A Semantic Framework for Integrating and Publishing Linked Data on the Web

by

Aparna Padki

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved June 2016 by the
Graduate Supervisory Committee:
Srividya Bansal, Chair
Ajay Bansal
Timothy Lindquist

ARIZONA STATE UNIVERSITY
August 2016

ABSTRACT

Semantic web is the web of data that provides a common framework and technologies for sharing and reusing data in various applications. In semantic web terminology, linked data is the term used to describe a method of exposing and connecting data on the web from different sources. The purpose of linked data and semantic web is to publish data in an open and standard format and to link this data with existing data on the Linked Open Data Cloud. The goal of this thesis to come up with a semantic framework for integrating and publishing linked data on the web. Traditionally integrating data from multiple sources usually involves an Extract-Transform-Load (ETL) framework to generate datasets for analytics and visualization. The thesis proposes introducing a semantic component in the ETL framework to semi-automate the generation and publishing of linked data. In this thesis, various existing ETL tools and data integration techniques have been analyzed and deficiencies have been identified. This thesis proposes a set of requirements for the semantic ETL framework by conducting a manual process to integrate data from various sources such as weather, holidays, airports, flight arrival, departure and delays. The research questions that are addressed are: (i) to what extent can the integration, generation, and publishing of linked data to the cloud using a semantic ETL framework be automated; (ii) does use of semantic technologies produce a richer data model and integrated data. Details of the methodology, data collection, and application that uses the linked data generated are presented. Evaluation is done by comparing traditional data integration approach with semantic ETL approach in terms of effort involved in integration, data model generated and querying the data generated.

DEDICATION

This thesis is dedicated to my husband, Srikar Deshmukh for his constant support, motivation and love. I also dedicate this thesis to my family for their blessings and support.

ACKNOWLEDGMENTS

I would like to thank my supervisor, Dr. Srividya Bansal, for her constant support, guidance and motivation during the course of the development of this thesis and through the entire coursework of my Master's degree. My appreciation and thanks also goes to Dr. Ajay Bansal and Dr. Timothy Lindquist for their support and encouragement throughout my time in Arizona State University.

I would also like to acknowledge and thank Jaydeep Chakraborty for his contribution in the evaluation process and Neha Singh for her contribution in researching various traditional ETL tools.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Motivation

With the widespread usage of internet, social media, technology devices and mobile applications, big data is only getting bigger and the sources of this data, types of format of data (structured and unstructured) are increasing as well. In order to be able to utilize this big data, a data integration process from these aforementioned heterogeneous sources is imperative. A wide variety of data warehouses exist that store these huge datasets to be analyzed and visualized for various business/research needs.

The definition of a data warehouse is a central repository for all or significant parts of data that an enterprise's various business systems collect. From this definition, we can understand two important concepts in a data warehouse: (1) The source(s) of data populated in this repository is usually multiple (2) the repository should hold this large amount of data in a coherent/useful format, irrespective of the disparate sources. The general architecture of a data warehouse is shown in Figure 1.

