2012 IEEE 18th International Conference on Parallel and Distributed Systems

Enabling On-Demand Mashups of Open Data with Semantic Services

Yuzhang Feng, Anitha Veeramani, Rajaraman Kanagasabai
Institute for Infocomm Research, A*STAR, Singapore
Email: {yfeng, vanitha, kanagasa}@i2r.a-star.edu.sg

***Abstract*—Analogous to software-as-a-service (SaaS), platform-as-a-service (PaaS) and infrastructure-as-a-service (IaaS), data-as-a-service (DaaS) is used to provide data on demand to users over the Internet and is gaining popularity in the current cloud computing era. In particular, several Open Data initiatives have led to a number of data services in various formats becoming available online, and it remains a challenge to make full use of the data by transferring between and answering queries from different sources in a automatic, dynamic and meaningful manner. In this paper we propose a service-oriented, composition-based approach towards tackling the integration of open data services. We provide a formal, semantic-based modeling of data services and convert the integration problem into the service composition problem. Then the composition graph can be used to create the executable queries to the various data services. We illustrate our idea by using a case study in the real estate domain.**

I. Introduction

One of the distinguishing features of cloud computing is the provision of information technologies as a service over a network. Such information technologies can be in the forms of applications, operating systems or hardware through the notions of software-as-a-service, platform-as-a-service and infrastructure-as-a-service.

In recent years, the notion of data-as-a-service has become popular. Data-as-a-service, as the name suggests, is the cousin of the as-a-services and is used to provide *data* on demand to users regardless of geographic or organizational differences between the provider and consumer. Indeed we have seen effort by both government bodies and companies towards realizing DaaS. For example, a number of countries such as USA[1] and Singapore[2] have provided a portal to search and access publicly-available data published by the respective governments. For another example, with the introduction of data marketplaces such as Windows Azure, it has now become common for a company to provide some of its data to other organizations.

With a multitude of open data services available on the web, we identify a number of challenges that remain to be resolved. Firstly while the number of data services is drastically increasing, the consumer of the data services need to understand what the data services does so that the appropriate services can be selected. This means that the services need to have substantial semantic annotation for both human and machine to comprehend. The second challenge we identify is that, while there are many individual data services available, the links and relationships between services from different sources are hard to identify and exploit. Such relationships allow us to feed data from one service to another to obtain some information relevant to some consumer scenarios. For example in many cases an organization or individual may want to create a single view of an entity such as a citizen. Such information may be available from different data sources and these data sources may be provided in different formats by different organizations. It would be useful to the requester of the data to identify the relationship among these data services and combining them automatically and dynamically so that it is as if the requester is querying a single virtual data source. In this way the data services can be used in combination instead of as individual service to satisfy more consumer needs.

The approach we are taking to tackle these challenges in this paper can be divided into three steps. Firstly we model data sources as data service. Here we take the analogy from the web service community and define data services by its input, output, preconditions and effects. Secondly we semantically annotate the data service by using semantic web ontologies. This allows concepts for different data services to be semantically matchable. Thirdly we achieve semantic-based data integration by using the technique of service composition.

The remainder of the paper is organized as follows. In Section II, we describe how we formally model data services and the relationships between them by using the notions from the semantic web services. In Section III, we describe how to approach the data integration problem by using the service composition techniques. We illustrate our ideas by using a case study in Section IV. We discuss related work and their differences from our approach in Section V and conclude the paper in Section VI.

II. Modeling Data Services

Typically data services are used to provide a subset of a database by using a given value for a fixed field. For example the Ministry of Education may provide a data service which returns the set of primary schools which are within 1 kilometre of a given location in Singapore and their specific distances. In the following, we formalize the notion of data service by defining inputs, outputs, preconditions and effects (IOPE). Here we adhere to the approach taken by OWL-S [1], the semantic markup language of web services. In the following

[1] www.data.gov
[2] www.data.gov.sg

1521-9097/12 $26.00 © 2012 IEEE
DOI 10.1109/ICPADS.2012.122

we first briefly introduce OWL-S and then describe how we model data services by their IOPEs.

