Context-Aware Automated Workflow Composition for Interactive Data Exploration

Diego Serrano(✉)

University of Alberta, Alberta, Canada
serranos@ualberta.ca

Abstract. Nowadays, the Web of Data contains a myriad of structured information sources on a large number of domains. Nevertheless, most of the information is available through Web APIs that act as isolated silos of data that cannot interoperate automatically with other resources and services on the Web. My dissertation aims at synthesizing semantic web technologies over Web APIs, in order to combine the easy data integration techniques offered by semantic web, with the flexibility and availability of web services. This paper discusses the two main aspects of the envisioned thesis: (a) a description language to semantically describe functional and non-functional components of web services, and the relationships among those components, and (b) a middleware that plans composition chains, based on user's specifications, optimizing their trade-offs.

Keywords: Semantic web · Web services · Data integration · Linked data

1 Introduction

In the early 2000s, Berners-Lee *et al.* [2] envisioned a transition from an Internet of loosely interlinked text documents, designed for human consumption, to the *Semantic Web*, a thoroughly described and tightly interlinked "Web of Data", intended for automatic machine processing. The most pragmatic effort to realize the Semantic Web vision has focused on publishing interlinked datasets, in what is called the Linking Open Data (LOD) project. The project integrates multiple databases that have been translated into RDF, using a mixture of common vocabularies and terms specific to the data sources; these datasets are interconnected through the use of URIs and equivalence relations to external databases. Those datasets are typically available through SPARQL endpoints, but, in general, cross-database SPARQL queries tend to be of high complexity, since their formalization strongly depend on the ontological structure of the RDF store model and the relationships among entities, which usually has considerable variations among datasets. Therefore, the complexity and poor performance of accessing LOD datasets has limited their usage in real-time scenarios.

K. Drira et al. (Eds.): ICSOC 2016 Workshops, LNCS 10380, pp. 152–157, 2017.
https://doi.org/10.1007/978-3-319-68136-8_16

Although, the LOD project has proven that, in principle, the linked-data approach is effective for integrating data from a large number of sources, nowadays, a substantial amount of Web data is exchanged through web services that expose data in formats such as JSON or XML. Those structured responses offer a common syntactic format, but they lack necessary semantics to interpret and interlink the content of the documents. In order to address this problem, the services have to include semantic annotations, as prescribed by the so-called *Semantic Web Services* (SWS).

The SWS approach attempts to describe services using domain ontologies, in order to enable automatic discovery, execution, and composition. Many SWS definitions have been proposed, such as WSMO [4], and SAWSDL [6], but few implementations of real semantic services have been produced, and the adoption of SWS in practice has fallen short. The main reason, suggested by Tosi *et al.* [8], is the focus of the research community in the definition of new ontologies and tools, disregarding real implementations of SWS, which has kept the discussion at an abstract level.

The objective of this thesis is to define a formalism that allows the creation and context-aware automatic exploration of semantically connected Web APIs, for different domains, taking into consideration functional and non-functional specifications of the resources, such as trustworthiness, authentication mechanisms and availability. Then, the middleware can specify execution plans for queries, that link and compose responses from independently deployed services, and optimize the trade-offs of data quality and performance. Additionally, this thesis will address service engineering approaches, supported by specific tools, to simplify the creation of SWS and interactive exploration of their data sets.

2 Composing and Integrating Web Services

In this section, we motivate our formalism in the context of related approaches, and we illustrate the functionalities of our middleware through an example.

2.1 Specification of Services and Their Implementation Contexts

Most of the current SWS formalizations follow a bottom-up approach, for example SAWSDL and WSMO-Lite [10]. In bottom-up approaches, Web services standards, such as WSDL, are extended by adding light-weight semantic annotations, which means that service providers need to produce the semantic and syntactic definitions. On the other hand, top-down approaches decouple the semantic description of the service from its syntactic description, on the principle that the semantics should not be influenced by implementation details. This approach was explored in classical approaches, such as WSMO, which provided a fully-fledged framework to define ontologies, goals, mediators and web services.

In our approach, we aim to combine the best of both approaches: the simplicity of the descriptions in the bottom-up approaches, to encourage its adoption, and the low coupling of top-down approaches to gain independence from the

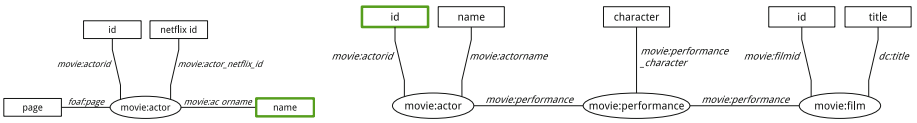
service providers, that allows us to reach a critical mass of service descriptions through crowdsourcing. In addition, one of the main guidelines in our design was to reuse and integrate existing formalisms into a simple common model, that supports publication and discovery. We introduce an RDF(S) integration ontology based on the Minimal Service Model (MSM) [5], that captures the maximum common denominator between existing conceptual models for services, enabling the representation of the semantics of services, including authentication mechanisms, provenance, quality of service metrics, and relationships among inputs and outputs.