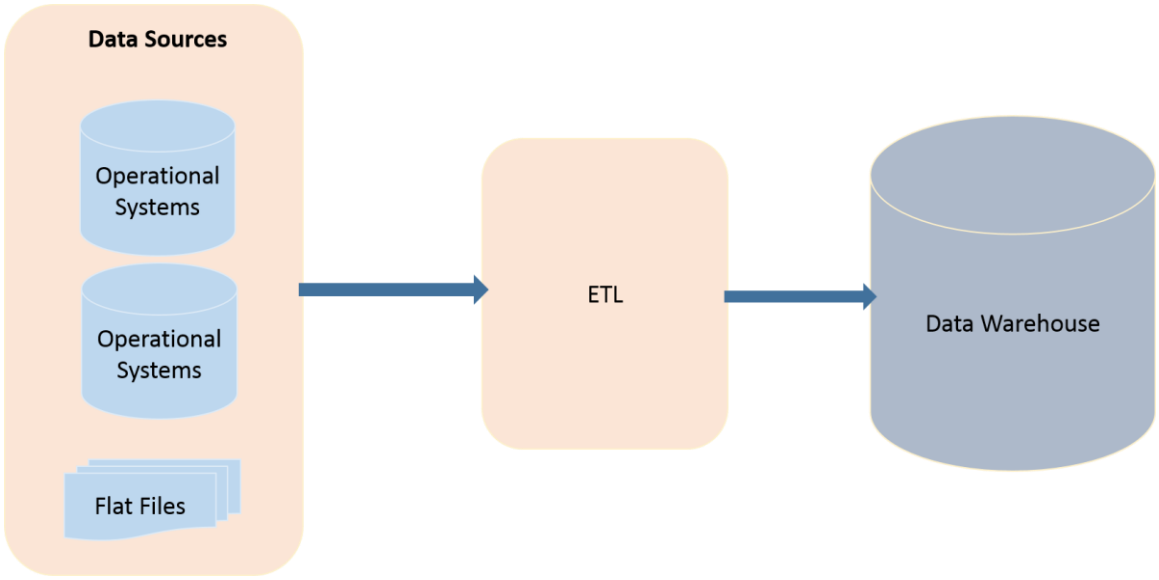


Figure 1 Data warehouse Architecture

In order to store data from a wide range of sources into a central repository, Extract-Transform-Load (ETL) tools are widely used [1]. Each stage in an ETL process can be defined as:

Extract the data – Obtain data from different sources and convert it into an intermediate format.

Transform the data - Business rules are applied to the data. Also, to make the data more efficient columns are split or merged. This is done in order to combine data from the extract phase to make it more meaningful and useful. This phase also involves cleaning of the data.

Load the data – The transformed data is loaded to a data warehouse for a single access of this data for analytics and visualizations.

In the semantic web world, we can draw a parallel to this process of loading a data warehouse for publishing linked data to the linked open data cloud [2]. A Semantic component (i.e., richer representation of the data model to provide more meaning, context, and relationships) in the transform phase is the major missing component in the traditional ETL tools if it has to be used for generating linked data. By introducing this semantic element, we can associate a domain-specific ontology to the key elements/fields in the dataset that is under integration. This is essential in generating linked data and connecting it to the linked open data (LOD) cloud. Using a semantic ETL framework, a richer dataset can be published onto the linked data cloud. This allows, like in a data warehouse, the ability for analytics and visualizations to be performed on this rich dataset. This thesis proposes an enhancement to the transform stage of the semantic ETL tool. The proposed semantic ETL process would involve following stages:

Extract the data – Obtain data from different sources, usually in different formats such as CSV, XML. JSON etc. and convert it into an intermediate format

Transform the data – In addition to the functions performed in the transform phase in the traditional ETL tools, this phase involves mapping of the newly created concepts/fields to an ontology, addition of rules and relationships, and generation of RDF data.

Load the data – The transformed linked data is loaded onto a triple store repository like HDFS and Hive, or Apache Fuseki [3] server with an endpoint for querying.

1.2 Problem Statement

The problem statement of this thesis is two-fold:

- RQ1: Can we introduce some level of automation in the integration, generation, and publishing of publish linked data from multiple data sources to the cloud by introducing a semantic component in the transform phase of the ETL process?
- RQ2: By using semantic technologies in the transform phase of ETL, can a richer data model and integrated data be published to the linked data cloud?

Based on these questions, following hypothesis is put forward:

The introduction of semantics in the form of ontology in ETL will enable the design of a semantic data model of the integrated data from multiple sources with relationships between common concepts as well as relationships with existing popular ontologies on the LOD cloud. The linked data generated in compliance with this data model will not only be integrated but also connected to the rest of the LOD cloud.

By providing scaffolding to a user to create an ontology, and add meaningful relationships, semi-automatic semantic ETL framework can be made available to the users with minimal human intervention.

