Semi-Automatically Modeling Web APIs to Create Linked APIs[⋆]

Mohsen Taheriyan, Craig A. Knoblock, Pedro Szekely, and Jose Luis Ambite

University of Southern California
Information Sciences Institute and Department of Computer Science
{mohsen,knoblock,pszekely,ambite}@isi.edu

Abstract. The objective of new service modeling approaches introduced by recent work on linked services is to integrate Linked Data and service APIs. Building these models is time consuming and difficult, which is an obstacle preventing wide adoption of these modeling approaches. We introduce an approach to semi-automatically build semantic models of the Web APIs by using examples of input values. We use these values to invoke the service and then use the input and output values to learn a semantic model of an API. These models enable a system to easily integrate data and services and to export the models as Linked APIs.

1 Introduction

The amount of data available in the Linked Data cloud continues to grow. This development has motivated studies on the relationship between services and Linked Data. Linked services [1] are the recent efforts to combine services and the Web of Data. The studies on linked services include two paths. The first one, investigated mostly by Pedrinaci et al. [8, 9], focuses on annotating services and publishing those annotations as Linked Data. The other one focuses on services that consume and produce Linked Data. Examples of this approach include Linked Data Services (LIDS) [10] and Linked Open Services (LOS) [7, 5].

Regardless of which approach we adopt to create linked services, modeling traditional services requires significant effort and time. We believe that this difficulty in wrapping traditional Web APIs is one of the main obstacles in moving toward Linked APIs. The main focus in this paper is on modeling Web APIs so that we can easily integrate them with available data, especially Linked Data, and compose them with other existing services. More precisely, we are looking for a semantic model of a service that enables the system to do the following tasks:

– Discover appropriate services based on user queries.
– Suggest to the user a list of services that can be invoked using the available (linked) data.

[⋆] This research is based upon work supported by the National Science Foundation under award number IIS-1117913.

- Invoke the service automatically from its descriptions.
- Link the service output with other available data.
- Compose data and services to achieve a specific goal defined by a user.
- Report to the user the additional information that a service can provide to augment existing data.
- Publish the service description as Linked Data.

We present an approach that enables both API providers and API users to semi-automatically model Web APIs. In our approach, examples of input values are used to invoke the API and obtain the output values. We use a machine learning technique to identify the data types of the input and output parameters according to given ontologies. Then, we explicitly specify the relationships among API parameters to expose the true functionality of the service. At the end, we are able to use the known vocabularies, such as Minimal Service Model (MSM), to publish the service models into the Linked Data cloud. We can also generate LOS descriptions from our models.

We use Karma [11] as a general information integration platform to implement our method. Karma already supports modeling structured sources including relational databases, spreadsheets, JSON and XML. Modeling services enables Karma to easily integrate data and services.

2 Semi-Automatically Building Service Models

In this section we describe the different steps of our approach to model a Web API. The main contribution of our approach is to interactively learn the semantic models of the Web APIs. Since most of the Web APIs use the HTTP GET method, in this paper, we focus on these kinds of APIs. In other words, we assume that all the required inputs for the service invocation will be embedded in the invocation URL.

The input to the system are examples of service requests provided by a user. The system uses these examples to extract the input parameters and their values. Next, input values are used to invoke the service and obtain the API response. The core part of building the service model is to align the combination of API inputs and outputs to the ontologies given by a user. The alignment has two parts. First, we exploit machine learning techniques to annotate the inputs and outputs with concepts and properties of the ontology. Second, we capture the functionality of the API by specifying the relationships between the inputs and outputs.

We use an example API to show the steps of building service models. The inputs to the API are city and state and the output values are current temperature, current dew point, humidity percent, wind speed, ICAO code of the weather station, the report date, and a description of the sky condition. An example of the API invocation URL and the API response is shown in Figure 1. The rest of this section explains how we plan to extend Karma's existing data modeling capabilities to semi-automatically build a semantic model of the APIs.

