2017 IEEE 24th International Conference on Web Services

Linked REST APIs: A Middleware for Semantic REST API Integration

Diego Serrano, Eleni Stroulia
Department of Computing Science
University of Alberta
Edmonton, Canada
{*serranos,stroulia*}*@ualberta.ca*

Diana Lau, Tinny Ng
Center for Advanced Studies
IBM Canada Lab
Toronto, Canada
{*dhmlau,tng*}*@ca.ibm.com*

***Abstract*—Over the last decade, an exponentially increasing number of REST services have been providing a simple and straightforward syntax for accessing rich data resources. To use these services, however, developers have to understand "information-use contracts" specified in natural language, and, to build applications that benefit from multiple existing services they have to map the underlying resource schemas in their code. This process is difficult and error-prone, especially as the number and overlap of the underlying services increases, and the mappings become opaque, difficult to maintain, and practically impossible to reuse. The more recent advent of the Linked Data formalisms can offer a solution to the challenge.**

In this paper, we propose a conceptual framework for REST-service integration based on Linked Data models. In this framework, the data exposed by REST services is mapped to Linked Data schemas; based on these descriptions, we have developed a middleware that can automatically compose API calls to respond to data queries (in SPARQL). Furthermore, we have developed a RDF model for characterizing the access-control protocols of these APIs and the quality of the data they expose, so that our middleware can develop "legal" compositions with desired qualities. We report our experience with the implementation of a prototype that demonstrates the usefulness of our framework in the context of a research-data management application.

***Keywords*-Data integration; Linked Data; REST APIs**

I. Introduction

Over the past 30 years, the World Wide Web revolutionized the way people access data and services. Every day the number of providers increases, as does the volume of accessible data, the variety of available services, and their overlap. And as the available content increases, efforts to enable machine-understandable integration abound. It is impractical to review these efforts in anything but a very eclectic manner, which is why, in this paper, we reflect on web-services formalisms and semantic data ontologies, in order to establish the background framing our work.

Today, a substantial amount of web data is exchanged through Web APIs that expose data in convenient structured formats, such as JSON or XML. Nearly all the top 100 websites from Alexa[1] provide their own APIs, which is an indicator of the degree to which web services have been adopted as the de-facto mechanism for interacting with external resources. In the majority of cases, the structured responses of these APIs follow a common syntactic format, as dictated by the REST style. However, these responses are typically not grounded in semantics, which is a necessary prerequisite for automatically interpreting and interlinking the content of the exchanged content. Thus, developing applications that rely on Web APIs still requires a considerable effort by application developers. They first need to obtain the necessary credentials from the providers to access the APIs. Next, they have to write code for invoking eachindividual API, which implies the need to understand the nature of the data these APIs consume and produce, in order to compose compatible APIs and integrate their data by mapping the different implied schemas.

A long line of semantic representations have been motivated by the need to simplify and automate this development effort. A relatively recent development is the Linked-Data effort, which has led to the development of knowledge-representation languages for complex domain models, and reasoning techniques for inferring new knowledge. Currently, the Linking Open Data (LOD) project has more than 1,000 interlinked datasets, spanning a number of diverse thematic areas such as government, media, and life sciences, among others [1]. In parallel, numerous[2] controlled vocabularies have been established as the preferred conceptual-modeling methodology for data integration, due to their flexibility and evolvability. Some prominent examples include schema.org, and Dublin Core.

The key intuition motivating our work is that the alignment of these two widely adopted technologies, REST APIs and Linked-Data vocabularies, can provide the level of interoperability required *for supporting automated composition of web APIs, in practice*. The emergence of REST APIs as the de-facto standard to securely exchange data on the web along with the increasingly popular Linked-Data repositories enable the automated semantic integration of data within a specific domain, which is the objective of our LRA framework.

More specifically, in this paper, we propose a methodology for semantically annotating REST APIs with Linked-Data ontologies. Accordingly, we have developed a mid-

[1] *Alexa* (http://www.alexa.com/topsites) is a company that provides information about web traffic data.

[2] *LOV* (https://lov.okfn.org/) lists 588 vocabularies.

978-1-5386-0752-7/17 $31.00 © 2017 IEEE
DOI 10.1109/ICWS.2017.26

dleware for automatically specifying execution plans for SPARQL queries that invoke these APIs, link and compose their responses, while taking into account their access-control constraints and the trade-offs of data quality and performance.

The remainder of this paper is organized as follows. Section II gives an overview of the evolution of semantic description of web services, in parallel with the development of integration systems based on web-services. Then, in order to understand the challenges of data integration based on web services, in Section III, we present an example that illustrates how different data sources may converge in an integrated framework. Section IV presents the proposed semantic description for Linked REST APIs, discussing its design guidelines and improvements over previous approaches. Section V introduces the sequence of steps that support the integration process. In Section VI, we present the implementation of a prototype for a research-data management application, that demonstrates the usefulness of our framework. And the final section concludes the article.