1.3 Scope

In this thesis, we discuss the related work from the research community working on semantic ETL frameworks. We analyze various software tools and present a comparison of these tools. The tools that we have analyzed are of two categories:

A. Non – traditional ETL frameworks:

a. Karma – Semi automatic mapping of structured sources into semantic web [4]
b. TABEL - A domain independent and extensible framework for inferring the semantics of table [5]
c. Google Refine with the RDF extension [6]
d. TalendforSW [7]
e. Tarql [8]

B. Traditional ETL frameworks:

a. Clover ETL [9]
b. Talend Studio [10]

We have retrieved data about flight delays and incidents, holidays, and weather details for different cities. A detailed analysis of the data, semantic data model generation, integrations of data, and publishing the data as linked open data is presented. Requirements for a semantic ETL framework is discussed by presenting the process involved in generating linked data using semantic technologies along with ETL. Existing algorithms for some of the steps involved in the process are discussed and possible solutions for proposed requirements are presented. A web application was built that uses the integrated datasets to query and present the data in a useful information. The design and implementation of the web application is presented. An evaluation of the semantic ETL approach is presented by comparing against a traditional ETL approach using a tool called CloverETL [9] in terms of effort involved in integration, quality of data model generated, and querying the data generated.

CHAPTER 2

BACKGROUND

Semantic Web technologies enable people to create data stores in the form of triples on the Web, build vocabularies, and write rules for handling data. Linked data is empowered by technologies such as RDF, SPARQL, OWL [11].

2.1 Semantic Technologies

2.1.1 RDF

RDF, Resource Description Framework [12], is a standard model for data interchange on the Web. RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed.

RDF extends the linking structure of the Web to use URIs [13] to name the relationship between things as well as the two ends of the link (this is usually referred to as a “triple”). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications [14].

For example, a statement, A Flight has destination city Phoenix, AZ can be represented as a graph:

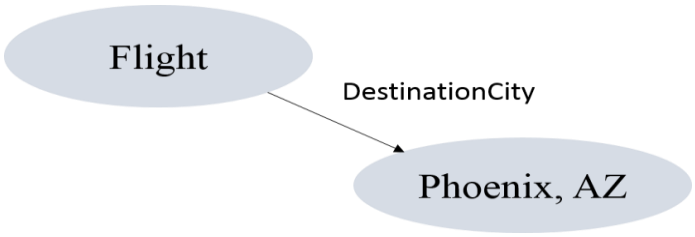


Figure 2 - Statement in form of a simple graph

In terms of the simple graph above, the Subject is Flight, Predicate is Destination City and object is Phoenix, AZ. RDF statement can be written in RDF/XML or TTL/Turtle – Terse RDF Triple Language format. This format allows RDF graphs to be written out ('serialized') in a compact and natural text form that is easier for humans to read than RDF/XML [15]. The simple graph above can be represented as TTL statement as:

```
@prefix ns0: <http://www.asu.edu/semanticWeb#> .

<http://semanticWeb/planYourTrip/date-2014-01-01-ID-AA-num-165-origin-
ChicagoIL> ns0:DestinationCity "Phoenix, AZ" .
```

The same statement written in RDF/XML format:

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:ns0="http://www.asu.edu/semanticWeb#">

  <rdf:Description rdf:about="http://semanticWeb/planYourTrip/date-2014-01-01-ID-
AA-num-165-origin-ChicagoIL">
    <ns0:DestinationCity>Phoenix, AZ</ns0:DestinationCity>
  </rdf:Description>

</rdf:RDF>
```

2.1.2 OWL

The W3C Web Ontology Language (OWL) [16] is a Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things. OWL documents, known as ontologies, can be published in the World Wide Web and may refer to or be referred from other OWL ontologies. OWL is part of the W3C's Semantic Web technology stack, which includes RDF, SPARQL etc. [14].

An example of OWL ontology defining the knowledge about flights:

```
<rdf:RDF xmlns="http://www.semanticweb.org/team7/Flight#"
    xml:base="http://www.semanticweb.org/team7/Flight"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
   <owl:Ontology rdf:about="http://www.asu.edu/semanticWeb">
      <owl:imports rdf:resource="http://schema.rdfs.org/all"/>
   </owl:Ontology>
    <owl:ObjectProperty rdf:about="http://www.asu.edu/semanticWeb#hasAirline">
      <rdfs:range rdf:resource="http://www.asu.edu/semanticWeb#Airline"/>
      <rdfs:domain rdf:resource="http://www.asu.edu/semanticWeb#Flight"/>
   </owl:ObjectProperty>
 <owl:DatatypeProperty rdf:about="http://www.asu.edu/semanticWeb#DestinationCity">
      <rdfs:domain rdf:resource="http://www.asu.edu/semanticWeb#Flight"/>
      <rdfs:range rdf:resource="&xsd;string"/>
   </owl:DatatypeProperty>
  <owl:Class rdf:about="http://www.asu.edu/semanticWeb#Airline">
      <rdfs:subClassOf rdf:resource="http://schema.org/Organization"/>
      <rdfs:comment>Class describing the airline</rdfs:comment>
   </owl:Class>
</rdf:RDF>
```

The ontology above represents a class Airline with object and data properties has Airline and DestinationCity and is subclassOf Organization from a public ontology called Schema. In addition, comments can also be provided.

The primary purpose of an ontology is to classify things in terms of semantics or meaning. In OWL, this is achieved through the use of classes and subclasses, instances of which in OWL are called individuals [17]. In the example shown above, following hierarchy is achieved using **subClassOf** property.

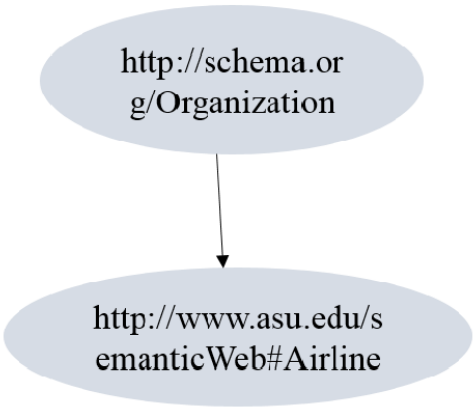


Figure 3 - Hierarchy in the Ontology

The other two properties covered in the example are:

- Object properties (owl:ObjectProperty) relates individuals (instances) of two OWL classes [17].
- Datatype properties (owl:DatatypeProperty) relates individuals (instances) of OWL classes to literal values [17].

2.1.3 SPARQL

SPARQL [18] Protocol and RDF Query Language is an RDF query language, that is, a semantic query language for databases, able to retrieve and manipulate data stored in Resource Description Framework (RDF) format. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph

patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries by source RDF graph. The results of SPARQL queries can be results sets or RDF graphs.

Using the running example, we can write a SPARQL query to return distinct destination cities that a particular Airline goes to:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX flight: <http://www.asu.edu/semanticWeb#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT  DISTINCT ?destination
WHERE{   ?temp flight:DestinationCity ?destination. ?temp flight:hasAirline ?airline.
?airline flight:hasName "AA"; }ORDER BY ( ?destination)
```

This query returns the result set:

- Chicago, IL
- Dallas/Fort Worth, TX
- Detroit, MI
- Miami, FL
- Newark, NJ
- Phoenix, AZ
- Santa Ana, CA

A SPARQL selection query has the following general form:

- PREFIX (Namespaces prefixes)
  Example: PREFIX flight: <http://www.asu.edu/semanticWeb#>
- SELECT (Result Set)
  Example: SELECT DISTINCT ?destination
- FROM (Data Set)
  the link to the URI where dataset(rdf/ttl) file resides
- WHERE (Query Triple Pattern)
  Example: WHERE{   ?temp flight:DestinationCity ?destination. ?temp flight:hasAirline ?airline. ?airline flight:hasName "AA";
- ORDER BY , etc (Modifiers)
  Example: ORDER BY (?destination)

2.1.4 Protégé

Protégé is a free, open-source platform that provides a suite of tools to construct domain models and knowledge based applications with ontologies. It is an OWL ontology development environment. Protégé has web and desktop version. Our technology stack uses desktop version. The features of this version are [19]:

- The GUI framework has a configurable, persistent layout of components
- User configured tabs can be easily created, imported and exported
- Same ontology can be viewed in multiple, alternative views. For example, class view, object property, data property view etc.
- Components can be easily torn-off and cloned
- The GUI provides various keyboard shortcuts
- Framework provides drag and drop support
- To improve speed and memory usage, lazy loading of components and plug-ins is done.

