Models and Adaptive Architecture for Smart Data Management

Pierre De Vettor, Michaël Mrissa and Djamal Benslimane
Université de Lyon, CNRS
LIRIS, UMR5205, F-69622, France
Lyon, France
E-mail: firstname.surname@liris.cnrs.fr

***Abstract*—Organizations, companies and Web platforms hold large amounts of unused data. These data are trapped in separate data sources, locked up in legacy formats and only reachable through several different protocols, making usage difficult. It is therefore necessary to manage this multiplicity of data sources in order to build a solution able to combine this multi-origin data into a coherent smart data set. We define a meta-model and models to describe data source diversity in a flexible way. We therefore propose an adaptive architecture that generates data integration workflows at runtime. We evaluate our approach to offer scalability, responsiveness, and dynamic and transparent data source management. We apply our approach in a live scenario from a French company to show how it adapts to industrial needs and facilitates smart data production and reuse. This paper describes our models and strategies and presents our resource-oriented architecture.**

***Keywords*-resource oriented architecture; data integration; data semantics; smart data;**

I. Introduction

During the last few years, governments, companies and organizations have opened their databases and information systems to the world across the Web, thanks to initiatives such as the open data project [1]. Data sources are typically exposed via Web APIs [2] or SPARQL endpoints and can be combined in service mashups [3] to produce highly valuable services. For example, the sets of APIs provided by Twitter, Amazon, Youtube or Flickr are used in thousands of mashups[1].

The smart use of data has caught the interest of the community as a natural development after the interest on big data. The objective of smart data [4] is focused on producing high-quality data that is directly useful to users, instead of focusing on massive data quantities. Building smart data architectures, i.e. automatically integrating data from diverse sources, in currently a hot research topic. Related approches are focused on data quality and service optimization [5] or public ontology alignment [6]. Industrial approaches exist to give a universal access to data sources [7], [8], but none of these approaches is focused on data source characteristics to adapt data processing.

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

The solution we propose is a generic meta-model, and associated models, to describe data source characteristics and data access strategies, together with an adaptive architecture that generates workflows at runtime. We isolate each task as a separate process within our resource oriented architecture implementation. We identify the following challenges and scientific locks to address during the data integration process. *Dynamic and transparent data source management:* the possibility to transparently add or remove a data source at runtime without any need of hard coded information. *Scalability and responsiveness:* the solution must support a large number of data sources while offering low time response. *Dynamic data processing:* the solution needs to adapt data sources that require different processing (large data volume at runtime, frequent update, latency). *Data consistency:* provide consistent, error and duplicate-free data.

This paper is organized as follows. Section II presents our meta model and models for describing data sources. Section III explains our different processing techniques in order to handle the constraints and characteristics data sources provided. Section IV presents our resource-oriented architecture, details the different components and their orchestration. Section V gives an evaluation of our prototype in terms of responsiveness and shows how it responds to user requests with acceptable timings. Section VI presents related work and highlights the advantages of our approach. Section VII discusses our results and provides guidelines for future work.

II. Data Source Models

In order to address the challenges presented above, we propose a *meta-model* to describe the characteristics of data sources. Based on this metamodel, we define a *data source model* describing *physical* data source characteristics, a *data source meta-model* to describe extracted data and, finally, we define adaptive *data processing workflows*. Hence, these models are used to implement our smart data architecture that provides a resource-oriented solution for data integration.

A. A Meta-Model for Describing Data Sources

Each data source might follow a different format, structure and logic, and require processing strategies. Therefore, we need a flexible data source representation to handle these different data source capabilities. This data source meta-model is shown in Figure 1. In this meta-model, characteristics can describe either the source itself (i.e., useful characteristics to guide the interaction with the data source), or the data instances to be extracted.

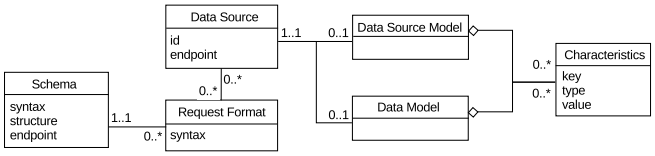

Figure 1. Data source metamodel

Our meta-model includes different core attributes. We identify the *URI* and the *request format* as the mandatory information to manage access to data sources. The *request format* characteristic is represented by a *syntax* attribute (e.g., XML, JSON, SQL) and a *schema* defined by three attributes: *endpoint*, *syntax* (e.g., XML, n3, JSON) and *structure* (e.g. RDFS, XSD, JSON Schema). One *syntax* field defines the request; the other defines the schema. Here we illustrate this meta-model with a set of specific characteristics, but it remains applicable to all kinds of characteristics and scenarios.