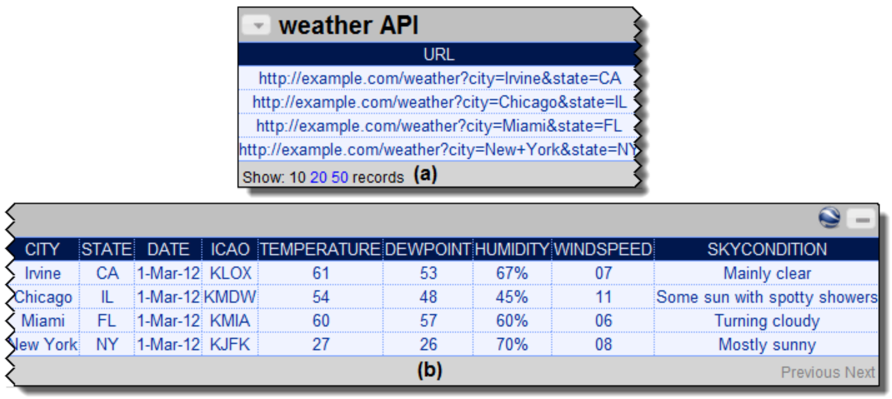
Invocation URL

```
http://example.com/weather?city=Los+Angeles&state=CA
```

JSON Results

```
"weatherObservations":[ {
    "date":"1-Mar-2012", "icao":"KLOX", "temperature":"63"
    "dewPoint":"55", "humidity":"70", "windSpeed":"09",
    "skyCondition":"Partly cloudy"
} ]
```

Fig. 1. An example weather API invocation URL and the JSON response.



weather API

URL
http://example.com/weather?city=Irvine&state=CA
http://example.com/weather?city=Chicago&state=IL
http://example.com/weather?city=Miami&state=FL
http://example.com/weather?city=New+York&state=NY

Show: 10 20 50 records (a)

CITY	STATE	DATE	ICAO	TEMPERATURE	DEWPOINT	HUMIDITY	WINDSPEED	SKYCONDITION
Irvine	CA	1-Mar-12	KLOX	61	53	67%	07	Mainly clear
Chicago	IL	1-Mar-12	KMDW	54	48	45%	11	Some sun with spotty showers
Miami	FL	1-Mar-12	KMIA	60	57	60%	06	Turning cloudy
New York	NY	1-Mar-12	KJFK	27	26	70%	08	Mostly sunny

(b) Previous Next

Fig. 2. (a) The user provides examples of the service requests. (b) Karma extracts the input parameters and invokes the API. The results of the invocation will be joined to the input data in one table.

2.1 Invoking Services

The first step in modeling an API is to collect examples of inputs and outputs. To do this, Karma will ask the user to provide some examples of the API request strings. Karma will parse the URL and automatically extracts the individual input parameters along with their values. For instance, the user just enters `http://example.com/weather?city=Los+Angeles&state=CA` as an example of a service invocation URL. Karma understands that `city` and `state` are the input parameters and `Los Angeles` and `CA` are the corresponding input values. For each request example, Karma will invoke the service to obtain the response data. Figure 2 illustrates the API invocation process in Karma.

Another way to collect examples of API inputs and outputs is to extract such information from the documentation page of the API. A study on the Web APIs [6] shows that from the APIs that are indexed by *ProgrammableWeb*[1], 83.8% of them provide an example request and 75.2% of them also provide an example response. Therefore, even without invoking many of the APIs, we can use wrappers to mine samples of input and output.

[1] http://www.programmableweb.com/

2.2 Annotating Inputs and Outputs

Once we have found examples of the inputs and outputs, the existing Karma system can model the API just like other data sources. The second step in modeling the API involves mapping each column to a node in the ontology. We use a technique based on Conditional Random Fields (CRF) to automatically assign semantic types to the columns based on the data values in each column and a set of learned probabilistic models constructed from assignments done in prior sessions [3]. Semantic types can be data properties or classes in the ontology. If the semantic type assigned by the system is incorrect, the user can select the correct one from a menu. The system learns from this assignment and records the learned assignment in its database.

In our example, we assume that in prior sessions, users trained Karma to recognize types like city, date, and temperature. For the columns that have values with the same pattern but with different types, the CRF model may be unable to distinguish them. For instance, the values in the TEMPERATURE and DEWPOINT columns are similar and Karma assigns the `temperature` data property to both columns. In this situation, the user interacts with the Karma and chooses the `dewPoint` data property as the correct type for the DEWPOINT column. In Figure 3, the gray boxes below the column names represent the assigned semantic types.