OWL-S is an ontology built on top of Web Ontology Language (OWL) [2] by the DARPA DAML program. The OWL-S ontology has been developed to enrich web services with semantics. The semantic markup of OWL-S enables the automated discovery, invocation, composition, interoperation and monitoring of Web services. This automation is achieved by providing a standard ontology for declaring and describing Web Services. Being an OWL ontology, OWL-S defines a set of essential vocabularies to describe the three components of a service, namely profile, model and grounding.

The service profile is used to describe what the service does. This information includes the service name and description, limitations on applicability and quality of service, publisher and contact information. The process model provides a process view on the service and describes how a client can interact with the service. This description includes the sets of inputs, outputs, pre-conditions and results of the service execution as well as the control flow of the service. The service grounding specifies the details that a client needs to interact with the service, such as communication protocols, message formats, port numbers, etc. The grounding can be thought of the concrete part of the Semantic Web service description, compared to the service profile and service model which both describe the service on an abstract level. The profile and process model are the focus of this paper.

Now we start the modeling of data services by formalizing the notions of inputs and outputs of a data service.

Definition 1 (Input and Output): An input i (or respectively output o) of a data service is a variable which represents an individual associated with a class defined in some ontology. The class is referred to as the input (or respectively output) type, denoted $Type(i)$ (or respectively $Type(o)$).

For our example data service, the input is a variable l and the outputs are variables s and d, where $Type(l) = Location, Type(s) = PrimarySchool, Type(d) = Distance$ and the concepts *Location*, *PrimarySchool* and *Distance* are defined in some ontologies.

Definition 2 (Pre-condition and Effect): A pre-condition of a data service is a logic formula defined on some inputs and some resources defined in some ontologies. An effect of a data service is a logic formula defined on some inputs, outputs and some resources defined in some ontologies.

Basically the pre-conditions are the conditions which must be satisfied for the service to be invoked. The effects are the conditions which are guaranteed to hold on the outputs after the service is invoked. For our example data service, the pre-condition is that the location provided is in Singapore and the effects are that the primary schools returned are within 1 kilometre from the given location.

Now we combine the notions of inputs, outputs, pre-conditions and effects to define data services.

Definition 3 (Data Service): A data service is a 4-tuple, $S = (\mathcal{I}, \mathcal{CI}, \mathcal{O}, \mathcal{CO})$ where $\mathcal{I}$ is a set of input parameters, $\mathcal{CI}$ is a set of pre-conditions, $\mathcal{O}$ is a set of tuples of output parameters and $\mathcal{CO}$ is a set of effects.

For our example data service, we have the following.

$$\mathcal{I} = \{l\}$$
$$\mathcal{CI} = \{isLocatedIn(l, Singapore)\}$$
$$\mathcal{O} = \{\langle s, d\rangle\}$$
$$\mathcal{CO} = \{d < 1\ km\}$$

After formally modeling the data services, we aim to discover the links and relationships between data services. We first establish a notion called type-matching between the parameters of two different data services as follows.

Definition 4 (Type-matching): An output parameter, p_1, of a service s_1 is defined to type-match an input parameter, p_2, of another service s_2, denoted $p_1 \preceq p_2$, if and only if $\text{Type}(p_1) \sqsubseteq \text{Type}(p_2)$.

The symbol $\sqsubseteq$ represents the standard sub-class relationship in Semantic Web ontologies. Then we say that the output p_1 of s_1 can be passed into the input p_2 of s_2.

Definition 5 (Partial and Total Service Dependency): A service dependency is defined as a 4-tuple (s_1, l_1, l_2, s_2) where s_1 and s_2 are both services, l_1 is a list of output parameters $[o_1, \cdots, o_n]$ associated with r and l_2 is a list of input parameters $[i_1, \cdots, i_n]$ of s_2. Furthermore it must hold that for all $1 \leq k \leq n$, $o_k \preceq i_k$. The dependency is defined to be partial if and only if l_2 is a strict subset of the set of all input parameters of s_2. The dependency is defined to be total if and only if l_2 is equal to the set of all input parameters of s_2.

Basically a service dependency represents a way in which a data service can be connected to another data service by passing **some** of its outputs to **some** of the inputs of the other data service. This set of service dependencies is created when the service repository is created and is updated when a new service is added into the repository or when an existing service is updated or removed from the repository. Although the update of service dependencies can be computationally intensive, especially when it is first created, it still helps to reduce the actual run time of the composition graph construction as the service dependencies are maintained offline rather than on-the-fly. Since it is done offline, a simple iterative approach is adopted to find all service dependencies and the result is stored in an indexed database for fast retrieval. Such a set of service dependencies are computed based on type matching of the services and are found with the help of some ontology mapping and an OWL ontology reasoner such as FaCT++ [3]. Here we do not elaborate on the specific algorithm for the step of creating and updating the service dependency set.

