Automatically Constructing Semantic Web Services from Online Sources

José Luis Ambite, Sirish Darbha, Aman Goel, Craig A. Knoblock,
Kristina Lerman, Rahul Parundekar, and Thomas Russ

University of Southern California
Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
{ambite,darbha,amangoel,knoblock,lerman,parundek,tar}@isi.edu

Abstract. The work on integrating sources and services in the Semantic Web assumes that the data is either already represented in RDF or OWL or is available through a Semantic Web Service. In practice, there is a tremendous amount of data on the Web that is not available through the Semantic Web. In this paper we present an approach to automatically discover and create new Semantic Web Services. The idea behind this approach is to start with a set of known sources and the corresponding semantic descriptions and then discover similar sources, extract the source data, build semantic descriptions of the sources, and then turn them into Semantic Web Services. We implemented an end-to-end solution to this problem in a system called DEIMOS and evaluated the system across five different domains. The results demonstrate that the system can automatically discover, learn semantic descriptions, and build Semantic Web Services with only example sources and their descriptions as input.

1 Introduction

Only a very small portion of data on the Web is available within the Semantic Web. The challenge is how to make Web sources available within the Semantic Web without the laborious process of manually labeling each fact or converting each source into a Semantic Web Service. Converting an existing Web service into a Semantic Web Service requires significant effort and must be repeated for each new data source. We have developed an alternative approach that starts with an existing set of known sources and their descriptions, and then goes on to automatically discover new sources and turn them into Semantic Web Services for use in the Semantic Web.

The system starts 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 tasked with finding additional sources that are similar, but not necessarily identical, to the known source. For example, the system

A. Bernstein et al. (Eds.): ISWC 2009, LNCS 5823, pp. 17–32, 2009.

may already know about several weather sources and then be given the task of finding new ones that provide additional coverage for the world. To do this it must build a semantic description of these new weather sources to turn them into Semantic Web Services. In general, the type of source that we focus on in this paper are information-producing sources where there is a web form that takes one or more input values and produces a result page that has the same format across all output pages. We have found that this type of source is much more common than Web services.

The overall 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. Finally, given the semantic description, construct a wrapper that turns the source into a Semantic Web Service that can be directly integrated into the Semantic Web.

In previous work we have developed independent solutions to each of these subtasks. Here, we describe the integration of these separate components into a single unified approach to discover, extract from, and semantically model new online sources. In the previous work each of these components made assumptions that were not 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 discovering and modeling new sources of data. Previous work, such as the ShopBot system [16], did 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).

In this paper we present Deimos, a system that provides an end-to-end approach to discovering and building Semantic Web Services. First, we review the previous work on which the system is built (Section 2). Second, we describe the architecture of Deimos (Section 3), and describe how we built on the previous work to discover new sources (Section 3.1), invoke and extract data from the discovered sources (Section 3.2), semantically type the inputs and outputs of these sources (Section 3.3), semantically model the function performed by these sources (Section 3.4), and then use this semantic model to turn the web source into a Semantic Web Service (Section 4). Third, we present results of an end-to-end evaluation in five different 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 (Section 5). Finally, we compare with related work (Section 6) and conclude with a discussion and directions for future research (Section 7).

2 Prior Work

In previous work, we have developed the core technologies used by the integrated system. Plangprasopchok & Lerman [15] developed an automatic *source* *discovery* method that mines a corpus of tagged Web sources from the social bookmarking site del.icio.us to identify sources similar to a given source. For example, given a weather service that returns current weather conditions at a specified location, the method can identify other weather services by exploiting the tags used to describe such sources on del.icio.us. Tags are keywords from an uncontrolled personal vocabulary that users employ to organize bookmarked Web sources on del.icio.us. We use topic modeling techniques [4,10] to identify sources whose tag distribution is similar to that of the given source.

Gazen & Minton [7] developed an approach to automatically structure Web sources and *extract data* from them without any previous knowledge of the source. The approach is based on the observation that 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 automatically extract data from the pages.

Lerman *et al.* [13] developed a domain-independent approach to *semantically label* online data. The method learns the structure of data associated with each semantic type from examples of that type produced by sources with known models. The learned structure is then used to recognize examples of semantic types from previously unknown sources.