The ontology defines a set of *Service* elements, which have a number of associated *Operation* elements. Operations, in turn, have a *Graph*, which is a collection of *Resource* triples that represent the underlying data schema of the service. The Operations also have links to input and output elements within such graph. The input elements may be defined as *required* or *optional*, and *full* or *partial*, for the cases when the input is used in partial match services, such as search functionalities. The outputs define a relation *responsePath*, that represents an XPath-like expression that is used in the lifting process, and avoids the use of other transformation technologies, such as XSLT, creating a self-contained semantic service description.

The ontology builds upon existing vocabularies. *hRESTS* [5], which is also used in MSM, is used to provide basic support for capturing grounding information necessary for Web APIs. *Web Api Authentication* [7] is used to annotate information about authentication information on top of Web API descriptions. The *Dataset Quality Vocabulary* [3] provides a lightweight vocabulary to describe functional and non-functional aspects of the service. And finally, *RDF Graph Patterns and Templates* [1] defines the terms used to describe the data schema graph patterns. The resources in the graph may contain other domain-specific ontologies and vocabularies to associate their classes and properties.

We illustrate our service modelling approach with an example based on complementary services. We consider two operations: *search actors*, and *get movie credits*. The former, searches for actors by name, and the latter, gets the list of movies where an actor, specified by his id, played a role. To keep the example concise, we only consider the graphs shown in Fig. 1 to represent the services, which were annotated using the Linked Movie Database Ontology. Traditionally, semantic descriptions consider ontological annotations on inputs and outputs to carry reference annotations between the service description and the domain ontology, for example, stating that the input of *get movie credits* is a person (or an attribute of it), and its output is a list of movies (or attributes of it). However, those annotations do not capture the specific relation between the two entities, leaving the description open to different interpretations, for instance, with other relations such as *produced* or *directed*.

In order to overcome the aforementioned limitation, we define a service operation as a 6-tuple $(E, G_s, I_{G_s}, O_{G_s}, A, Q)$ where E denotes the endpoint and other grounding parameters of the service, such as the URL and the HTTP method. G_s defines a graph representation of the service, as a finite collection of



(a) Search actors (b) Get movie credits

Fig. 1. Graph representations of the data in the example services. The green box denotes the attribute used as input. (Color figure online)

triple patterns. A triple pattern $t_s = (s, p, o)$ is a tuple composed of a subject, a predicate and an object, where $t_s \in (U \cup V) \times U \times (U \cup L \cup V)$, being U, L, and V disjoint infinite sets of URIs, literals and variables. I_{G_s} and O_{G_s} define the inputs and outputs of the service, respectively, pointing to subjects or objects in graph G_s. Finally, A denotes the authentication mechanism used by the operation, and Q defines the quality of service attributes associated to the particular operation or to the service in general.

2.2 Context-Aware Discovery and Composition

In order to query the services, we represent the queries as a conjunctive SPARQL query. More formally, a query consists of a triple (G_q, I_{G_q}, O_{G_q}), where G_q denotes a graph composed of triples, where each triple $t_q = (s, p, o)$ and $t_q \in (U \cup V) \times U \times (U \cup L \cup V)$. In turn, I_{G_q} and O_{G_q} represent the inputs and outputs, respectively. In our example, if we want to know the name of the characters that the actor *Brad Pitt* has played, we would write the following SPARQL query:

```
SELECT ?character
WHERE {
  ?person a movie:actor ;
    movie:actorname"Brad Pitt" ;
    movie:performance ?performance .
  ?performance movie:performance_character ?character
}
```

From the query, we can extract inputs and outputs. Outputs are simply the projected variables of the data query, `?character` in our example; and inputs are the bound values that appear in the triples of the query, in the `WHERE` clause, like the name `"Brad Pitt"` and the type `movie:actor` in the example. Then, the query answering problem is transformed into a subgraph isomorphism problem, in which the query graph G_q and the graph formed by all the service graphs G_S are given as input, and we must determine, first, whether G_S contains a subgraph that is isomorphic to G_q, and then extract the individual service graphs in the isomorphic graph. However, at the size of the Web, and even with a moderate load, this problem can be untractable for ad-hoc queries. In our approach, in order to discover services dynamically, we start by extracting elements from G_q that can be used as input parameters of a service request. Then, we use the input elements and authentication credentials as a filter, to consider only the

services that can be invoked with the current input parameters and authentication credentials. In our example, the only service in consideration is *search persons*, since it only needs a name as an input, represented by `"Brad Pitt"` in the query.