II. Related Work

The original web-services activity advocated for SOAP-based services, described through WSDL specifications in terms of their operations and their input and output types, and explicitly composed in BPEL processes. A variety of development tools were produced for this stack of standards, and a wave of metadata proposals were put forward to semantically describe these web services, such as OWL-S [2], WSMO [3], WSDL-S [4] (W3C submissions), and SAWSDL [5] (a W3C recommendation). Numerous related proposals followed SAWSDL, such as hRESTS & MicroWSMO [6], WSMO-Lite [7], and the Minimal Service Model (MSM) [8], which proposed a composition of concepts shared by SAWSDL, MicroWSMO, WSMO-Lite, and hRESTS. None of these proposals were supported by working middleware with enough traction to drive the development of a community, which led to their abandonment. A notable example of a framework is the METEOR-S [9] system, which covered the complete service-integration lifecycle, including semi-automatic annotation, discovery, and composition of web services, relying on WSDL-S. Some more recent work [10], [11] has focused on enabling developers to semantically annotate web services, and use the annotations to assist the users in the manual creation of executable process models, based on workflow templates.

Closer to our work, is the work of Sbodio *et al*. [12], who use SPARQL to represent preconditions and postconditions of web-service operations. This work was has only been evaluated on a synthetic data set and is fundamentally limited in that the SPARQL meta-data do not express non-functional aspects, such as access-control rules or quality-of-service requirements. Similarly, the Linked Data Services (LIDS) description [13] uses RDF and SPARQL graph patterns to describe the service inputs and outputs and to capture the relationships among the attributes. However, this meta-data uses a string encoding to describe the data relationships, which precludes the use of automated reasoners for validation and composition. More recently, Rodriguez *et al*. [23] proposed a framework for graph-based service composition focused on the semantic input-output parameter matching of services interfaces. Nevertheless, the composition algorithm may produce incorrect solutions, since the semantic descriptions do not capture the relations between input and output elements.

The ultimate goal that has been motivating all the above efforts is to provide rich and unambiguous representations of heterogeneous APIs, that machines can reason about in sufficient depth, for automatically discovering, composing, and invoking them. Despite the preponderance of related-research literature, practical semantic integration of services remains an elusive goal. In practice, only very few experimental services have such semantic descriptions. A key reason for the limited adoption is the fear of typical developers to use, seemingly complex, Semantic-Web technologies. The desire to mitigate this fear has driven semantic-annotator projects, but they also lack of broad adoption, due to the unpopularity of semantic web services.

REST services have now become more common in practice, but there are no universally accepted standards for describing them. Some formats, such as OpenAPI, and RAML, have emerged to describe REST APIs, that were traditionally documented with natural language. Until now, these formats have been used, mainly, to automatically generate SDKs and to create documentation and API consoles for testing web services. Such formats include invocation details, like request methods, status codes, and input arguments, but completely ignore the underlying service semantics. Our work is motivated by our belief that the emergence of REST APIs as the preferred syntax for exchanging data on the web, and RDF, as the formalism for representing data semantics and linking data, provide a unique opportunity for building a practical, semantic service-integration methdology.

III. The Research-Evaluation Example

Let us demonstrate the need for automated service-integration systems with an example in the application domain of research-program evaluation. Key to a good evaluation is the consideration of several activities and outputs, as well as different evidence of impact. While many organizations, such as digital libraries and funding agencies, offer Web APIs that provide programmatic access to metadata of millions of publications and grants programs, each one of them offers only a fragmented view of a researcher's production. None of them contain all the publications of an individual author (publisher-specific repositories are unlikely to include publications in venues beyond the ones owned

Table I: Sample data for *Library1*

ID	ISBN	TITLE	AUTHORS	YEAR	PUBLICATION
10001	123-1-23-789012-3	Schema mediation	J. Smith M. Johnson A. Williams	2003	ICDE
10002	123-4-56-789012-3	Structured Data	M. Johnson	2010	WWW
10003	123-7-89-789012-3	Nonlinear Dimensionality	J. Morris M. Johnson E. Davis	2014	J-STARS
10004	123-4-56-789021-3	Object exchange	S. Walker J. Wood	1995	ICDE

Table II: Sample data for *Library2*

DOC_ID	NAME	AUTHORS	YEAR	CONFERENCE
123	Query Answering	Rachel Brown Michael Johnson	2001	The Very Large Databases Journal
124	Structured Data	Michael Johnson	2010	The World Wide Web Conference
245	Documents Clustering	Steve Parker James Wood	1998	The World Wide Web and Databases
345	Nonlinear Dimensionality	Joshua Morris Michael Johnson Emma Davis	2014	Journal on Selected Topics in Applied Earth...

by the publisher) nor do they contain all the information relevant to program budgets assigned to researchers.