B. Data Source Description Model

Based on this meta-model, it is possible to define adapted data source models (as instances of the meta-model) to describe data sources. Fig. 2 shows the different characteristics of the data source model example used in our scenario.

URI identifies the data source and contains the necessary information to enable interactions with this data source (protocol, domain and resource name). *Request Format* defines how to interact with the data source (e.g., SQL, SPARQL, XML). *Update frequency* indicates the recommended average duration between each request to a data source. *Volume* represents the global quantity of data that a data source manages. This characteristic takes its value from the following enumeration: $[small, medium, high]$. This helps to adapt the access strategy (direct access, cache, synchronization, ...). *Latency* represents the average network time (regularly updated) required to obtain a response message to a request on a data source. *Authentication* describes the data source access restriction (HTTP-Auth, OAuth, SSL, Public). In some cases, auth parameters can be specified in the URI, e.g., `http://user:pass@test.com/`. *Semantics* aggregate the information required to perform the semantic transformation from raw data to linked data. The semantic description contains: an URI of the data model ontology, an URI of the mapping file that gives information about required data transformation and an attribute identifying the system used to perform the transformation. *Privacy* (of data source) agreements [9] define whether data is limited to a specific usage context or not, according to a set of conditions. Agreements can, as an example, avoid to provide a piece of data to a third party system, or prevent any modification or commercial use of a data piece.

C. Data Source Meta-Model

At the data level, there is a need for a model to describe the data we extract from sources. These attributes can apply to specific pieces or to global data sets, and *always override* the data source characteristics.

The model is composed of the following characteristics: *Privacy* (of data) attributes which override data source privacy for specific pieces of data (given by owner). *Validity* specifies the date after which data can be considered as obsolete. *Semantics* are conceptual information about extracted data, provided by data source or updated during semantic annotation. *Filters* are specific attributes which identifies the quality in the data set. Filters can specify a detected malformed piece of data, or a forbidden value.

D. Data Access Strategies

In the following, we present our data access strategies to adapt to cases where volume or latency problems hamper data access. A data access model describes several characteristics that affect the way a client connects to and downloads data from data sources. According to characteristic values, different data access policies can be adopted. We adapt our access strategies to the three following characteristics: *data volume*, *latency* and *update period*. High latency and medium volumes require the data to be cached. Update period helps to set the synchronization delay. In the following section, we present the different processing tasks required to perform the integration process.

III. Data Processing Workflows

From the needs we have identified in the previous sections, we define processing tasks, which we combine in workflows to generate smart data. In this section, we envision different data processing workflows as combinations of functions in different orders. We list in the following the different kind of functions that our architecture manages as resources, before presenting how the different possible workflows are constructed.

A. Defining Processing Functions

We consider a data source DS_a, defined by a set of characteristics. We define a download function $DL()$ that extracts a quantity of data D from a data source DS_a.

Definition *1:* $DL(URI_a, S_a) = D_a$ where URI_a, S_a represent the data source DS_a (URI and Model) and D_a the data extracted.

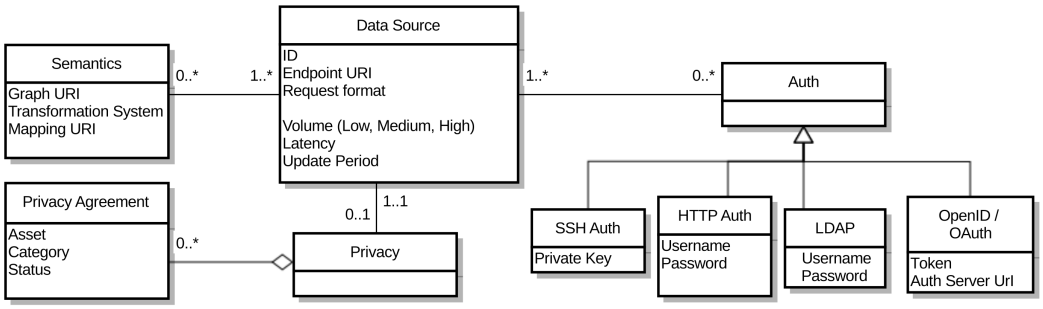

Figure 2. A data source model based on our scenario

An access function can accept optional parameters in order to handle the auth protocol (query for databases, authentication parameters, etc.). In order to be processed, data need to be transformed from its raw extracted format to a format we can manipulate. We define a decoding function $Dec()$ which will transform the data into our standard format.