Carman and Knoblock [5] developed a method to learn a *semantic description* of a source that precisely describes the relationship between the inputs and outputs of a source in terms of known sources. This is done as a logical rule in a relational query language. A data integration system can then use these source descriptions to access and integrate the data provided by the sources [14].

3 End-to-End Discovery, Extraction, and Modeling

The overall architecture of the Deimos system is shown in Figure 1. Deimos 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 examples from the weather domain.

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

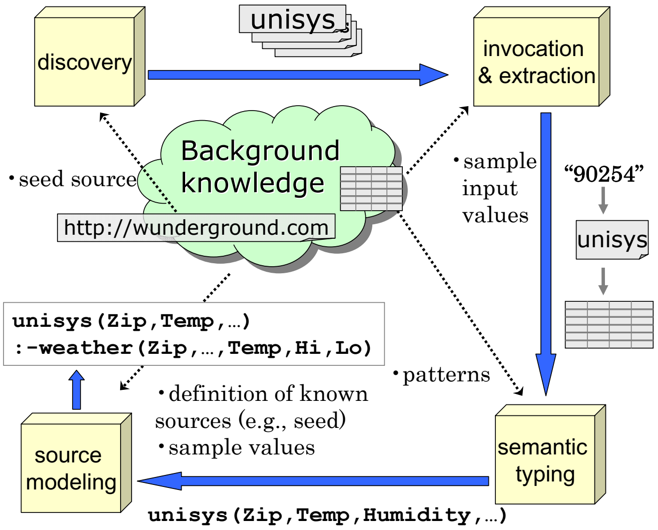

Fig. 1. DEIMOS system architecture

systems. For example, the following Local-as-View [14] Datalog rule[1] 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,_,_,_,_,_).
```

which has an input attribute (denoted by "$") Z (of type Zip) and outputs CS (CityState), T (Time), FHi and FLi high and low temperatures in Farenheit degrees (TempInF) on the ith forecast day (0= today, 1= tomorrow, ...), D (Date), S (Sky conditions), Hu (Humidity), WS (Wind speed in MPH), WD (WindDirection), P (Pressure in inches), and V (Visibility in miles). The semantics of the source are specified by the conjunctive formula in the body of the rule that uses predicates from a domain ontology (weather() in this example).

[1] We use the usual rule syntax for LAV rules common in the data integration literature. However, logically this rule should be interpreted as:

$$\text{wunderground}(\ldots) \rightarrow \text{weather}(\ldots) \wedge \text{weather}(\ldots) \wedge \ldots$$

This means that every tuple from wunderground satisfies the formula over the domain predicates (weather), but not viceversa. That is, the source is not complete.

Deimos first uses the discovery module to identify sources 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, along with the challenges in building an end-to-end solution.

3.1 Source Discovery

This module identifies sources likely to provide functionality similar to the seed. Deimos first collects popular tags annotating the seed from the social bookmarking site del.icio.us. As of October 2008, http://wunderground.com has been tagged by over 3200 people. Among popular tags are useful descriptors of the service: “weather,” “forecast,” and “meteo.” Next, Deimos retrieves all other sources that were annotated with those tags on del.icio.us. By analogy to document topic modeling, we view each source as a document, and treat the tags created by users who bookmarked it as words.

The system uses Latent Dirichlet Allocation (LDA) [4] to learn a compressed description, or ‘latent topics’, of tagged sources [17]. The learned topics form the basis for comparing similarity between sources. If a source’s topic distribution is similar to the seed’s, it is likely to have similar functionality. We rank retrieved sources according to their similarity to the seed and pass the 100 top-ranked sources to the next module. In the weather domain, among sources similar to http://wunderground.com are weather sources such as http://weather.yahoo.com and http://weather.unisys.com

3.2 Source Invocation and Extraction

To retrieve data from the discovered Web sources, Deimos has to figure out how to invoke the source. These sources typically use standard HTML forms for input and return a result page. During the invocation step, Deimos analyzes the sources’s document model and extracts forms and form elements. For each of the forms, Deimos identifies the input fields, which can be text (input) or menu (select) fields. Deimos relies on background knowledge to constrain the search for valid inputs. The background knowledge contains information about the typical input types expected by sources in the domain and sample values for each input type: e.g., *weather* sources expect zipcodes or city and state combinations, while *mutual funds* sources typically expect a fund symbol as input.

Deimos uses a brute force approach, trying all permutations of input types in the input form’s fields. We do allow for optional input fields, leaving some input fields blank. Since our domains generally have only a small number of possible input types, the combinatorics of the brute force approach are manageable. However, some types of sources, such as hotel booking sites, had so many possible form inputs that searching all combinations of possible inputs was intractable. We believe the solution to this problem is to exploit more of the context information on the form, such as the form label and variable name to narrow the possible inputs to each field. This will be a direction for future work so that we will be able to model information domains that have more numerous input fields in the forms.