To illustrate the challenges of integrating data from multiple Web APIs, we consider as an example two different publication sources *Library1* and *Library2*, each producing records associated with different publishers. The information managed by the two repositories is similar, but linking the two data sources provides broader coverage and more detailed information about publications, for example, *Library1* can include ISBNs, and *Library2* can complement the papers written by Michael Johnson, with paper 123, which does not exist in *Library1*.

In the absence of a semantic grounding of the data of the above resources, integrating this data is a complex task. For example, *Library1* models publication titles using *title* as the attribute name, while *Library2* uses *name* for the same piece of information. A prominent ontology that can represent semantic concepts in this domain is the *VIVO* ontology [14], a unified, formal, and explicit specification of information about researchers, organizations, and the scholarly activities, outputs, and relationships that link them together. Thus, a composition of such Web APIs will rely on explicit mappings (whether user-defined or automatically inferred) between the elements in the request and response elements, and the target concepts of a mediated schema, in this case the *VIVO* ontology.

In today's API economy, businesses seek flexible and on-demand integration with external systems that offer relevant data and can potentially create additional value for their own services. Then, once the services are described and mapped to the target ontology, the goal is to enable automatic discovery and composition, which could in turn reduce the time and manual effort to maintain code and integrate the logical connections among data sources. As a consequence, a Web API-based integration has four main challenges in answering a question like "Which are the titles of all the documents written by Michael Johnson?": (1) representing the data sources using a common model, (2) understanding the question, (3) identifying and composing potential data sources, and (4) obtaining and unifying the information itself. To address the first challenge, services need to be described in terms of their data semantics and their non-functional properties. For example, *Library2* must specify that it can provide information about the publications of an author, given the author's name. In Section IV, we propose a compact and comprehensive data model for describing Linked REST APIs (LRA), based on graph patterns. For the second challenge, the developers have to formulate the question in terms of the mediated schema used to describe the REST APIs. For example, if the query asks about publication titles, then the system should be able to recognize that *Library1* and *Library2* provide titles, although under different attribute names. In Section V-A, we discuss the most popular query languages and interfaces used to pose queries in data integration systems. In the third challenge, the middleware has to automatically transform the query, expressed in a declarative language, using domain terms, into a composition of invocations to existing web services that satisfies the query. In our example, the middleware searches for a matching service, or for a chain of composable services, that can provide titles of publications given an author name. Thus, the results of *Library1* and *Library2* can be used to create a consolidated answer. In Sections V-B and V-C, we present a novel methodology to automatically discover and compose APIs, based on subgraph isomorphism, which also takes into account security (Section V-D) and quality (Section V-E) restrictions defined by service providers and application developers. For the fourth challenge, the middleware invokes the services, incorporating inputs extracted from the initial query, and security credentials provided by the application developers. When the system receives the responses from the data sources, the middleware translates the data into its semantic representation, in a process known as *lifting*. In addition, as the data comes from multiple sources, the middleware proceeds to identify records with references to the same entity, across data sources. In Section V-F, we discuss our approaches to inject information to describe semantic mappings in service responses, and link responses from multiple providers.

In the rest of the paper, we will use a set of operations from each data source, exposing data through REST services. *Library1* has two operations: `searchAuthors` and `getPapers`. The `searchAuthors` operation receives as an input a person name, and returns a set of authors having that specific name, while the `getPapers` operation returns a list of academic papers created by an author, whose identifier is received as an input. And *Library2* has the operation `findAuthors`, which, given a name of an author, returns a response with information about the author and their associated documents.

IV. Description Language for Linked REST APIs

A key component in any integration system is the language in terms of which the define semantic mappings

among the underlying resources are expressed. In this section, we introduce the semantic model underlying our LRA framework, which uses an ontology-based integration approach, and abstracts the complexities and weaknesses originated by previous semantic web-services approaches.

In our approach, we introduce a vocabulary that enables the semantic representation of REST services, based on MSM, and the structural organization of OpenAPI and RAML, also including information about authentication mechanisms, quality, and relationships between inputs and outputs. The vocabulary defines a *Service* element that acts as the container of the resources and operations of a business unit. Each *Service* has a number of associated *Operations*, which, in turn, are associated with *Graph*s, that represent the underlying service data schema. The *Operations* include an attribute for the type of operation, based on Hydra [15], to indicate that an operation results in resources being read, created, deleted, or replaced. The *Operations* also have links to input and output elements within the associated graphs. 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 property *responsePath* that represents an XPath-like expression that is used in the lifting process. The rationale behind connecting inputs and outputs in a graph is that simple model references may lead to inconsistencies and ambiguities, and cannot always fully express the required semantics. For example, knowing that, in the `getPapers` operation of the *Library1* service, the input is a person identifier, and the output a set of academic articles, is not enough to identify the *creator* relationship, since there are many possible relations between person and papers, such as reviewer or editor.