Definition *2:* $Dec(D_{a,r}) = D_{a,f}$ where $D_{a,r}$ is the data extracted into its raw format and $D_{a,f}$ is the transformed data.

In order to aggregate data from multiple data sources, concepts from both data source must belong to the same ontology, which we called G. In order to transform the data extracted into linked data instances I_a, we define a mapping function $Sem_a()$ which is defined as:

Definition *3:* $Sem_a(D_a, G) = I_a$ where D_a is the data extracted from data source DS_a, G is the ontology graph, and I_a the linked data graph produced from DS_a.

Once data has been extracted and semantically enhanced, it can easily be combined into a new data set. We define an integration function called $I()$ which takes as input the data sets that have been previously annotated and combines them into a new one.

Definition *4:* $I(G, D_a, D_b, ...) => D_{mix}$ where G is the semantic graph of manipulated data and (D_a, D_b, ...) are semantically transformed extracted data from data source (DS_a, DS_b, ...). Finally, D_{mix} is the smart data set result.

This function aggregates data and links common data concepts together. Before this combination, the integration function analyzes data, detects heterogeneities and provides mediation based on our previous Data Mediation as a Service approach [10]. This approach proposes a architecture that solves inconsistencies in service compositions.

Finally, we define a function $F()$ that removes the malformed part, noise and inconsistencies that may appear in a data set D_a after processing. This function can also take as input a set of conditions to filter data.

Definition *5:* $F(D_a, < Filters >) => D_{a,clean}$

The different processing tasks defined here will help to complete the tasks that participate in the integration process. In the following, we present how these functions can be combined into different processing workflows depending on the characteristics described in the data source and data models.

B. Interaction models in the architecture

Figure 3 presents an example of process orchestration in order to integrate data coming from two data sources called S_1 and S_2. Processes are executed from left to right, where boxes represent the previously defined functions. Data is going through the following steps: download (Dl), decode (Dec), transformation into linked data with help from a mapping description (Sem), then both data set are integrated (I) and finally filtered (F).

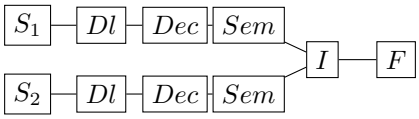

Figure 3. Typical integration workflow

During execution data goes through different states, from the raw original format following data extraction, to a format that facilitates manipulation, and finally to the linked data format once annotated. The move from one state to another may have an impact on processing in terms of response time (especially when processing a huge data volume) or data consistency (streams VS static DB). In a classical workflow, data is typically transformed into linked data before the integration task, because semantic annotations facilitate the integration process.

The originality of our approach is that our adaptive architecture generates workflows at runtime in order to fit with the data source characteristics. Tasks are moved forward or backward in workflows to limit volume of data to be processed. In the following example, the workflow is optimized by placing the filtering process before integration and semantic transformation tasks.

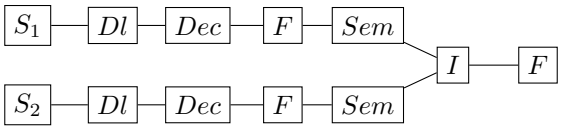

Figure 4. Optimized workflow

IV. A Smart Data Architecture

Using these models and strategies, we define a resource-oriented architecture in which we define the different tasks required to prepare, semantically annotate and clean data. Through these different steps, data is transformed into a consistent "smart data" set

A. Global Overview: a Resource-Oriented Architecture

Our architecture follows the principles of SOA [11], which makes our components decoupled, cohesive and reusable, thanks to its properties: *Loose Coupling*, *Abstraction*, *Reusability*, *Autonomy* and *Composability*.

