conjunctive query, since there may be provably redundant predicates.

As an example, the type signatures learned for *ontok.com* and *geocoder.ca* are:

```
ontok($Address, Longitude, Latitude, Street, StateAbbr)
geocoder.ca($Address, CityState, StateAbbr, Street,
    Latitude, Longitude)
```

Semantic Modeling *Geospatial Domain*. As background knowledge for the *geospatial* domain, we defined the following a source description for the seed source:

```
geocoder.us(Address, Street, City, StateAbbr, ZIP,
    Latitude, Longitude):-
  Address(Address, Street, City, StateAbbr, State, ZIP,
    CountryAbbr, Country, Latitude, Longitude)
```

Given this source, DEIMOS was able to infer the following hypothesis and corresponding source descriptions:

```
ontok($Address, Longitude, Latitude, _, _):-
  geocoder.us(Address, _, _, _, _, Latitude, Longitude)

ontok($Address, Longitude, Latitude, _, _):-
  Address(Address, _, _, _, _, _, _, _, Latitude, Longitude)

geocoder.ca($Address, _, StateAbbr, Street, Latitude, _):-
  geocoder.us(Address, Street, _, StateAbbr, _, Latitude, _)

geocoder.ca($Address, _, StateAbbr, Street, Latitude, _):-
  Address(Address, Street, _, StateAbbr, _, _, _, _, Latitude, _)
```

Analyzing the results, the reason that some of the attributes could not be mapped hinged on extraction errors. For the street and state attributes of *ontok*, the extractor had left some HTML tokens in the output values (for example, "<font size='-1'>601 WHITEHEAD ST KEY WEST, FL 33040" and "<font size='-1'>FL"), so that the similarity metric associated with these types could not consider them similar to the predicted values. For the longitude attribute of *geocoder.ca* the extractor omitted the sign of the longitude, for example, it extracted 118.440470 instead of -118.440470. Again this thwarted the comparison between values and prevented learning a mapping.

Weather Domain. As background knowledge for the *weather* domain we used these sources: *wunderground.com* and a conversion function:

```
wunderground($Zip, Humidity, TempF_hi, TempF_low,
    TempF_hinextday, Sky, PressureInches, WindDirection):-
  weather(Zip, TempF_hi, TempF_low, TempF_hinextday,
    Humidity, Sky, PressureInches, WindDirection)

ConvertC2F($TempC, TempF):-convertTemp(TempC, TempF)
```

The system was able to discover and model two additional websites: *weather.unisys.com* and *timetemperature.com*. The learned source descriptions are:

```
unisys($Zip, Sky, TempF_hi, TempC, _, _, _):-
  weather(Zip, TempF_hi, _, _, _, Sky, _, _),
  convertTemp(TempC, TempF_hi)

timetemperature($Zip, _, Sky, _, _, TempF_low, TempF_hinextday, _):-
  weather(Zip, _, TempF_low, TempF_hinextday, _, Sky, _, _)
```

The ground truth signatures for these sources are:

```
unisys($Zip, Sky, TempF_current, TempC_current, Humidity,
  WindDirection, TemperatureF, PressureMb)

timetemperature($Zip, TimeZone, Sky, Humidity,
  PressureInches, TempF_lowtonight, TempF_hinextday, TempF_lownextday)
```

For *unisys.com* the system was able to learn a correct conjunctive source description that included not only the `weather` domain predicate but also the `convertTemp` translation function. The system correctly learned four attributes and missed three. Labeling $\texttt{TempF}_{\texttt{current}}$ with $\texttt{TempF}_{\texttt{hi}}$ is the best that the system could do given that the seed source did not include $\texttt{TempF}_{\texttt{current}}$. The system is able to correctly infer the translation between $\texttt{TempF}_{\texttt{current}}$ and $\texttt{TempC}_{\texttt{current}}$. The three missing attributes could not be learned due to extraction errors and to value variability between the sources. For example, since the extracted values for the last attribute were a combination of temperature and pressure values, no correct mapping could be made. Also, we discovered that the sites actually report significantly different values for `Humidity`, even though we extracted values for the seed and the target sources in close temporal proximity.

For *timetemperature.com* the system correctly learns the source description and four of the attributes, while it misses another four attributes. We argue that matching $\texttt{TempF}_{\texttt{lowtonight}}$ and $\texttt{TempF}_{\texttt{low}}$ is acceptable given the seed description. The system correctly learned $\texttt{TempF}_{\texttt{hinextday}}$, but could not learn $\texttt{TempF}_{\texttt{lownextday}}$ since such values are not present in the seed. `Humidity` values were again too variable. The values of the second argument of *timetemperature.com* were actually time zone codes, which were extracted and incorrectly labeled as `WindDirection` (due to three capitalized letters being a pattern for wind direction, e.g., WSW). Obviously these values would not match the seed source. We remark that the end-to-end system did not produce any incorrect source description, only incomplete ones. Since the final source modeling step relies on actual values from the sources, in addition to the semantic types, we conjecture that it is unlikely that the both values and semantic types would both match incorrectly.