In addition to the functional description of the service, most Web APIs have one or more mechanisms to secure data access, identify requests, and determine access level and data visibility. This issue has been identified by other approaches, like RAML's Security Schemes, OpenAPI's Security Definitions, and Web API Authentication ontology [16], which are used to annotate configuration information about authentication mechanisms on service descriptions. Finally, in the context of automated service composition, the inclusion of web services in a composition chain may depend on other non-functional properties, like latency, reliability, or subjective preference. Our LRA framework. describes quality attributes using the Dataset Quality Vocabulary [17].

Formally, a Linked REST API is specified as a set of operations, each defined by a 6-tuple $(E, G_s, I_{G_s}, O_{G_s}, A, Q)$. E denotes the grounding parameters of the service, such as the endpoint, the URL, and the HTTP request method. G_s defines a graph representation of the service, as a finite collection of triple patterns, $t_s = (s, p, o)$, composed of a subject, a predicate and an object. $t_s \in (U \cup V) \times U \times (U \cup L \cup V)$, where U, L, and V are the disjoint infinite sets

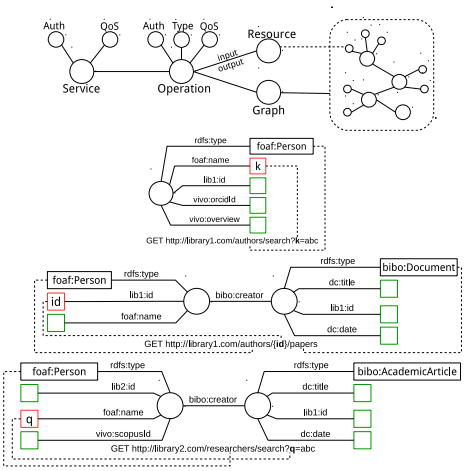

Figure 1: Data model and graph representations of the data provided by the web services in the example. Red indicates an input attribute; green indicates an output attribute. Dashed lines denote mappings between the REST API syntax and the Linked-Data RDF elements.

of URIs, literals and variables, respectively. I_{G_s} and O_{G_s} define the inputs and outputs of the service correspondingly, pointing to subjects or objects in the 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. Figure 1 shows a diagrammatic representation of the model for Linked Web APIs, and the graphs of the example services introduced in Section III for *Library1* and *Library2*.

V. The LRA Middleware

The main objective of the LRA middleware is to leverage the semantic descriptions of Linked REST APIs in order to automate the process of invoking and composing these APIs in software applications, and to reduce the manual work required by software developers. The LRA middleware assumes that developers will specify the data needs of their applications in SPARQL queries; in response, it provides a consolidated data graph (in RDF), produced through the composition of possibly multiple REST API chains, taking into account the application developer's credentials and quality requirements.

A. End-User Interaction

The LRA middleware provides an interface for users to submit their queries, described declaratively using SPARQL. This implies a new model for application development using REST APIs: developers, instead of writing code to formulate the HTTP requests corresponding to the invocations of the necessary REST APIs and to parse the returned data, will have to develop code that formulates SPARQL queries and parses the RDF graph resulting from the execution of the automatically invoked API chain. Despite the complexity of SPARQL query evaluation [18], the community appears to

have reached a consensus in favor of SPARQL for linked-data exploration. A SPARQL representation of the question "Which are the titles of all the documents written by Michael Johnson?", introduced in the motivating example, can be expressed in the following query.

```
SELECT ?title
WHERE { ?doc a bibo:Document ;
              dc:title ?title ;
              dc:creator ?author .
        ?author rdfs:label "Michael Johnson" }
```

B. Discovery

Our example service, `findAuthors` introduced in Section III and represented in Figure 1d, returns only academic articles from *Library2*, which is a subset of the data required by the example query. However, the concepts and properties used in the definition of the web service, i.e., `bibo:AcademicArticle` and `foaf:name`, do not exactly match those of the query, i.e., `bibo:Document` and `rdfs:label`, respectively. Our middleware, endowed with a registry of Linked REST APIs, can reason about the knowledge represented in the data query, and infer that (a) `bibo:Article` is a subclass of `bibo:Document` and (b) `foaf:name` is a subproperty of `rdfs:label`, thus discovering this web service as a candidate service for answering the question at hand. In effect, discovering such services enables the integration of the data they expose.