Deimos repeatedly invokes the source with the different permutations of domain input values, looking for a set of mappings that yields results pages from which it can successfully extract data.

Next, Deimos extracts data from pages returned by the source in response to a query. For this, Deimos uses the Autowrap algorithm, described in [7], which exploits the regularity of dynamically generated pages. It assumes that the organization of dynamically generated page is specified through a *page template* that is 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.

A *template* is a sequence of alternating stripes and slots. Stripes are the common substrings and slots are placeholders for data. Autowrap uses the Longest Common Subsequence algorithm to induce a template from sample pages. The common substrings are the template stripes and the gaps between stripes are the slots. Given snippets from two pages, "`HI:65<br>LO:50`" and "`HI:73<br>LO:61`", it induces the template "`HI:*<br>LO:*`" where "`*`" 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`".

We modified the basic approach described above to deal with the challenges encountered while integrating the technology within Deimos. One extraction problem we encountered was that some of the strings that were useful for disambiguating data values, such as units on numbers, ended up being considered part of the page template by Autowrap. Consider, for example, a temperature value of '10 C' and a wind speed value of '10 mph', which look very similar once you remove the units. The extraction module finds strings that *change* across pages, and in structured sources such as these, the units will not be extracted because they rarely change. 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 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 the next step would attempt to determine the semantic type of each of them.

Another challenge was that for seemingly minor variations across pages, there were significant difference in the page structure, which prevented the system from finding the page template. An example of this was in a weather source where some of the cities had a weather advisory on the page. This resulted in a different underlying DOM structures and Autowrap failed to find the shared portion of the structure. To address this problem requires searching a much larger space to find the page template, so for the current set of results Deimos fails on some sources that should be learnable.

3.3 Semantic Typing of Sources

This module semantically types data extracted from Web sources using the approach described in [13]. This approach represents the structure of a data field as a sequence of tokens and syntactic types, called a pattern [12]. The syntactic types, e.g., alphabetic, all-capitalized, numeric, one-digit, have regular expression-like recognizers. The patterns associated with a semantic type can be efficiently learned from example values of the type, and then used to recognize 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 patterns match data, how specific they are, and how many tokens in the examples are matched [13].

The output of this module is a semantically typed signature of a source with its input and output parameters assigned to semantic types in the domain. For example, a subset of the type signature learned for source weather.unisys.com is:

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

The most significant challenge encountered in this module is that the typing component did not always have enough information to distinguish between two alternative types and chose the incorrect one. We plan to improve the semantic typing by using additional features of the values, such as numeric ranges, which will allow the system to make finer-grained semantic-typing distinctions.

3.4 Source Modeling

The typed input/output signature of a new source offers 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. We use the approach described in Carman & Knoblock [5] to learn a Local-as-View (LAV) description of the source (a Datalog rule) [14]. We illustrate the main ideas of the inference algorithm using our running example.

Consider the following *conjunctive* LAV 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/ontology* (consisting of predicates weather and centigrade2farenheit in the example) assigns precise semantics to sources (such as unisys) 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 the weather domain, the system already knows the description of wunderground (cf. Section 3) and the following temperature conversion service:

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

Using these known sources, the system learns the following join, which describes some of the input/output values of the previously unknown unisys source:

```
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)
```

Replacing the known sources, wunderground and convertC2F, by their definitions yields a version of the above LAV source description for unisys (cf. Section 4 for a Semantic Web version of this source description).

Learning this definition involves searching the space of possible hypotheses (Datalog conjunctive rules) that could explain the observed inputs and outputs. Deimos uses an approach based on 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 (e.g., 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. For some types we found a large variation in the values returned for the same inputs by different sources. In the weather domain, for example, the temperature and humidity values reported for the same location had a high variance. We had to allow for larger differences in the similarity function used by the source modeling module in order to discover any matches on these fields.

We encountered several challenges when integrating the source modeling component within Deimos. First, there are often synonyms for values that are critical to invoking sources and comparing resulting values. For example, in the flight domain some sources take the airline name as input and others the corresponding

3-letter airline code. We addressed the problem of synonyms and functionally-equivalent values by providing synonym mapping tables as additional sources that can be used in the source modeling step.

The second challenge is that sometimes closely-related attributes would be typed incorrectly due to precision errors on the values. For example, in the weather domain the forecast for the high temperature on the 3rd day would get confused with the high temperature for the 5th day. The problem arose because the 3rd-day and 5th-day high temperature values were very close for the set of sample input cities. This problem can be addressed by using additional input examples that can disambiguate between the attributes. However, a larger number of examples sometimes entails a greater variability of the resulting pages, which makes the extraction task harder (e.g., recall the page structure change due to weather advisory events discussed in Section 3.2).

4 Automatically Building Semantic Web Services

After the source modeling phase, Deimos constructs a semantic web service (SWS) encapsulating the discovered web source. The SWS accepts RDF input and produces RDF output according to the domain ontology. Internally, the SWS calls the discovered web form using the input values from the input RDF to the semantic web service. It then extracts the data from the resulting HTML using the learned *page template* (cf. Section 3.2). The output data obtained by applying the page template is filtered according to the learned source description (cf. Section 3.4). In this way the system is certain of the semantics of the extracted values. Finally, the extracted values are converted to RDF according to the specification of the source description.

We describe the construction of the semantic web service using our running unisys weather source example. For brevity and convenience earlier in the paper, we have used a domain model with n-ary predicates (such as weather()). However, since we are interested in producing RDF-processing semantic web services, in our source descriptions we actually use a domain ontology composed of unary and binary predicates, which can be straightforwardly translated to RDF. For example, the definition for wunderground is:

```
wunderground($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(@w0),hasForecastDay(@w0,0),hasZIP(@w0,Z),hasCityState(@w0,CS),
  hasTimeWZone(@w0,T),hasCurrentTemperatureFarenheit(@w0,F0),
  hasCurrentTemperatureCentigrade(@w0,C0),hasSkyConditions(@w0,S0),
  hasHumidity(@w0,Hu0),hasPressure(@w0,P0),hasWindSpeed(@w0,@ws1),
  WindSpeed(@ws1),hasWindSpeedInMPH(@ws1,WS0),hasWindDir(@ws1,WD0),
  hasVisibilityInMi(@w0,V0),
 Weather(@w1),hasForecastDay(@w1,1),hasZIP(@w1,Z),hasCityState(@w1,CS),
  hasLowTemperatureFarenheit(@w1,FL1),hasHighTemperatureFarenheit(@w1,FH1),
  hasSkyConditions(@w1,S1), ...
```

Thus, the RDF-like source description learned for unisys (cf. Section 3.4) is:

```
unisys($Z,_,_,_,_,_,_,_,F9,_,C,_,F13,F14,Hu,_,F17,_,_,_,_,
        S22,_,S24,_,_,_,_,_,_,_,_,_,S35,S36,_,_,_,_,_,_,_,_) :-
 Weather(@w0),hasForecastDay(@w0,0),hasZIP(@w0,Z),
  hasCurrentTemperatureFarenheit(@w0,F9),centigrade2farenheit(C,F9),
  hasCurrentTemperatureCentigrade(@w0,C),hasHumidity(@w0,Hu0),
 Weather(@w1),hasForecastDay(@w1,1),hasZIP(@w1,Z),hasCityState(@w1,CS),
  hasTimeWZone(@w1,T),hasLowTemperatureFarenheit(@w1,F14),
  hasHighTemperatureFarenheit(@w1,F17),hasSkyConditions(@w1,S24),
 Weather(@w2),hasForecastDay(@w2,2),hasZIP(@w2,Z),hasSkyConditions(@w2,S22),
 Weather(@w3),hasForecastDay(@w3,3),hasZIP(@w3,Z),hasSkyConditions(@w3,S35),
 Weather(@w4),hasForecastDay(@w4,4),hasZIP(@w4,Z),hasSkyConditions(@w4,S36),
 Weather(@w5),hasForecastDay(@w5,5),hasZIP(@w5,Z),
  hasLowTemperatureFarenheit(@w5,F13).
```

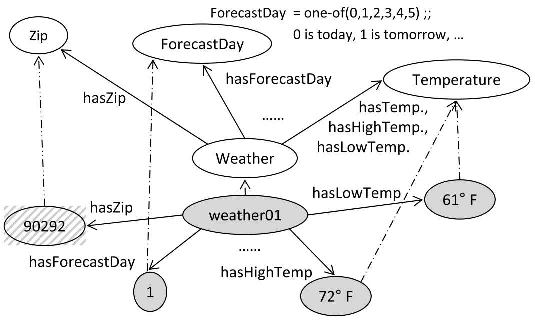

Fig. 2. Results from invoking the Semantic Web Service generated for the Unisys source

This rule means that given an RDF object Z (of type zip) as input, the SWS generated by Deimos produces as output an RDF graph consiting of 6 new objects (@w0 ... @w5) with literals for some of their properties as extracted from the web form. For example, the learned SWS for unisys produces the current temperature in centigrade and farenheit degrees, as well as the low temperature of the fifth forecast day among other data. Note that all of the weather forecast objects refer to the same zip code. This learned source description can be seen as a *lifting* rule à la SA-WSDL [11]. Figure 2 illustrates the input/output behaviour of the unisys SWS. The input RDF instance is the 90292 zip (shown with diagonal shading) and the output RDF graph of weather objects (solid shading).

5 Results on Discovering and Modeling New Services

We performed an end-to-end evaluation of Deimos on the *geospatial*, *weather*, *flight*, *currency converter*, and *mutual fund* domains. The seeds for these domains are, respectively: geocoder.us, which returns geographic coordinates of a specified address; wunderground.com, which returns weather conditions for a specified location; flytecomm.com, which returns the status of a specified flight; xe.com, which converts amounts from one currency to another based on conversion rates; and finance.yahoo.com, which provides information about mutual funds.

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 that users applied to this resource; (ii) for each of the tags, retrieve

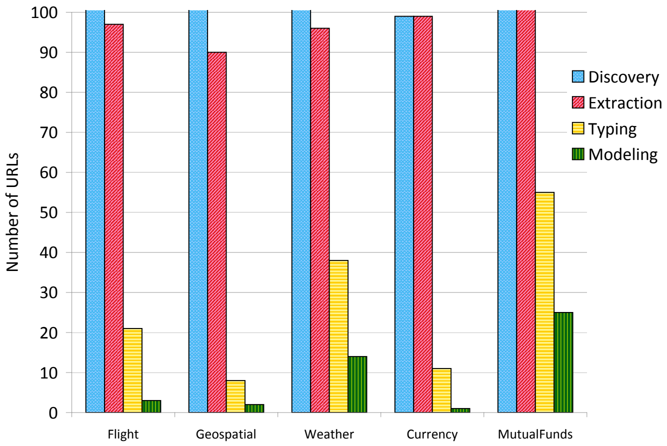

Fig. 3. URL filtering by module for all domains

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, and applied LDA, with the number of topics fixed at 80 to learn the hidden topics in each domain. We then ranked sources according to how similar their topic distributions are to the seed.

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. For the successful extractions, the semantic typing module, produces a typed input/ouput signature that allows Deimos to treat the web sources as web services. Finally, for each typed service, the source modeling module learns the full semantic description. In these experiments, Deimos invoked each target source with 10–30 sample inputs.

Figure 3 shows the number of target sources returned by each Deimos module.[2] The Invocation & Extraction module provides very little filtering because it is often able to build a template, even in cases where there is no useful data to extract. This happens in some cases when it turns out that the site really is not a good domain source. It can also occur 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 that the Source Typing module cannot recognize.

The Semantic Typing and Semantic Modeling modules provide most of the filtering. The Semantic Typing filters a source if it cannot recognize any semantic types other than the input types (which often appear in the output). The Source

[2] Note that we started with only 99 sources for the currency domain because one source was dropped from the experiments at the request of a site administrator.

Table 1. Confusion matrices (A=Actual, P=Predicted, T=True, F=False) 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

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

Currency

	PT	PF
AT	56	15
AF	15	14

Mutual funds

	PT	PF
AT	21	16
AF	16	47

(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

Currency

	PT	PF
AT	1	10
AF	0	0

Mutual funds

	PT	PF
AT	17	4
AF	8	26

(b) Source Modeling

Modeling filters a source it fails to build a model that describes any of the source outputs. 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.

We use two check-points, at the first and last module's output, to evaluate the system by manually checking the retained 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 *geospatial* had $n = 16$ relevant sources, *weather* $n = 61$, *flight* $n = 14$, *currency* $n = 71$ and *mutual funds* $n = 37$.