III. Data Integration as Semantic Service Composition

In the last section, we have described how we model the data services using semantic web services and how we capture dependency relationship between data services. The whole purpose of this is to be able to integrate data from different data services in a more meaningful, automated and efficient manner. In this section we investigate, given a user query

which is typically in the form of some user provided data and some user expected data, if it is possible to query different data services in a particular order so that the result datasets from different data services can be combined in a way to answer the user's query as if the user were querying a single data service. Our goal is to be able to find out such order of invocation automatically and efficiently.

We first formalize the notion of user queries as follow.

Definition 6 (User Query): A user query is a 4-tuple, $(\mathcal{I}_\mathcal{Q}, \mathcal{CI}_\mathcal{Q}, \mathcal{O}_\mathcal{Q}, \mathcal{CO}_\mathcal{Q})$, where $\mathcal{I}_\mathcal{Q}$ is the set of inputs that the user can provide, $\mathcal{CI}_\mathcal{Q}$ is the set of conditions on the inputs, $\mathcal{O}_\mathcal{Q}$ is the set of outputs that the user expects, and $\mathcal{CO}_\mathcal{Q}$ is the set of conditions that the user expects on the outputs and inputs.

Before we give the formal definition of service composition, we first re-define an arbitrary user query as two special services in our approach. This will facilitate the notions of service compositions and composition graphs later.

Definition 7 (User Query as Services): A user query $\mathcal{Q} = (\mathcal{I}_\mathcal{Q}, \mathcal{CI}_\mathcal{Q}, \mathcal{O}_\mathcal{Q}, \mathcal{CO}_\mathcal{Q})$ can be defined as two services, namely the starting service $s_s = (\varnothing, \varnothing, \mathcal{I}_\mathcal{Q}, \mathcal{CI}_\mathcal{Q})$ and the ending service $s_e = (\mathcal{O}_\mathcal{Q}, \mathcal{CO}_\mathcal{Q}, \varnothing, \varnothing)$.

In other words, the user query is simulated by using two services s_s and s_e. The starting service s_s takes nothing as input and does not have any pre-condition. It produces as output the user-provided input with the user-guaranteed pre-conditions as the effects of the service. Similarly the ending service takes as input the user-expected outputs with the user expected post-conditions as the input conditions and has an empty set of outputs and effects.

We now describe the problems of service composition and see how it can be used to solve the data integration problem. Informally, given a repository of services and a user query which cannot be satisfied by any single service, the service composition problem is to find a possibly sequential or parallel trace of service invocations such that

- all services in the trace use only the user provided input parameters or outputs from some preceding services,
- for each service in the invocation, its pre-conditions are satisfied
- each query output parameter is produced by exactly one of the services in the trace, and
- the query output parameters satisfy the user expected post-conditions.

Now formally the notion of service composition is defined as follows.

Definition 8 (Service Composition): Given a set of services $\mathcal{Z}$ and a user query $\mathcal{Q} = (\mathcal{I}_\mathcal{Q}, \mathcal{CI}_\mathcal{Q}, \mathcal{O}_\mathcal{Q}, \mathcal{CO}_\mathcal{Q})$, the service composition problem is to find a directed acyclic graph (DAG), $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges of the graph. $\mathcal{V} = \{\mathcal{V}_s, \mathcal{V}_e\} \cup \{\mathcal{V}_1, \ldots, \mathcal{V}_n\}$, where $\mathcal{V}_s$ and $\mathcal{V}_e$ are two vertices representing the starting service and the ending service derived from the user query and each $\mathcal{V}_i$ represents an instance of a service s_i in $\mathcal{Z}$. Each edge e_i from the vertex v_j to the vertex v_k is in the form of $(s_j, \langle o_1, \ldots, o_m\rangle, \langle i_1, \ldots, i_m\rangle, s_k)$ where o_1 through o_m are m outputs of the service s_j and i_1 through i_m are m inputs of the service s_k such that for all $1 \leq l \leq m, o_l \preceq i_l$. $\{o_1, \ldots, o_m\}$ is called the output parameter set of the edge and $\{i_1, \ldots, i_m\}$ is called the input parameter set of the edge. Furthermore the graph $\mathcal{G}$ satisfies the following conditions.