In order to build a completely generic and modular architecture, we deploy our components as RESTful resources, i.e., identified by URIs and accessible through HTTP methods. Thanks to the SOA and REST principles [12], our architecture is generic, scalable and modular. It is composed of different resources that can be dynamically orchestrated as presented in the following.

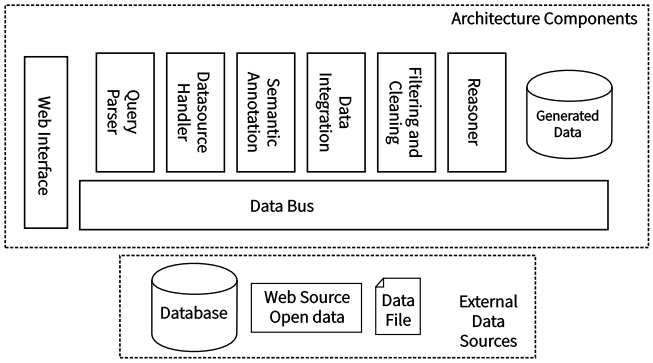

Figure 5. Architecture Resources

We define generic RESTful resources [13] to handle the main data processing and management tasks, as shown in Fig. 5.

B. Core Resources

In the following, we present the *core* resources that handle the tasks presented in Section III.

The *data source handler* manages the access and data extraction from data sources (DL and Dec tasks of Section III-A).

Semantic annotation resources help to annotate and transform data coming from diverse sources into linked data (Sem task). We rely on existing semantic transformation approach such as D2R[14] and Tarql [15] to perform the semantic annotation and transformation.

Data integration resources help combine multi-origin data (I task) and resolve heterogeneities that appear relying on our previous DMaaS approach [10].

Filtering and cleaning resources filter data and remove duplicates as well as malformed pieces of data (F task).

We added a *reasoner* running as a background task to infer new facts from existing data.

Finally, the *Web Interface* combined with the *Query Parser* handle user interaction and data requests to the architecture.

V. Tests and Evaluations

Our architecture implementation and our adaptive models and strategies provide a dynamic and transparent data source processing. Data consistency is provided by filtering and cleaning tasks, which can be placed anywhere in our workflows. On the other hand, the scalability to a large number of data source cannot be guaranteed *a priori* by our model and implementation. We answer the scalability challenge by evaluating the evolution of complex query response time over a growing number of data sources. We regularly increase the number of data sources and measure the response time.

The objectives we identified are classical in the field of data integration from data sources. Scalability, responsiveness and data consistency are inherent to quality-aware data architecture. On the other hand, transparent and dynamic data source management and processing are specific objectives we tried to address in our work, by taking a provenance point of view.

A. Scenario

In order to evaluate our approach, we focus on a live scenario from a company, which has a need for an adaptive system to automatically combine their data with information from Web sources in order to study the campaign broadcasting impact over their customers. The scenario describes the following data sources, each of them presenting different characteristics.

1) an internal linked service giving access to our company business data
2) a SQL database containing a large volume of information (around 10GB)
3) a SQL database that records user activities (high volume of changing data) with 10.000/20.000 new tuples per day
4) a RSS stream that contains user update requests
5) external APIs (DbPedia sparql endpoint, FOAF ontology, etc.)

Relying on this scenario, we create two requests, involving different concepts. We populate our scenario with a set of data sources covering the different subgraphs. Query 1

involves four concepts, implying different types of data sources with different characteristics such as high volume (big database in our scenario) and privacy sensitive information (linked service in our scenario).

```
PREFIX al: <http:// restful . alabs . io / concepts/0.1/>
SELECT ?email_value ?campaign WHERE {
    ?email          a                       al:email  ;
        al:has_email_value   ?email_value  .
    ?email_value  a                         al:email_value  .
    ? clic          a                       al: clic  ;
        al: clic_email          ?email  ;
        al:clic_campaign        ?campaign .
    ?campaign    a                          al:campaign .
}
```

Listing 1. Query 1 involving four data sources

Query 2 involves less concepts, but includes user specific filters. This query introduces freshness sensitive data sources (streams in our scenario).

```
PREFIX al: <http:// restful . alabs . io / concepts/0.1/>
PREFIX xsd: <http://www.w3.org/TR/xmlschema-2/>
SELECT ?email_value ?campaign WHERE {
    ?email          a                       al:email  ;
        al:has_email_value   ?email_value  ;
        al:  blacklist_status   ? status  .
    ? clic          a                       al: clic  ;
        al: clic_email          ?email  ;
        al: clic_date           ?date  .
    FILTER (? status  != 1 && ?date >= "1411477450"^^xsd:date)
}
```

Listing 2. Query 2 introducing user specific filters

Tests are performed on a double core 2.3 GHz machine, with 4 GB of RAM. Restful resources and architecture are implemented through PHP frameworks, Zend and Slim[2]. Resources are hosted on scenario company servers (Apache/PHP hosting servers), while architecture is hosted locally.