Table 1(a) shows the confusion matrices associated with the top-ranked 100 sources in each domain. The numbers in the column PT show how many of the top-ranked n sources were relevant (AT) and not relevant (AF) to the domain in question. The R-precision[3] for each domain is 50%, 75%, 29%, 79%, and 57%, respectively (with the same recall values). Although there is a similar number of *geospatial* and *flight* sources, there were twice as many relevant *geospatial* sources (measured by R-precision) among the top-ranked results compared to the *flight* sources. We suspect that the reason for this is less consistency in the vocabulary of users tagging the flight sources.

At the second check-point, we count the services for which DEIMOS learned a semantic description. Table 1(b) presents confusion matrices for this test. In the *geospatial* domain DEIMOS learned 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

[3] R-precision is the precision of the n top-ranked sources, where n is the number of relevant sources in our set of 100 sources.

Table 2. Precision, Recall and F1-measure for actual sources in each domain for which Deimos learned descriptions

domain	Precision	Recall	F_1-measure
weather	0.64	0.29	0.39
geospatial	1.00	0.86	0.92
flights	0.69	0.35	0.46
currency	1.00	1.00	1.00
mutualfund	0.72	0.30	0.42

Deimos correctly identified 15 relevant (true positives) and 14 not relevant (true negatives) sources; 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 with a weak pattern for its semantic type. In the *currency* domain, Deimos learned the description of one source accurately. It failed to learn description for most of the other sources because the resultant currency value after conversion could not be extracted from them because of their use of Javascript to perform the conversions without generating a new result page. In the *mutualfund* domain, Deimos correctly learned source descriptions for 17 sources. There were 8 sources that were incorrectly identified to be from this domain (false positives) because their forms returned a result page where the reported time taken to process the query (e.g., 0.15 s) was incorrectly typed as the change in net value of the fund over a day.

We are ultimately interested in learning logical source descriptions, not just identifying sources input/ouputs types. Therefore, we evaluated the quality of the learned semantic source descriptions. We do this by comparing the learned description to the model a human expert 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). The average precision, recall, and F_1-measure for the attributes in the source descriptions learned by Deimos for actual services in each domain are shown in Table 2. As an example of our evaluation methodology 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 (of semantic type 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, which is shown in Section 3.4, has a precision of 64% (7/11) and a recall of 29% (7/24).

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 quite 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. We would like to improve the precision and recall of the learned source models and we believe this can be done largely by improving the semantic typing module and learning over more data.

6 Related Work

Early work on learning semantic descriptions of Internet sources was the *category translation problem* of Perkowitz *et al.* [16]. 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. There has also been a significant amount of work on extracting and labeling data found on structured web pages (e.g., the work on Lixto [2]), but this work assumes that a user provides examples of the data to extract and a label for the extracted data, while the approach in this paper requires no labeling.

More recently, there has been work on classifying web services into different domains [8] and on clustering similar services [6]. These techniques can indicate that a new service is likely a *weather* service based on similarity to other weather services. This knowledge is 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. Hess & Kushmerick [9] developed an approach that helps users to semantically annotate Web services for data integration. It uses an ensemble of classifiers to predict how various elements of the WSDL should be annotated. The goal of this work is similar, but we handle the more general problem of supporting web sources and our approach works in a completely unsupervised fashion.