In order to discover services dynamically, the query is analyzed to extract elements that can be used as input and output parameters of a service request. Outputs are simply the projected variables of the data query; inputs are the bound values that appear in the triples of the data query, in the `WHERE` clause. Then, graph-match operations are only performed on services that use, at least, one of the inputs, and as a consequence, the search space is reduced. In addition, this step also considers services with entities related by super/sub-clasees and similar concepts (extracted through `owl:sameAs` and `owl:equivalentClass` links).

C. Composition

Given the services presented in our motivating example for *Library1*, there is no single service that can satisfy the example query. Nevertheless, if the data from `searchAuthors` is joined with the data returned from `getPapers`, the papers of a given author can be identified.

In other words, when a query graph is not contained entirely by any of the service graphs, it can still be answered through graph traversal, where each vertex represents a web service, and each edge corresponds to a data production-consumption relationship, where the outputs of the source service are consumed as inputs by the target service. The recursive process to compose the services dynamically, based on graph matching, is outlined in Algorithm 1. The process starts by extracting the triples, using Apache Jena, and identifying the potential inputs (line 6). Then, the first discovery step, aims at restricting the number of eventual pairwise graph comparisons. This step uses a filter on the predicates of the inputs, only considering service graphs that can use at least one of the inputs in the query graph (line 7). The next step uses subgraph isomorphism to determine whether the query graph is present within one of the service graphs, using an adaptation of Ullman's algorithm [19] that considers subsumption and equivalence relations (line 9). As long as the match covers new elements of the query graph, the process keeps exploring composition chains from the current service (line 10). If the addition of the current service covers all the required elements of the query graph, then the service is added to the composition chain, and the algorithm stops (lines 11-12). The algorithm is recursive, integrating the output of previous services in the composition chain, until no additional nodes are covered by the composed service chain (lines 14-18).

Algorithm 1: Composition through graph matching

Input: A query graph G_q, A finite set $G_S = \{G_{s_1}, G_{s_2}, ..., G_{s_n}\}$ of service graphs, a maximum exploration depth max, and a current depth $depth$.
Output: A list of composition chains L.

```
1  Function compose(G_q, G_S, max, depth)
2      L = ∅
3      if depth > max then
4          return L
5      while G_q is not covered do
6          in = getInputPredicates(G_q)
7          services = getCandidates(G_S, in)
8          for each s in services do
9              match = getMatch(G_q, s)
10             if match covers new elements of G_q then
11                 if match covers the rest of G_q then
12                     add s to L
13                 else
14                     G_q^+ = G_q + output of s
15                     next = compose(G_q^+, G_S, max, depth + 1)
16                     if next is not empty then
17                         chain s to services in next
18                         add s to L
19     return L
```

In the case of data-modification operations, namely insertion, update, and deletion, LRA adopts a conservative approach by restricting the composition chains to only one service, and thus, ensuring the consistency of the data, and delegating the transaction management to the local data sources.

D. Access Control

Security, namely protection against unauthorized access, has to be an essential part of service integration. Maleshkova *et al*. [16] conducted a survey of web-service authentication mechanisms, finding that more than 80% of the APIs have some kind of authentication, with the API key mechanism being the most popular. Then, the LRA middleware must ensure that only users with the proper credentials on a service can include its operations in their applications.

LRA service descriptions include annotations for authentication requirements, in such a way that the middleware knows which security credentials have to be provided by

the user, in order to invoke the web services. Then, the composition chains generated from the previous step can be filtered. For example, if *Library2* requires an API Key, but the user does not provide it, then, the system would not be able to create a valid request, therefore, invalidating all the composition chains where a service from *Library2* appears, at design time.

E. Quality

In LRA, information is obtained from numerous sources, each exhibiting a different level of quality. In the LOD project, for example, it is common to find inconsistencies among different sources. Similar problems with inconsistencies and variable data quality are expected when integrating web services. This leads to the general optimization problem of selecting a composition of web services for each query so that the overall quality and cost requirements are satisfied. Quality-aware composition has been addressed extensively by other authors [20], proposing several approaches based on exact algorithms or elaborate heuristics. Thus far, our middleware has adopted a simple approach, where developers or informations system managers declare desired levels of quality, for attributes such as response time and availability, using the Dataset Quality Vocabulary. These features can be used to rank and discard composition chains that, in conjunction with other services in the chain, fail to provide the quality specified by the users, or have a low rank. For example, if the user specifies a maximum latency of 1.5 seconds, but the composition of the services in *Library1* are of 1 second for `searchAuthors`, and 1 second for `getPapers`, then the composition chain should be discarded.