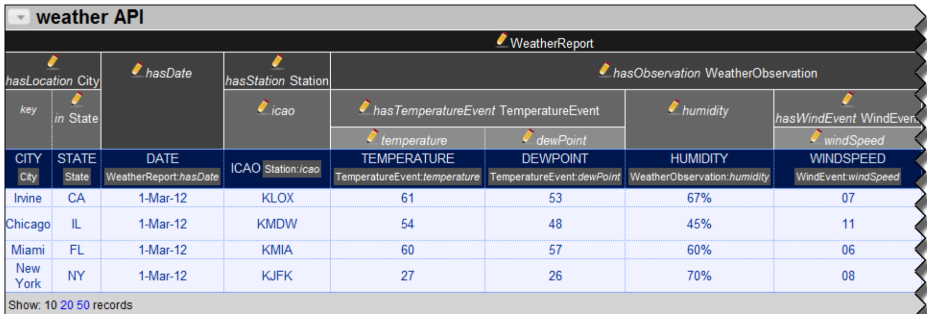
2.3 Specifying the Relationships

Annotating the service input and output by mapping them to the concepts of the ontology does not fully characterize the functionality of the API. We still need to know the relations between different columns, especially between the input and the output columns.

The next step in modeling an API is to represent the relationships between the semantic types in terms of the properties defined in the ontology. Karma first constructs a graph where nodes are the classes in the ontology and the edges correspond to ontology properties that relate them [4]. Karma uses an approximate Steiner tree algorithm to compute the minimal tree in this graph that connects the semantic types [12]. The minimal tree corresponds to the most succinct model that relates all the columns in a source, and this is a good starting point for refining the model. It is possible that multiple minimal trees exist or the minimum model is not the right model of the service. To resolve such cases, Karma provides a user interface to select the desired relationships. Karma represents the extracted relationships in a tree-based structure at the top of the service worksheet. Figure 3 shows the visualization of the weather API model in Karma.

3 Integrating Data and Services

In this part, we discuss how API models help Karma to integrate data and services. Suppose that the user imports into Karma a table of people profiles.

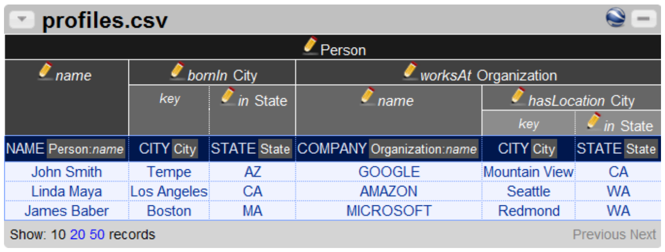


weather API

WeatherReport							
hasLocation City		*hasDate*	*hasStation* Station	*hasObservation* WeatherObservation			
key	*in* State		*icao*	*hasTemperatureEvent* TemperatureEvent		*humidity*	*hasWindEvent* WindEven
				temperature	*dewPoint*		*windSpeed*
CITY City	STATE State	DATE WeatherReport:*hasDate*	ICAO Station:*icao*	TEMPERATURE TemperatureEvent:*temperature*	DEWPOINT TemperatureEvent:*dewPoint*	HUMIDITY WeatherObservation:*humidity*	WINDSPEED WindEvent:*windSpeed*
Irvine	CA	1-Mar-12	KLOX	61	53	67%	07
Chicago	IL	1-Mar-12	KMDW	54	48	45%	11
Miami	FL	1-Mar-12	KMIA	60	57	60%	06
New York	NY	1-Mar-12	KJFK	27	26	70%	08

Show: 10 20 50 records

Fig. 3. Karma screen showing the weather API model.



profiles.csv

Person					
name	*bornIn* City		*worksAt* Organization		
	key	*in* State	*name*	*hasLocation* City	
				key	*in* State
NAME Person:*name*	CITY City	STATE State	COMPANY Organization:*name*	CITY City	STATE State
John Smith	Tempe	AZ	GOOGLE	Mountain View	CA
Linda Maya	Los Angeles	CA	AMAZON	Seattle	WA
James Baber	Boston	MA	MICROSOFT	Redmond	WA

Show: 10 20 50 records Previous Next

Fig. 4. Karma screenshot after importing the profiles data by the user.