B. Results

In this subsection, we evaluate our architecture response time to queries $Q1$ and $Q2$ respectively, when the number of data sources increases. We compared two composition techniques, identified as $WF1$ and $WF2$. $WF1$ is composed of the different steps of the integration process in a static order, whereas $WF2$ introduces a dynamic composition, with the ability to optimize component orchestration.

As can be seen in Fig. 6 and Fig. 7, when the number of data source increases, there is an exponential increase of execution time for the static composition *WF1*. In parallel, the response time of composition *WF2* increases linearly.

For query *Q1*, the combination step of the data integration process becomes time-consuming, and the response time increases exponentially. The evolution of response time for query *Q2* is less significant than query *Q1*, because of introduction of filters in the query. When the number of data sources exceeds 16 for *Q1*, and 15 for *Q2*, composition technique *WF1* is unable to provide a response in less than a minute.

[2]See http://www.slimframework.com/

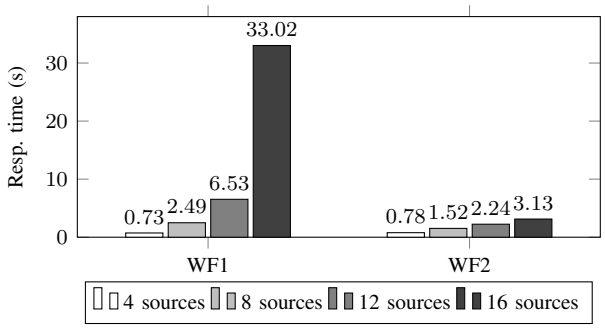

Figure 6. Average response time for Query 1

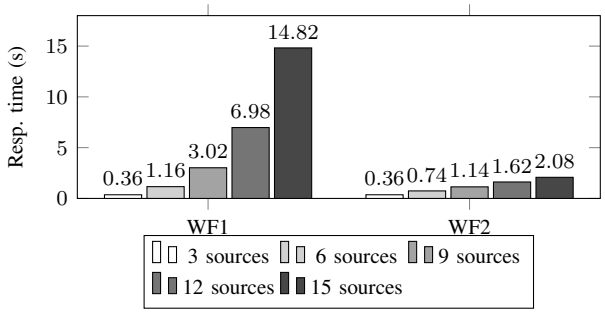

Figure 7. Average response time for Query 2

These graphs clearly show that our architecture can adapt to a large number of data sources, as long as we use a dynamic composition model.

VI. Related Work

Building an architecture to automatically integrate data from diverse resources in order to produce smart data is currently a hot research topic explored by the community.

Dustdar et al. present a *peer data network* architecture in [5], where data sources are independent databases. Their infrastructure solution focuses on quality of data and provides service-based optimization, such as peer replication, to resolve data issues. However, the paper does not address data heterogeneity problems, assuming that schema mapping is sufficient.

QuerioCity [6] presents a smart platform to catalog, index and query heterogeneous information from open data portals[3]. They focus on data integration, annotating data with public ontologies (Dublin Core [16] or FOAF [17]). They do not provide any information about data sources, assuming that sources have to meet the format that the architecture supports.

On top of that, there has been a lot of approaches to transform raw data to linked data in order to make these data reusable, like *D2R* [14], *RDF123* [18] and *Tarql* [15]. These approaches bring valuable tools, but the aggregation work has to be performed on top of them.

[3]Such as Dublinked http://www.dublinked.ie/

Some approaches, such as *SmartData.io* [7] or *Apache Metamodel* [8] present more industrial or technical automated management of data coming from heterogeneous sources. These solutions are APIs and frameworks that provide transparent interface to data sources but they did not address challenges related to data combination or semantics.