Within the bioinformatics space, where web services are widely used, there is a pressing need to build semantic descriptions of existing bioinformatics services. Belhajjame et al. [3] exploit the fact that many of these Web services have been composed into workflows and the connections in the parameters of the workflows can be used to infer constraints on the semantic types of the inputs and outputs of each of these Web services. This is a clever way to infer semantics for Web service parameters, but this method does not provide a complete semantic description of a Web service. Afzal et al. [1] developed an NLP-based approach to learning descriptions of bioinformatics Web services that attempts to extract both the type and the function performed by a service. This approach can provide broad coverage since it can be applied to a wide variety of services, however, it can only provide a high level classification of services (e.g., algorithm, application, data, etc.) and a limited description of the function. In contrast, the goal of Deimos is to build a semantic description that is sufficiently detailed to support automatic retrieval and composition.

7 Conclusion

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. These results allow us to turn an online source into a Semantic Web Service. We also presented empirical results showing that the system can learn semantic models for previously unknown sources. Our approach is general and only requires a small amount of background knowledge for each domain. This work makes it possible to automatically take existing online sources and make them available for use within the Semantic Web.

A limitation of the current work is that it can only learn a new source if it already has models of sources that contain the same information. 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 plan to develop an approach that can learn new semantic types (e.g., barometric pressure), new attributes (e.g., 10th-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 would like to thank Mark Carman, Cenk Gazen, Jeonhyung Kang, Steve Minton, Maria Muslea, and Anon Plangprasopchok for their contributions to the system and ideas presented in this paper. This research is based upon work supported in part by the National Science Foundation under award numbers IIS-0324955 and CMMI-0753124, 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

1. Afzal, H., Stevens, R., Nenadic, G.: Mining semantic descriptions of bioinformatics web services from the literature. In: ESWC 2009: Proc. of 6th European Semantic Web Conference (2009)
2. Baumgartner, R., Flesca, S., Gottlob, G.: Declarative information extraction, web crawling, and recursive wrapping with lixto. In: Eiter, T., Faber, W., Truszczyński, M. (eds.) LPNMR 2001. LNCS (LNAI), vol. 2173, pp. 21–41. Springer, Heidelberg (2001)
3. Belhajjame, K., Embury, S.M., Paton, N.W., Stevens, R., Goble, C.A.: Automatic annotation of web services based on workflow definitions. ACM Trans. Web 2(2), 1–34 (2008)
4. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. Journal of Machine Learning Research 3, 993–1022 (2003)
5. Carman, M.J., Knoblock, C.A.: Learning semantic definitions of online information sources. Journal of Artificial Intelligence Research (JAIR) 30, 1–50 (2007)
6. Dong, X., Halevy, A.Y., Madhavan, J., Nemes, E., Zhang, J.: Simlarity search for web services. In: Proceedings of VLDB (2004)
7. Gazen, B., Minton, S.: Autofeed: an unsupervised learning system for generating webfeeds. In: KCAP 2005: Proceedings of the 3rd international conference on Knowledge capture, pp. 3–10. ACM, New York (2005)
8. Heß, A., Kushmerick, N.: Learning to attach semantic metadata to Web services. In: Proc. of the International Semantic Web Conference, pp. 258–273. Springer, Heidelberg (2003)
9. Heß, A., Kushmerick, N.: Iterative ensemble classification for relational data: A case study of semantic web services. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) ECML 2004. LNCS (LNAI), vol. 3201, pp. 156–167. Springer, Heidelberg (2004)
10. Hofmann, T.: Probabilistic latent semantic analysis. In: Proc. of UAI, pp. 289–296 (1999)
11. Kopecky, J., Vitvar, T., Bournez, C., Farrell, J.: Sawsdl: Semantic annotations for WSDL and XML schema. IEEE Internet Computing 11(6), 60–67 (2007)
12. Lerman, K., Minton, S., Knoblock, C.: Wrapper maintenance: A machine learning approach. Journal of Artificial Intelligence Research 18, 149–181 (2003)
13. Lerman, K., Plangprasopchok, A., Knoblock, C.A.: Semantic labeling of online information sources. International Journal on Semantic Web and Information Systems, Special Issue on Ontology Matching 3(3), 36–56 (2007)
14. Levy, A.Y.: Logic-based techniques in data integration. In: Minker, J. (ed.) Logic-Based Artificial Intelligence. Kluwer Publishers, Dordrecht (2000)
15. Plangprasopchok, A., Lerman, K.: Exploiting social annotation for resource discovery. In: AAAI workshop on Information Integration on the Web, IIWeb 2007 (2007)
16. Perkowitz, M., Doorenbos, R.B., Etzioni, O., Weld, D.S.: Learning to understand information on the Internet: An example-based approach. Journal of Intelligent Information Systems 8, 133–153 (1999)
17. Plangprasopchok, A., Lerman, K.: Modeling social annotation: a bayesian approach. Technical report, Computer Science Department, University of Southern California (2009)