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

8. Related Work

Early work on the problem of learning semantic definitions for Internet sources was performed by [14], who defined the *category translation problem*. 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. To find solutions to this problem, the authors too used a form of inductive search based on an extension of FOIL.

More recently, there has been some work on classifying web services into different domains [6] and on clustering similar services together [4]. This work is closely related, but at a more abstract level. Using these techniques one can state that a new service is probably a *weather* service because it is similar to other weather services. This knowledge is very useful for service discovery, but not sufficient for automating service integration. In our work we learn more expressive descriptions of web services, namely view definitions that describe how the attributes of a service relate to one another.

The schema integration system CLIO [20] 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 [2] 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. Since our system can perform a similar task to iMAP, we tested it on the hardest problem used to evaluate iMAP. The problem involved aligning data from two online cricket databases. Our system, despite being designed to handle a more general task, was able to achieve 77% precision and 66% recall, which is comparable to the performance (accuracy) range of 50-86% reported for iMAP on complex matches with overlapping data.

Finally, the Semantic Web community have developed standards [12, 16] for annotating sources with semantic information. Our work complements theirs by providing a way to automatically generate semantic information, rather than relying on service providers to create it manually. The Datalog-based representation used in this paper (and widely adopted in information integration systems [11]) can be converted to the Description Logic-based representations used in the Semantic Web.

9. Discussion and Future Work

In this paper we presented a completely automatic approach to discover new online sources, extract the data from those sources, learn the semantic types of the inputs and outputs, and learn a semantic description of the function performed by the source. We also presented some initial results that demonstrate the the system can learn semantic models for source in which it had no previous knowledge. This work is important because it makes it possible for a system to grow the sources and data to which it has access. This work is directly relevant to other work on data mining since it makes it possible to both find new sources that can be exploited and to find connections with existing data sources that make it possible to discover new knowledge or patterns based on the relations between them.

There are a number of directions for future work. So far we have completed a proof of concept that demonstrates that we can perform the end-to-end discovery and learning. The next step is to flesh out these components and improve the interoperation between them. For example, by learning more fine-grained semantic types, we can improve the ability to learn the semantic descriptions of the sources. Similarly, by improving the extraction of the data from the web page, we can improve the ability to assign the semantic types. We also plan to run a much more comprehensive set of experiments to demonstrate that we can discover and model sources across a variety of problem domains.

Acknowledgements

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.

The U.S.Government is authorized to reproduce and distribute reports for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

References

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

[2] R. Dhamanka, Y. Lee, A. Doan, A. Halevy, and P. Domingos. imap: Discovering complex semantic matches between database schemas. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of data*, 2004.

[3] X. Dong, A. Y. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Simlarity search for web services. In *Proc. of VLDB*, pages 372–383, 2004.

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

[5] 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*, pages 3–10, New York, NY, USA, 2005. ACM.

[6] A. Heß and N. Kushmerick. Automatically attaching semantic metadata to web services. In *IJCAI-2003 Workshop on Information Integration on the Web*, 2003.

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

[8] K. Lerman, A. Plangprasopchok, and C. Knoblock. Automatically labeling the inputs and outputs of web services. In *Proc. of National Conference on Artificial Intelligence (AAAI-06)*, 2006.

[9] 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.

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

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

[12] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara. Bringing semantics to web services: The owl-s approach. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, 2004.

[13] 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.

[14] M. Perkowitz and O. Etzioni. Category translation: Learning to understand information on the internet. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1995.

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

[16] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web service modeling ontology. *Applied Ontology*, 1(1):77–106, 2005.

[17] S. Thakkar, J. L. Ambite, and C. A. Knoblock. Composing, optimizing, and executing plans for bioinformatics web services. *VLDB Journal*, 14(3):330–353, 2005.

[18] J. D. Ullman. Information integration using logical views. In *Proceedings of the Sixth International Conference on Database Theory*, pages 19–40, Delphi, Greece, January 1997.

[19] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, March 1992.

[20] L. L. Yan, R. J. Miller, L. M. Haas, and R. Fagin. Data-driven understanding and refinement of schema mappings. In *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD International Conference on Management of data*, 2001.