F. Data Lifting and Linkage

Once the invoked services return, the middleware 'lifts' the data contained in the response documents based on the service semantics. In some web-service formalizations, such as SAWSDL, lifting is performed by XSLT scripts, which is quite complex and never widely adopted. JSON-LD provides a good alternative to 'lift' service responses, but it is not popular among web-service implementations. The LRA middleware supports JSON-LD, but also uses XPath-like expressions to associate output resources of the Linked REST APIs graph to elements in the response of the service. For example, an output resource with a *responsePath* represented by `/author/name` would retrieve the values of the "name" object inside the "author" object, at the root of the response document. After the responses are lifted, the middleware merges the responses in a composition chain, by joining the responses based on the attribute values generated as output by an API invocation, and reusing them as inputs in subsequent service calls.

After mapping the data to a common ontology, the middleware cross-references equivalent entities across repositories. But beyond LOD repositories, and for most domains, there

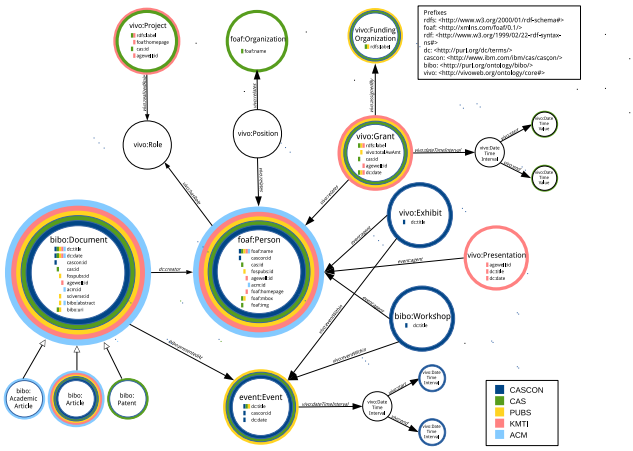

Figure 2: Graph representation of the five data sources used in the implementation of the middleware. The colors show the sources that can provide the particular data types or attributes.

are no `owl:sameAs` links, even when there is significant overlap in the repository contents, which, in effect renders these repositories into isolated silos. The problem is exacerbated by naming-convention differences, different representation formats, and errors, such as typos. The LRA middleware employs blank nodes to represent objects whose URI is not provided, and in the post-processing step, it uses the properties associated to the nodes in order to identify references to the same real-world entity across different data sources, and deduplicate the records in the generated graph, in a process known as record linkage. In our motivating example, for instance, the paper *1002* of *Library1* should be linked to the paper *124* of *Library2*, because their properties have a high similarity. To integrate the results across the composition chains, we implement a clustering algorithm, based on the attributes of entities, and a similarity function, depending on the data type of the values. From this last step, the response is consolidated and returned to the user.

VI. The Example Revisited

To evaluate the applicability of the middleware, we partnered with IBM Canada in a project designed to analyze the activities and productivity of their collaborations with members of academic institutions. Considering that no service can cover all the types of activities, in our IBM project, we used LRA to integrate services provided by five different data sources. *CASCON* contains data about interactions, represented by papers, workshop, exhibits, demos, and keynotes, as part of the Conference of the Centre for Advanced Studies on Collaborative Research, at IBM Canada. *CAS* is a proprietary system that provides data about collaborations (teams, grants, and research outcomes) with researchers from academic institutions, IBM researchers and technologists within the Centre for Advanced Studies (CAS) in IBM Toronto Laboratory in Canada. *PUBS* exposes a

dataset, semi-automatically extracted, from several sources, including Mendeley, Sciverse, and Canadian funding agencies. *KMTI* contains information about publications, grants, and awards for a specific project. *ACM* is a data source based on a data dump from the ACM Digital Library, accessible through a REST API that we developed ourselves.

The aforementioned web services expose operations for searching by name or title, and for obtaining detailed information about a repository entity, given its identifier. The operations in *CASCON* and *KMTI* APIs return JSON documents with embedded objects. For example, the *search author* operation returns authors, with embedded publications, in a similar way as other web services, such as the Digital Public Library of America (https://dp.la/info/developers/codex/), or Twitter (https://dev.twitter.com/rest/public). On the other hand, *CAS*, *PUBS*, and *ACM* provide similar operations as the other data sources, but in the search operations, the responses only contain information about the target entity, in the same way as Scopus (https://dev.elsevier.com/sc_apis.html) or DBLP(http://dblp.org/search/). Finally, the web services do not require authentication, except for *CAS*, which implements an API key security mechanism.

The five data sources are described as Linked REST APIs, using the model presented in Section IV, and mapping the concepts to the VIVO ontology. The descriptions were materialized using RDF. A diagrammatic representation of the data in each data source and its mapping to VIVO, can be found in Figure 2. The Linked REST API descriptions are stored as a queryable TDB dataset, using Apache Jena, and other information, such as user's security credentials is stored in a relational database.