We have taken into account the strengths and weaknesses of these different approaches to build our proposal, improving the reusability and loose coupling through usage of linked data services, automating the linked data efforts by proposing a distributed approach for the different tasks to perform on data.

We chose to focus on data sources, in response to our specific objective by design. At the same time, we focused on scalability and responsiveness, justifying our approach with a set of tests and response time evaluation where we give acceptable results regarding these objectives.

VII. Conclusion

In this work, we build the foundation of a smart data architecture, which is able to extract, annotate and combine data coming from different data sources. We propose a flexible solution to model data sources and data according to their characteristics, allowing to use different data access and processing strategies. Our adaptive architecture generates workflows at runtime, adapting process to data source characteristics. It aims at being as generic as possible, independent of data sources, and adaptable to any use case. We implement and evaluate our architecture in the context of a scenario that answers the needs of our partner company. Future work includes performing additional evaluation over large data sets and exploring issues related to data management such as data quality or freshness issues. It also includes handling uncertainty that can appear in data aggregation from multi-origin data sources.

References

[1] T. Heath and C. Bizer, *Linked Data: Evolving the Web into a Global Data Space*, ser. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.

[2] M. Maleshkova, C. Pedrinaci, and J. Domingue, "Investigating web apis on the world wide web," in *Web Services (ECOWS), 2010 IEEE 8th European Conference on*, Dec 2010, pp. 107–114.

[3] D. Benslimane, S. Dustdar, and A. P. Sheth, "Services mashups: The new generation of web applications," *IEEE Internet Computing*, vol. 12, no. 5, pp. 13–15, 2008.

[4] A. Sheth, "Transforming big data into smart data: Deriving value via harnessing volume, variety, and velocity using semantic techniques and technologies," in *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, March 2014, pp. 2–2.

[5] 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, 2012.

[6] V. Lopez and S. Kotoulas, "Queriocity: A linked data platform for urban information management." in *International Semantic Web Conference (2)*, ser. Lecture Notes in Computer Science, vol. 7650. Springer, 2012, pp. 148–163.

[7] AtosWorldline, "Smartdata.io : A dedicated solution to the data management," http://api.docs.v2.smartdata.io/, 2013. [Online]. Available: http://api.docs.v2.smartdata.io/

[8] Apache, "Apache metamodel : A data access framework," http://metamodel.incubator.apache.org/, 2013. [Online]. Available: http://metamodel.incubator.apache.org/

[9] H.-L. Truong, S. Dustdar, J. Goetze, T. Fleuren, P. Mueller, S.-E. Tbahriti, M. Mrissa, and C. Ghedira, "Exchanging data agreements in the daas model," in *The 2011 IEEE Asia-Pacific Services Computing Conference*. IEEE, Dec. 2011, pp. 153–160.

[10] M. Mrissa, M. Sellami, P. D. Vettor, D. Benslimane, and B. Defude, "A decentralized mediation-as-a-service architecture for service composition," *2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, vol. 0, pp. 80–85, 2013.

[11] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.

[12] L. Richardson and S. Ruby, *Restful Web Services*, 1st ed. O'Reilly, 2007.

[13] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," in *Proceedings of the 22Nd International Conference on Software Engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 407–416. [Online]. Available: http://doi.acm.org/10.1145/337180.337228

[14] C. Bizer, "D2rq - treating non-rdf databases as virtual rdf graphs," in *In Proceedings of the 3rd International Semantic Web Conference (ISWC2004*, 2004.

[15] R. Cyganiak, "Tarql (sparql for tables): Turn csv into rdf using sparql syntax," January 2015, http://tarql.github.io/about/. [Online]. Available: http://tarql.github.io/about/

[16] S. Weibel and T. Koch, "The dublin core metadata initiative : Mission, current activities, and future directions," *D-Lib Magazine*, vol. 6, no. 12, 2000.

[17] D. Brickley and L. Miller, "The friend of a friend (foaf) vocabulary specification," November 2007, http://xmlns.com/foaf/spec/. [Online]. Available: http://xmlns.com/foaf/spec/

[18] L. Han, T. Finin, C. Parr, J. Sachs, and A. Joshi, "Rdf123: From spreadsheets to rdf," in *The Semantic Web - ISWC 2008*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, vol. 5318, pp. 451–466.