2.2 Extract-Transform-Load Frameworks

2.2.1 Clover ETL

CloverETL [9] is a Java-based data integration ETL platform for rapid development and automation of data transformations, data cleansing, data migration and distribution of data into applications, databases, cloud and data warehouses. CloverETL is a Java-based ETL tool with open source components. It is either used in standalone mode – as a command-line or server application – or embedded in other applications – as a Java

library. CloverETL is accompanied by the CloverETL Designer graphical user interface available as either an Eclipse plug-in or standalone application.

A data transformation in CloverETL is represented by a transformation dataflow, or graph, containing a set of interconnected components joined by edges. A component can either be a source (reader), a transformation (reformat, sort, filter, joiner, etc.) or a target (writer). The edges act as pipes, transferring data from one component to another. Each edge has a certain metadata assigned to it that describes the format of the data it transfers. The transformation graphs are represented in XML files and can be dynamically generated. A sample graph in clover ETL is shown below

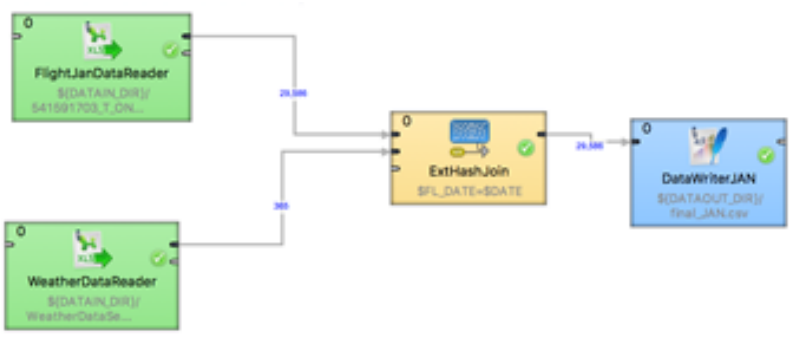


Figure 4 - Clover ETL sample workflow

Each component runs in a separate thread and acts either as a consumer or a producer. This is used to drive data through the transformation for both simple and complex graphs and makes the platform extendable by building custom components, connections etc. Transformation graphs can then be combined into a job flow, which defines the sequence in which the individual graphs are executed.

2.2.2 Talend Open Studio

Talend [10] is an open source data integration product developed by Talend and designed to combine, convert and update data in various locations across business. Talend Open Studio for Data Integration operates as a code generator, producing data-transformation scripts and underlying programs in Java. Its eclipse based GUI gives access to metadata repository and to a graphical designer. The metadata repository contains the definitions and configuration for each job - but not the actual data being transformed or moved. All of the components of Talend Open Studio for Data Integration use the information in the metadata repository.

Talend Open Studio is a powerful and versatile open source solution that addresses all of an organization's data integration needs:

- Synchronization or replication of databases
- Right-time or batch exchanges of data
- ETL (Extract Transform Load) for BI or analytics
- Data migration
- Complex data transformation and loading
- Basic data quality

2.2.3 Pentaho Data Integration (Kettle)

Pentaho Data Integration (PDI, also called Kettle) [20] is the component of Pentaho responsible for the Extract, Transform and Load (ETL) processes. Though ETL tools are most frequently used in data warehouses environments, PDI can also be used for other purposes:

- Migrating data between applications or databases

- Exporting data from databases to flat files
- Loading data massively into databases
- Data cleansing
- Integrating applications

PDI is easy to use. Every process is created with a graphical tool where you specify what to do without writing code to indicate how to do it; because of this, you could say that PDI is metadata oriented.

PDI can be used as a standalone application, or it can be used as part of the larger Pentaho Suite. As an ETL tool, it is the most popular open source tool available. PDI supports a vast array of input and output formats, including text files, data sheets, and commercial and free database engines. Moreover, the transformation capabilities of PDI allow you to manipulate data with very few limitations.