Every profile includes the person name, the place where he or she was born, the organization where he or she currently works, and the address of the organization in terms of city and state. Once the data is imported, Karma builds a semantic model of this data source. Figure 4 shows the model that Karma builds for this source.

Exploiting both source and service models, Karma provides support to achieve the goals we described in Section 1 for service modeling. Karma knows that the source has some columns that are mapped to `City` and `State`. Using the service models, Karma would know that there is an API that takes two inputs with the `City` and `State` types. Thus, it can suggest to the user that the weather API can be invoked on this source. Karma can also show the user a list of additional attributes it can join with the source. For example, Karma would be able to tell users that they can augment the person dataset with the current weather conditions for the work location.

It is important to note that declaring the relationships among the columns provides richer semantics than just annotating the columns. For instance, the profile source has two CITY and two STATE columns. In the weather API model that Karma has already built, the semantic types that are assigned to the columns CITY and STATE are related to each other by the `in` relationship in the ontology. This knowledge enables Karma to select meaningful combinations of the data source columns to invoke a service (e.g., the system would never attempt to invoke the weather service with the city someone is born in and the

```
...
:WeatherAPI a msm:Service;
    msm:hasOperation :getWeather;
    hrests:hasAddress "http://isi.edu/integration/services/weather?"^^
                        hrests:URITemplate.
:getWeather a msm:Operation;
    msm:hasInput :getWeatherInput;
    hrests:hasMethod "GET";
    hrests:hasAddress "city={p1}&state={p2}"^^hrests:URITemplate.
:getWeatherInput a msm:MessageContent;
    msm:hasPart :part1, :part2.
:part1 a msm:MessagePart;
    sawsdl:modelReference places:City
    hrests:isGroundedIn "p1"^^rdf:PlainLiteral
:part2 a msm:MessagePart;
    sawsdl:modelReference places:State
    hrests:isGroundedIn "p2"^^rdf:PlainLiteral
...
```

Fig. 5. The weather API described with MSM ontology.

state that they work in). Therefore, specifying the relationships in the service model helps Karma to identify the appropriate inputs for API invocation.

4 Publishing Service Models as Linked APIs

In section 1 we introduced two known studies related to linked services. The first one uses a simple RDF ontology, called Minimal Service Model (MSM), to annotate the service and publish it as Linked Data. The second one, called Linked Open Services (LOS), describes the API input and output as SPARQL graph patterns. Here, we show how Karma is able to generate both of these service descriptions.

An MSM description of an API includes its address, the HTTP method, the input and output parameters, and their mappings to the concepts of the ontologies. MSM uses the `modelReference` property of the SAWSDL vocabulary [2] to annotate service parameters. Thus, even without specifying the relationships between API parameters, the Karma models include the information necessary to generate a MSM description that can be used by clients to invoke the API. If we are dealing with RDF data as input and output, it is necessary to generate the lowering and lifting schema as well. Karma already has the ability to import JSON and XML sources, and it can use its models to generate RDF. We believe that similar techniques will enable Karma to generate such schemas for API models built in Karma. Figure 5 is the MSM description of our weather example without lifting and lowering. Due to space limitations, we removed the namespaces and only the input model is shown.

LOS uses SPARQL graph patterns to represent the inputs and outputs of a service. Graph patterns provide support for representing the relationships among service parameters. Figure 6 illustrates the graph patterns generated for the

Input Pattern

```
?city a places:City; places:in ?state. ?state a places:State
```