1) For each vertex v_i, the input parameter sets of all edges ending at v_i are disjoint and their union is equal to the set of input parameters of service s_i.
2) For each edge $(s_j, \langle o_1, \ldots, o_m\rangle, \langle i_1, \ldots, i_m\rangle, s_k)$, the pre-conditions of s_k are not logically conflicting with the effects of s_j.

Now with the definitions of data services, user queries and service composition, we take the following approach to DaaS-based data integration.

Annotating Data Service
: The first step is to semantically annotate the published data services with concepts and relationships defined in some common ontologies. We have illustrated the idea when we formally model data services in Section II. In the literature there are a lot of commonly used ontologies[3] to model the concepts and relationships used in the data service. This step allows the functionality of the data service to be comprehensible to both human and the composition engine to be used for data integration.

Aligning User Queries to Ontologies
: The second step is to match entities used in the user queries to the concepts and relations defined in some common ontologies. Typically data consumers will not do this while he or she queries the data services. So users may be given the options to select from some ontologies before he or she executes the queries or a typical ontology matching tool can be used after the user has produced the query.

Generating Composition Graph
: With the data services and user query semantically marked up, the third step is to invoke a service composition engine to find a DAG-based composition graph.

Translating Composition Graph to Queries
: The fourth step is to translate the composition graph into a set of queries which are to be executed on the data services in the composition graph. Because the composition graph clearly represents how user-provided inputs and service outputs flow into the inputs of other services, this step is fairly straightforward.

IV. Case Study

In this section, we illustrate our approach with a case study in the real estate domain. Typically when a consumer wants to purchase a property, he or she usually consider a range of factors with respect to the convenience level of the property.

[3]For example FOAF http://xmlns.com/foaf/0.1/ and vCard http://www.w3.org/2001/vcard-rdf/3.0#

These factors may include distances to reputable schools, super markets, bus stops, subway stations and malls, as well as some upcoming development in the vicinity. In addition to providing these information to the consumers as most property searching websites are already doing, it would be more direct and comparable to provide something like a "convenience index" by taking these factors into account. Suppose now that a real estate company has already developed the algorithm which computes this index from the schools, super markets, bus stops, subway stations and malls and their distances from the property as well as recent development news. What remains is just to get the data from the respective data sources on demand. Suppose the data services shown in Table I are available and their cfunctionalities are explained as follows.

- GetPropertyAddress (GPA) returns the address of a given property name.
- GetGPS (GG) returns the GPS coordinates of a given address.
- GetEligiblePrimarySchool (GEPS) returns a list of primary schools within 1 km from a given GPS coordinates and their distance from the location.
- GetPrimarySchoolRanking (GPSR) returns the average ranking of a given primary school over the past three years.
- GetFairPrice (GFP) returns the nearest FairPrice supermarket from a given GPS coordinates and its distance from the location.
- GetShopNSave (GSNS) returns the nearest Shop N Save supermarket from a given GPS coordinates and its distance from the location.
- GetColdStorage (GCS) returns the nearest Cold Storage supermarket from a given GPS coordinates and its distance from the location.
- GetBusStop (GBS) returns a list of bus stops within 1 km from a given GPS coordinates and their distance from the location.
- GetService (GS) returns a list of bus services available at a given bus stop.
- GetMRTStation (GMS) returns a list of MRT stations within 1 km from a given GPS coordinates and their distance from the location.
- GetMall (GM) returns a list of malls within 2 km from a given GPS coordinates and their distance from the location.
- NewsAggregator (NA) returns the url of the news in the past month from different news sources
- NewsFilter (NF) returns the url of a subset of a given set of news filtered by some keywords

In an actual scenario like this the factors may include many more such as the nearest child care centres and petrol stations. We believe the idea is illustrated and is extensible.