The basics of clustering in Pentaho Data Integration is very simple. The user configures one or more steps as being configured: A cluster schema is essentially a collection of slave servers. In each collection you need to pick at least one slave server that we will call the Master slave server or master. The master is also just a carte instance but it takes care of all sort of management tasks across the cluster schema. In the Spoon GUI you can enter this metadata as well once you started a couple of slave servers.

2.2.4 Oracle Warehouse builder

Oracle Warehouse Builder [21] is a single, comprehensive tool for all aspects of data integration. Warehouse Builder leverages Oracle Database to transform data into high-quality information. It provides data quality, data auditing, fully integrated relational and dimensional modeling, and full lifecycle management of data and metadata. Warehouse Builder enables you to create data warehouses, migrate data from legacy systems,

consolidate data from disparate data sources, clean and transform data to provide quality information, and manage corporate metadata.

Oracle Warehouse Builder is comprised of a set of graphical user interfaces to assist you in implementing solutions for integrating data. In the process of designing solutions, you create various objects that are stored as metadata in a centralized repository, known as a workspace. The workspace is hosted on an Oracle Database. As a general user, you do not have full access to the workspace. Instead, you can access those workspaces to which you have been granted access. Starting the Design Center enables programmers to log in, which is the primary graphical user interface. Use the Design Center to import source objects, design ETL processes such as mappings, and ultimately define the integration solution.

A mapping is an object in which you define the flow of data from sources to targets. Based on a mapping design, Warehouse Builder generates the code required to implement the ETL logic. In a data warehousing project, for example, the integration solution is a target warehouse. In that case, the mappings you create in the Design Center ultimately define a target warehouse.

After you complete the design of a mapping and prompt Warehouse Builder to generate the code, the next step is to deploy the mapping. Deployment is the process of copying the relevant metadata and code you generated in the Design Center to a target schema. The target schema is generically defined as the Oracle Database which will execute the ETL logic you designed in the Design Center. Specifically, in a traditional data

warehousing implementation, the data warehouse is the target schema and the two terms are interchangeable.

Table 1 Comparison of ETL tools

CRITERIA	**CLOVER**	**TALEND**	**PENTAHO**
Engine design	Each component is run in a separate thread and acts as a consumer or producer.	All components are run on a single thread unless multi-threaded environment is enabled.	Multi-threaded with meta-data driven approach
Transformation Graphs	Represented as XML files and can be dynamically generated	Data transformation scripts are generated. Talend open Studio is like a code generator.	Saved as XML files.
Extract Load Transform (ELT) support	Absent	Present	Present
Large scale data support	Present	Present	Present
Semantic web support	None	Custom standalone plugins is available [7]	None

CloverETL and Pentaho do not provide any semantic web support. As a part of this thesis, a custom component was added to perform RDF conversion of data. CloverETL

provides the flexibility to develop a custom component and RDF conversion of the integrated data was possible with the use of API Apache Anything to triples (Any23) [22]. It is a library, a web service and a command line tool that extracts structured data in RDF format from a variety of Web documents. Any23 supports input in the form of HTML5 microdata, CSV etc. The challenge faced in this approach was that adding of a GUI was not permitted in CloverETL's custom component. The architecture of this framework is such that no additional GUI can be added. Since allowing a user to select the appropriate classes and properties to provide relationships between them is an essential part of this framework, this approach was not feasible.

2.3 Linked Data

Linked Data is simply about using the Web to create typed links between data from different sources. Technically, Linked Data refers to data published on the Web in such a way that it is machine-readable, its meaning is explicitly defined, it is linked to other external data sets, and can in turn be linked to from external data sets [23]. Linked Data is about using the Web to connect related data that wasn't previously linked, or using the Web to lower the barriers to linking data currently linked using other methods [24]. Figure 4 shows datasets that have been published in Linked Data format, by contributors to the Linking Open Data community project and other individuals and organizations. Linking Open Data cloud diagram (Figure 4) 2014, by Max Schmachtenberg, Christian Bizer, Anja Jentzsch and Richard Cyganiak [2]. It is based on metadata collected and curated by contributors to the Data Hub as well as on metadata extracted from a crawl of the Linked Data web conducted in April 2014 [2].