When a query is submitted to the system, the query processor has to find a isomorphic subgraph from the set of service graphs. If such graph is not found, then it has to search a sequence of service graphs that can be composed together to fulfill the goal represented by the query. In our iterative approach, if a graph G_s can be invoked and matches partially with graph G_q, then we use the outputs of G_s as inputs of G_q in the next iteration. The algorithm stops when a sequence of service graphs $\{G_{s_1}, ..., G_{s_n}\}$ has completely covered the graph G_q, or when we reach a fixed number of iterations. In our example, the graph for *search persons* only covers the triples for the name and type of the person, leaving the performance triples uncovered. Then, in the next iteration, we can use the actor identifiers, and webpage produced as outputs from *search persons*. In this second iteration, we can use the identifier of the person to invoke the service *get movie credits*, and cover the remaining triples of the query.

The algorithm for subgraph isomorphism used in our approach, is adapted from Ullman's algorithm [9], checking recursively that the adjacent nodes and edges are equivalent. The equivalence relation considers ontological equivalence (through relations such as `owl:sameAs`), and subsumption relationships of resources and properties in the ontology.

After all the possible composition chains have been discovered, the system ranks them, according to user's preferences and non-functional properties criteria like response time, and availability. In this process, the non-functional properties values can be obtained by statistics collected during service invocations, or gathered from service descriptions.

In our system, we provide a web-based data analysis tool that empowers regular web users to explore and analyze the Linked Data produced by our integration endpoint. Currently, the system supports three different types of visualizations: (a) a tabular view, provides a familiar visualization style to read and interpret complex JSON files; (b) a force-directed graph, that provides a natural and interactive visualization, where the user can follow how the graph evolves and how clusters are formed, and (c) a JSON document, that provides support for machine-to-machine communication for third-party applications. Users can interact with the graph by expanding nodes. When the user request the expansion of the node, a SPARQL is created using the URIs and literals directly attached to the node as input graph patterns, and using the syntax `SELECT *`, which in our system is an abbreviation that selects all of the variables that are declared in the service definition.

3 Research Plan Overview

In the future, we plan to support semi-automatic description of services, based on a collection of sample invocation URLs provided by users. The system is able

to extract input and output parameters from the example service invocations, and then, use the datatypes, names of the parameters, and popular ontologies to suggest mappings between a mediated schema and the service. In addition, we intend to provide different kinds of visualizations, depending on the nature of the data returned, such as maps, temporal charts, and different layouts for graphs and tables. In the long term, we envision a middleware that will enable system developers and end-users to transparently and efficiently access heterogeneous resources, supported through our middleware.

The evaluation strategy is divided in two phases, that involve controlled empirical studies with developers using the tools we plan to develop. First, we will demonstrate how effectively the proposed approach can be integrated into current Web APIs, including metrics of the cognitive difficulty faced by developers, and performance exhibited by the mediator system. Second, we will evaluate the exploration and visualization tools through usability assessments involving the presentation of the data, interaction with the data, the perceived cognitive offload, and the usability of the data itself.

Acknowledgement. I would like to thank my supervisor, Prof. Eleni Stroulia for her helpful suggestions. This work was supported by the KMTI NSERC Strategic project.

References

1. RDF graph patterns and templates. http://vocab.org/riro/gpt. Accessed 18 July 2016
2. Berners-Lee, T., Hendler, J., Lassila, O., et al.: The semantic web. Sci. Am. **284**(5), 28–37 (2001)
3. Debattista, J., Lange, C., Auer, S.: daQ, an ontology for dataset quality information. In: LDOW (2014)
4. Feier, C., Polleres, A., Dumitru, R., Domingue, J., Stollberg, M., Fensel, D.: Towards intelligent web services: the web service modeling ontology (WSMO) (2005)
5. Kopeckỳ, J., Gomadam, K., Vitvar, T.: hRESTS: an HTML microformat for describing Restful web services. In: IEEE/WIC/ACM International Conference on Web Intelligence (WI 2008), vol. 1, pp. 619–625. IEEE (2008)
6. Kopecky, J., Vitvar, T., Bournez, C., Farrell, J.: SAWSDL: semantic annotations for WSDL and XML schema. Internet Comput. 60–67 (2007). IEEE
7. Maleshkova, M., Pedrinaci, C., Domingue, J., Alvaro, G., Martinez, I.: Using semantics for automating the authentication of web APIs. In: Patel-Schneider, P.F., et al. (eds.) ISWC 2010. LNCS, vol. 6496, pp. 534–549. Springer, Heidelberg (2010). doi:10.1007/978-3-642-17746-0_34
8. Tosi, D., Morasca, S.: Supporting the semi-automatic semantic annotation of web services: a systematic literature review. Inf. Softw. Technol. **61**, 16–32 (2015)
9. Ullmann, J.R.: An algorithm for subgraph isomorphism. J. ACM (JACM) **23**(1), 31–42 (1976)
10. Vitvar, T., Kopecký, J., Viskova, J., Fensel, D.: WSMO-lite annotations for web services. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 674–689. Springer, Heidelberg (2008). doi:10.1007/978-3-540-68234-9_49