Output Pattern

```
?report a weather:WeatherReport.
?report eg:hasLocation ?city.
    ?city a places:City; places:in ?state.
    ?state a places:State.
?report eg:hasDate ?date.
?report weather:hasObservation ?observation.
    ?observation a weather:WeatherObservation.
    ?observation weather:hasTemperatureEvent ?tempEvent.
       ?tempEvent a weather:TemperatureEvent.
       ?tempEvent weather:temperature ?temp.
       ?tempEvent weather:dewPoint ?dew.
    ?observation weather:hasWindEvent ?windEvent.
       ?windEvent a weather:WindEvent.
       ?windEvent weather:windSpeed ?windspeed.
    ?observation weather:humidity ?humidity.
    ?observation weather:description ?skycondition.
```

Fig. 6. Input and output graph patterns for the weather API.

weather API. The models that Karma builds also contain the relationships between inputs and outputs, expressed using known vocabularies from the Linked Data cloud. Karma can currently export them as a datalog specification and exporting them as graph patterns would be straightforward. To wrap the APIs that do not understand RDF, we have to add lowering and lifting schemas. We are working to add this functionality to Karma. This would enable Karma to publish LOS descriptions so that other people can invoke them on top of their RDF data.

5 Discussion

This paper presents our approach to interactively build semantic models of Web APIs. The key contribution of this work is that rather than manually enriching the traditional APIs with semantics, we use a semi-automatic approach to model a service. These models allow the system to help the user in service discovery, invocation, and composition. We are working to apply our modeling approach to the large number of Web services available. We are mining the Web for examples of service invocations in documentation pages, blogs and forums to automatically construct datasets of sample data to invoke services.

One of the other directions for future work is to extend our approach to model RESTful APIs. Extracting the input parameters from a RESTful API is not as straightforward as a Web API. Suppose that the corresponding REST API of the Web API in Section 2 is `http://example.com/state/CA/city/LA`. From just the URL, we do not know the actual input parameters. However, collecting more samples of this API invocation and analyzing the variable parts of the URLs will enable us to automatically extract the inputs of RESTful APIs.

References

1. Domingue, J., Pedrinaci, C., Maleshkova, M., Norton, B., Krummenacher, R.: Fostering a relationship between linked data and the internet of services. In: The Future Internet: Future Internet Assembly 2011: Achievements and Technological Promises, pp. 351–364. No. 6656 in Lecture Notes in Computer Science, Springer-Verlag (2011)
2. Farrell, J., Lausen, H.: Semantic annotations for wsdl and xml schema (Aug 2007), `http://www.w3.org/TR/sawsdl/`, w3C Recommendation
3. Goel, A., Knoblock, C.A., Lerman, K.: Using conditional random fields to exploit token structure and labels for accurate semantic annotation. In: Proceedings of AAAI-11 (2011)
4. Knoblock, C., Szekely, P., Ambite, J.L., Goel, A., Gupta, S., Lerman, K., Muslea, M., Taheriyan, M., Mallick, P.: Semi-automatically mapping structured sources into the semantic web. In: 9th Extended Semantic Web Conference 2012 (ESWC 2012) (2012)
5. Krummenacher, R., Norton, B., Marte, A.: Towards linked open services and processes. In: FIS'10. pp. 68–77 (2010)
6. Maleshkova, M., Pedrinaci, C., Domingue, J.: Investigating web apis on the world wide web. In: The 8th IEEE European Conference on Web Services (ECOWS 2010) (2010)
7. Norton, B., Krummenacher, R.: Consuming dynamic linked data. In: First International Workshop on Consuming Linked Data (COLD2010) (2010)
8. Pedrinaci, C., Domingue, J.: Toward the next wave of services: Linked services for the web of data. Journal of Universal Computer Science 16(13), 1694–1719 (jul 2010)
9. Pedrinaci, C., Liu, D., Maleshkova, M., Lambert, D., Kopecky, J., Domingue, J.: iserve: a linked services publishing platform. In: Ontology Repositories and Editors for the Semantic Web Workshop at The 7th Extended Semantic Web. vol. 596 (June 2010)
10. Speiser, S., Harth, A.: Towards linked data services. In: 9th International Semantic Web Conference (ISWC2010) (November 2010)
11. Tuchinda, R., Knoblock, C.A., Szekely, P.: Building mashups by demonstration. ACM Transactions on the Web (TWEB) 5(3) (2011)
12. Winter, P.: Steiner problem in networks - a survey. Networks 17, 129–167 (1987)