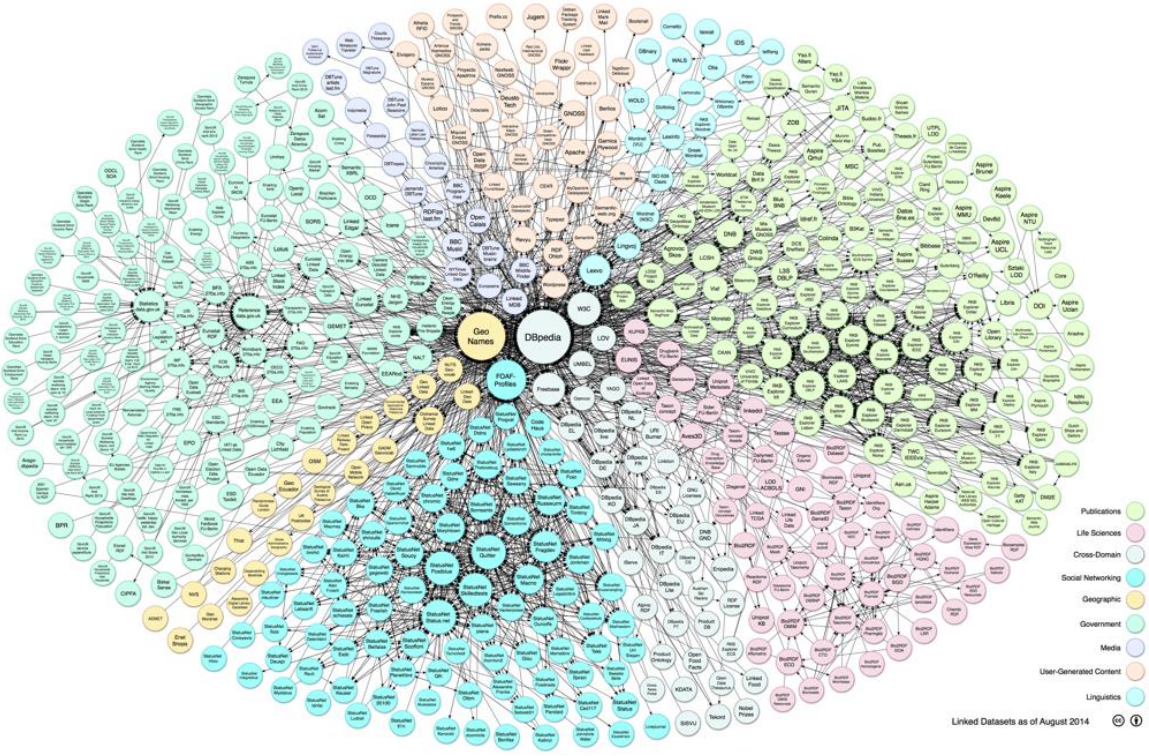


Figure 5 - LOD Cloud Diagram

The technology stack of linked data is RDF, OWL, as discussed earlier. Including links to other URIs in your own ontology is what makes it possible to discover more things and hence the term linked data. The example discussed previously under the section OWL, makes use of subClassOf property to link the user defined class to an existing schema.org class. This makes the user defined class discoverable using this link. To make the data inter-connected, W3C recommends following these rules in order to achieve high quality linked RDF data [25]:

- Use URIs as names for things
- Use HTTP URIs so that people can look up those names.

- When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)
- Include links to other URIs, so that they can discover more things.

CHAPTER 3

RELATED WORK

An extensive literature review of existing related work on semantic ETL frameworks was conducted. This section presents our findings. To achieve an end-to-end solution for the challenge of integrating data from heterogeneous sources and publishing as linked data, a number of different components are involved. In this section we present various efforts made towards this goal, analyze and identify which of these are useful towards this thesis.