For IBM Canada Lab, it is crucial to consider multiple types of research activities in the assessment of projects and collaborations with research institutions. One general method by which communicative activity may be explained and interpreted, is to consider the objects, agents, events, products, and contexts of such activity as entities to be counted, measured, or quantified [21]. In the following, we demonstrate how diverse types of techniques available to map scientific progress and influence were answered, considering multiple data sources, and using our middleware. Figure 3 shows the resulting composition chains for each evaluation technique, following the same color convention as in Figure 2.

Research Production Indicators Relevant indicators of research output are publications, including conference papers, journal articles, posters, patent applications and more. Given the variety of types of publications, we can take advantage of the flexibility of knowledge representation provided by ontologies, by using super-classes, in this case `bibo:Document`, which covers, for example, journal articles (`bibo:AcademicArticle`) from *ACM*, patents (`bibo:Patent`) from *CAS*, and general academic articles from all the data sources (`bibo:Article`). Query 1,

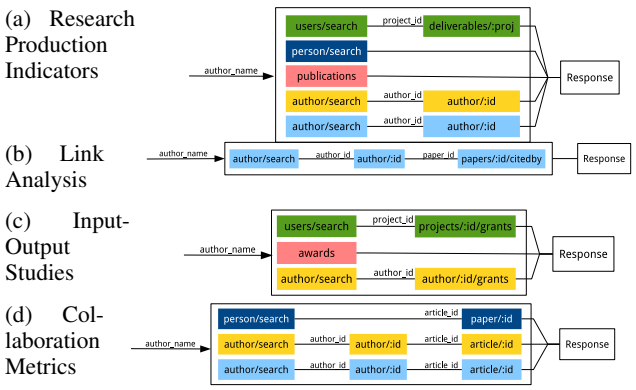

Figure 3: Composition chains for 'Research Evaluation' examples.

shown in Figure 4, retrieves the name of the author, title, and date of publications for *Michael Johnson*[3].

Link Analysis In evaluative link analysis, citation counts are used as indicators of quality, importance, and more generally impact of a researcher's output. The query shown in Query 2 in Figure 4 fetches the documents that cite any of the publications of *Michael Johnson*.

Input-Output Studies Some studies have attempted to correlate research expenditures with publication output, which have become recurrent in institutions evaluating researcher's performance. Query 3 shown in Figure 4 collects information about grants awarded to *Michael Johnson*, which are consolidated from data in *CAS*, *PUBS*, and *KMTI*.

Collaboration Metrics The study of collaboration, influenced by the work of Katz and Martin [22], has shown that research collaboration can bring co-authors greater research productivity and research impact. For that reason, measuring sociability, defined simply as the number of coauthors, is important for researcher's evaluation. Query 4 in Figure 4 extracts the name of the coauthors of *Michael Johnson*.

```
Query 1: SELECT ?name ?title ?date
         WHERE { ?author a foaf:Person ;
                   foaf:name ?name ; foaf:name "Michael Johnson" .
           ?doc a bibo:Document ; dc:creator ?author ;
                   dc:title ?title ; dc:date ?date }

Query 2: SELECT ?name ?title ?date
         WHERE { ?author a foaf:Person ;
                   foaf:name ?name ; foaf:name "Michael Johnson" .
           ?doc a bibo:Document ; dc:creator ?author ;
                   bibo:citedBy ?citing .
           ?citing a bibo:Document ; dc:title ?title ;
                   dc:date ?date }

Query 3: SELECT ?name ?title ?date ?amount
         WHERE { ?author a foaf:Person ;
                   foaf:name ?name ; foaf:name "Michael Johnson" .
           ?grant a vivo:Grant ; vivo:relates ?author ;
                   vivo:totalAwardAmount ?amount ;
                   dc:title ?title ; dc:date ?date }

Query 4: SELECT ?author2_name
         WHERE { ?author1 a foaf:Person ; foaf:name "Michael
                 Johnson" .
           ?doc a bibo:Document ; dc:creator ?author1 ;
                   dc:creator ?author2 .
           ?author2 a foaf:Person ; foaf:name ?author2_name }
```

Figure 4: Query for research production indicators

[3]'Michael Johnson' is a fictional name as are all the data reported in this section.

VII. Conclusions

In this paper, we described the LRA middleware for automatically discovering, composing and invoking REST APIs. The middleware relies on a semantic specification of the input-output functionalities of the APIs and their non-functional attributes, described in terms of Linked-Data. The LRA language for the semantic description of web services builds on lessons learned from a long history of semantic web services and service-description formats. We have chosen Linked-Data RDF ontologies for semantically annotating REST APIs, because of their wide adoption, and their flexibility to express functional relationships among elements in a web service. Accordingly, the LRA middleware presents a novel approach to answering SPARQL queries through a fully automatic process. This process, supported by the LRA middleware, involves the discovery of REST APIs relevant to the input SPARQL query, the composition of these APIs through their data production-consumption relationships, the execution of the composed plans through the invocation of the APIs, and the linking and merging of their responses. In addition, the middleware process takes into account the access-control constraints and the trade-offs of data quality and performance of the registered APIs.