Now we apply our approach to the case study. First we create an ontology which include the concepts and relations involved in this case study and then we use this ontology to create the service ontologies in OWL-S [1]. Also using the ontology entities, we create the two services for the user query, namely the starting service and ending service. We invoke a service composition engine with the candidate services and the composition graph as shown in Fig. 1 is returned. We have implemented a service composition engine for the validation of our approach. The composition engine is a backward-chaining, DAG-based one with constraint solving for the service conditions. The composition produced is illustrated in Fig. 1. For visual clarity we omit the input/output parameters on the edges of the graph. The last step is to translate the composition graph into the following executable queries on the various data services on a given condominium called "La Casa".

- SELECT DISTINCT FilteredNews FROM GPA, NewsFilter, NewsAggregator, GG WHERE Addr = PAddress AND GPSOrd = CondoLocn AND Propname = "la casa" AND NewsList = NewsGroup ;
- SELECT DISTINCT BSdist,BSList,BSServiceList FROM GBS, GS, GPA, GG WHERE Propname = "la casa" AND Addr = PAddress AND GPSOrd = GBSord AND BSList = GSList ;
- SELECT DISTINCT SSAddress,SSdist FROM GPA, GSNS, GG WHERE GPSOrd = GSSord AND Addr = PAddress AND Propname = "la casa" ;
- SELECT DISTINCT Mrtdist,MRTStn FROM GMS, GPA, GG WHERE Addr = PAddress AND GPSOrd = GMrtord AND Propname = "la casa" ;
- SELECT DISTINCT FPdist,FPAddress FROM GFP, GPA, GG WHERE GPSOrd = GFPord AND Addr = PAddress AND Propname = "la casa" ;
- SELECT DISTINCT RankList,PSDist,PSList FROM GPSR, GEPS, GPA, GG WHERE Propname = "la casa" AND Addr = PAddress AND PSList = PrimList AND GPSOrd = GPSCo ;
- SELECT DISTINCT Mall,Malldist FROM GPA, GM, GG WHERE Propname = "la casa" AND Addr = PAddress AND GPSOrd = GMallord ;
- SELECT DISTINCT CSAddress,CSdist FROM GPA, GCS, GG WHERE Propname = "la casa" AND Addr = PAddress AND GPSOrd = GCSord ;

V. Related Work

A number of service-oriented approaches to integrate data have been reported in the literature [4], [5], [6]. Data federation is one of the most popular approaches where a virtual integration of the data sources is done through a mediating schema, and queries are translated through the latter into queries on the sources [7], [8]. The challenge however is in constructing the mediating schema and maintaining it. Our approach does not require a mediating schema, and instead can perform data integration dynamically based on the queries.

Web data mashups investigate data integration from web data and web API's perspective. Much of the focus is on assembly canvases for guiding development of web mashups.

TABLE I
DATA SERVICES

Service Name	Input	Output	Provider
GetPropertyAddress (GPA)	Property Name	Address	Urban Redevelopment Authority
GetGPS(GG)	Address	GPS Coordinates	Land Transport Authority
GetEligiblePrimarySchool (GEPS)	GPS Coordinates	Primary School, Distance	Ministry of Education
GetPrimarySchoolRanking (GPSR)	Primary School	Ranking	Ministry of Education
GetFairPrice (GFP)	GPS Coordinates	FairPrice Shop, Distance	FairPrice
GetShopNSave (GSNS)	GPS Coordinates	Shop & Save Shop, Distance	Shop & Save
GetColdStorage (GCS)	GPS Coordinates	Cold Storage Shop, Distance	Cold Storage
GetBusStop (GBS)	GPS Coordinates	Bus Stop, Distance	Land Transport Authority
GetService (GS)	Bus Stop	Bus Service	Land Transport Authority
GetMRTStation (GMS)	GPS Coordinates	MRT Station, Distance	SMRT
GetMall (GM)	GPS Coordinates	Mall, Distance	Singapore Tourism Board
NewsAggregator (NA)	None	News URL	Singapore Press Holding
NewsFilter (NF)	News URL, Location	Filtered News URL	Singapore Press Holding