We tested the usefulness of our middeleware on a research-program evaluation example, for the IBM Centre for Advanced Studies. Note that this is a real system, built based on our industrial-partner's requirements. This example involved (a) specifying a number of services, both proprietary and open-data based, in terms of the LRA model, and (b) demonstrating how the middleware responds to a number of queries required in our partner's application. Our scenario motivates our contribution and demonstrates the feasibility of our approach.

In the future, we plan to conduct larger-scale integrations and support semi-automatic description of services, based on a collection of sample invocation URLs provided by users. In addition, we are working on controlled empirical studies that involve developers using our middleware, in order to quantify the effectiveness of the proposed approach and the ease-of-use of our middleware tools.

References

[1] M. Schmachtenberg, C. Bizer, and H. Paulheim, "State of the lod cloud 2014," *University of Mannheim, Data and Web Science Group [online]*, vol. 30, 2014.

[2] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne et al., "Owl-s: Semantic markup for web services," *W3C member submission*, vol. 22, 2004.

[3] C. Feier, A. Polleres, R. Dumitru, J. Domingue, M. Stollberg, and D. Fensel, "Towards intelligent web services: The web service modeling ontology (wsmo)," 2005.

[4] R. Akkiraju, J. Farrell, J. A. Miller, M. Nagarajan, A. P. Sheth, and K. Verma, "Web service semantics-wsdl-s," 2005.

[5] J. Kopecky, T. Vitvar, C. Bournez, and J. Farrell, "Sawsdl: Semantic annotations for wsdl and xml schema," *Internet Computing*, IEEE, 2007.

[6] J. Kopecky, T. Vitvar, D. Fensel, and K. Gomadam, "hrests & microwsmo," *CMS WG Working Draft*, 2009.

[7] T. Vitvar, J. Kopecky, J. Viskova, and D. Fensel, *The Semantic Web: Research and Applications*. Springer Berlin Heidelberg, 2008, ch. WSMO-Lite Annotations for Web Services.

[8] C. Pedrinaci and J. Domingue, "Toward the next wave of services: Linked services for the web of data," *Journal of Universal Computer Science*, vol. 16, no. 13, jul 2010.

[9] A. A. Patil, S. A. Oundhakar, A. P. Sheth, and K. Verma, "Meteor-s web service annotation framework," in *13th international conference on World Wide Web*. ACM, 2004.

[10] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher et al., "The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud," *Nucleic acids research*, 2013.

[11] G. C. Hobold and F. Siqueira, "Discovery of semantic web services compositions based on sawsdl annotations," in *19th International Conference on Web Services (ICWS)*. IEEE, 2012.

[12] M. L. Sbodio, D. Martin, and C. Moulin, "Discovering semantic web services using sparql and intelligent agents," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 8, no. 4, 2010.

[13] S. Speiser and A. Harth, "Integrating linked data and services with linked data services," in *The Semantic Web: Research and Applications*. Springer, 2011.

[14] J. Corson-Rikert, S. Mitchell, B. Lowe, N. Rejack, Y. Ding, and C. Guo, "The vivo ontology," *Synthesis Lectures on Semantic Web: Theory and Technology*, 2012.

[15] M. Lanthaler and C. Gutl, "Hydra: A vocabulary for hypermedia-driven web apis." *LDOW*, vol. 996, 2013.

[16] M. Maleshkova, C. Pedrinaci, J. Domingue, G. Alvaro, and I. Martinez, "Using semantics for automating the authentication of web apis," in *International Semantic Web Conference*. Springer, 2010.

[17] J. Debattista, C. Lange, and S. Auer, "daq, an ontology for dataset quality information." in *LDOW*, 2014.

[18] M. Arenas and J. Perez, "Querying semantic web data with sparql," in *13th SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2011, pp. 305316.

[19] J. R. Ullmann, "An algorithm for subgraph isomorphism," *Journal of the ACM (JACM)*, vol. 23, no. 1, 1976.

[20] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient qos-aware service composition," in *18th International Conference on World Wide Web*. ACM, 2009.

[21] C. Borgman and J. Furner, "Scholarly communication and bibliometrics," *Annual Review of Information Science and Technology*, vol. 36, 2002.

[22] J. S. Katz and B. R. Martin, "What is research collaboration?," *Research policy*, vol. 26, no. 1, 1997.

[23] P. Rodriguez-Mier and C. Pedrinaci and M. Lama and M. Mucientes, "An Integrated Semantic Web Service Discovery and Composition Framework" *IEEE Transactions on Services Computing*, vol. 9, no. 4, 2016.