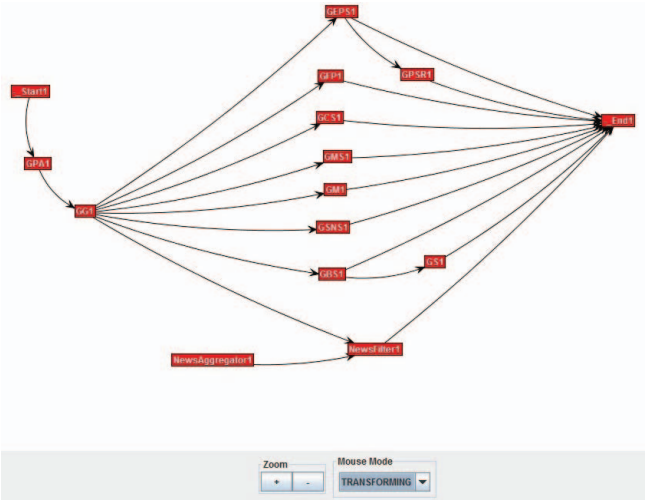

Fig. 1. Composition Graph

Popular tools include Micrsoft's Popfly[4], IBM's QEDWiki[5] and Yahoo! Pipes[6]. The tools are useful but can be overwhelming when a large repository of services are present. Our approach can automate much of the mashup process and free the user from having to create the entire mashup workflow.

Recently open data mashups have been investigated by the Linked Data community. The basic idea is to expose the data sources through Semantic Web standards such as RDF and OWL and connect them through URIs, so that a SPARQL query can be executed to retrieve data across the sources [9]. Composing the SPARQL is however non-trivial and can get tedious. SADI proposes a novel query interface to perform part of the SPARQL composition transparent to the user [10]. However SADI takes a simplified notion of services whereas our approach can be thought as generalization of these approaches to handle services which can be captured by the OWL-S framework.

[4]http://www.popfly.com

[5]http://blog.programmableweb.com/2007/02/08/enterprise-mashups-with-ibms-qedwiki/

[6]http://pipes.yahoo.com/pipes/

VI. Conclusion

In this paper we proposed a service-oriented, composition-based approach to the integration of open data services. We provided a formal, semantic-based modeling of data services and converted the integration problem into the service composition problem. The composition graph can be used to create the executable queries to the various data services to retrieve the user requested data.

Currently we are developing an enabling system which allows data service providers to easily create service ontologies for their data services which may be based on data sources in various formats such as databases, CSV files or even text files.

References

[1] The OWL Services Coalition, "OWL-S: Semantic Markup for Web Services," 2004, http://www.daml.org/services/owl-s/.

[2] D. L. McGuinness and F. van Harmelen (editors), "OWL Web Ontology Language Overview," 2003, http://www.w3.org/TR/owl-features/.

[3] D. Tsarkov and I. Horrocks, "FaCT++ Description Logic Reasoner: System Description," in *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, 2006, pp. 292–297.

[4] V. Hoyer and M. Fischer, "Market overview of enterprise mashup tools," in *Proceedings of the 6th International Conference on Service-Oriented Computing*, ser. ICSOC '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 708–721. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89652-4_62

[5] G. Di Lorenzo, H. Hacid, H.-y. Paik, and B. Benatallah, "Data integration in mashups," *SIGMOD Record*, vol. 38, no. 1, pp. 59–66, Jun. 2009. [Online]. Available: http://doi.acm.org/10.1145/1558334.1558343

[6] S. Dustdar, R. Pichler, V. Savenkov, and H.-L. Truong, "Quality-aware service-oriented data integration: requirements, state of the art and open challenges," *SIGMOD Record*, vol. 41, no. 1, pp. 11–19, Apr. 2012. [Online]. Available: http://doi.acm.org/10.1145/2206869.2206873

[7] P. A. Bernstein and L. M. Haas, "Information integration in the enterprise," *Communication ACM*, vol. 51, no. 9, pp. 72–79, Sep. 2008. [Online]. Available: http://doi.acm.org/10.1145/1378727.1378745

[8] L. Haas, "Beauty and the beast: the theory and practice of information integration," in *Proceedings of the 11th international conference on Database Theory*, ser. ICDT'07. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 28–43. [Online]. Available: http://dx.doi.org/10.1007/11965893_3

[9] C. Bizer, T. Heath, and T. Berners-Lee, "Linked Data - The Story So Far," *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 5, no. 3, pp. 1–22, MarMar 2009. [Online]. Available: http://dx.doi.org/10.4018/jswis.2009081901

[10] M. Wilkinson, B. Vandervalk, and L. McCarthy, "The semantic automated discovery and integration (sadi) web service design-pattern, api and reference implementation," *Journal of Biomedical Semantics*, vol. 2, no. 1, p. 8, 2011.