3.1 Data Integration

The task of a data integration system is to provide the user with a unified view, called global schema, of a set of heterogeneous data sources. Once the user issues a query over the global schema, the system carries out the task of suitably accessing the different sources and assemble the retrieved data into the final answer to the query. In this context, a crucial issue is the specification of the relationship between the global schema and the sources, which is called mapping. Traditionally, there are two basic approaches for specifying a mapping in a data integration system. The first one, called global-as-view (GAV), requires that to each element of the global schema a view over the sources is associated. The second approach, called local-as-view (LAV), requires that to each source a view over the global schema is associated [26]. A lot of interesting work is taking place in the research community towards automating the data integration process. One such approach that has been used is a centralized global schema and a mediator to integrate data from semi-structured heterogeneous sources with large scale structured sources, focusing on Linked Open Data cloud. To make it user friendly for non-expert

users, a search interface is also provided in their proposed architecture [27]. This solution is targeted towards users who want to see the result of a particular search query in different sources. When a user searches for a term, the sources meta-data and Ontology crawler feeds information to the query distributor. With the help of adapters, the query distributor connects to different sources like Database, Web API and Linked Open Data cloud. The result set from the query distributor is handed over to a global schema composer that combines data from these different sources. The information extractor then easily extracts information from the global schema and presents the results to the user in an intuitive manner in order to hide the complexity from the user. As mentioned earlier, this framework is targeted to obtain the result set from existing data sources. It does not create new linked data.

Another related work, Karma data integration tool is a tool that aims to semi automatically aims to map structured sources onto the semantic web [28]. Using Karma, adding semantics to the data is part of the conversion process, i.e. from structured to RDF data. The tool learns the model that has been used to publish RDF data so that the semantic type need not be added every time. The model is set up by the user by importing an ontology into the system. There is also PyTransform [4] available where a user can "transform" the columns as per their business needs using simple user interface. The highlight of this tool is that it builds a graph as semantic types are set. This visualization helps in determining the correctness of the semantic types set. Using these column transformations and semantic type setting, the RDF data is generated. Karma also provides a SPARQL endpoint to query on the published data.

Data integration efforts have been towards fixed data sets. But there are some application that requires temporal data, data that varies over time. Related work in this area uses a preference aware integration of temporal data [29]. The authors have developed an operator called PRAWN (preference aware union), which is typically a final step in an entity integration workflow. It is capable of consistently integrating and resolving temporal conflicts in data that may contain multiple dimensions of time based on a set of preference rules specified by a user. First, PRAWN produces the same temporally integrated outcome, modulo representation of time, regardless of the order in which data sources are integrated. Second, PRAWN can be customized to integrate temporal data for different applications by specifying application-specific preference rules. Third, experimentally PRAWN is shown to be feasible on both "small" and "big" data platforms in that it is efficient in both storage and execution time. Finally, a fundamental advantage of PRAWN is that it can be used to pose useful temporal queries over the integrated and resolved entity repository [29].

Although there are a lot of approaches towards data integration, there is very little research work done towards benchmarking data integration. One such effort is done by authors in [30] where they propose TPC-DI, the first industry benchmark for data integration. The TPC-DI benchmark combines and transforms data extracted from a (fictitious) brokerage firm's On-Line Transaction Processing (OTLP) system along with other sources of data, and loads it into a data warehouse. The source and destination data models, data transformations and implementation rules have been designed to be broadly representative of modern data integration requirements, characterized by [30]:

- The manipulation and loading of large volumes of data

- Multiple data sources, utilizing a variety of different data formats
- Fact and dimensional table building and maintenance operations
- A mixture of transformation types including data validation, key lookups, conditional logic, data type conversions, complex aggregation operations, etc.,
- Initial loading and incremental updates of the Data Warehouse
- Consistency requirements ensuring that the data integration process results in reliable and accurate data.

3.2 Linked Data Generation

In the previous section, we discussed two studies on data integration approaches. One of the important tasks in the ETL process is linked data generation. Study by Erickson, etal [31] presents the process followed to publish linked open government data. This work is an important one in the field of Linked Data as it demonstrates the importance of integrating data from the government domain by providing a standard process in building a semantic data model and instances. The work of open linked government data establishes the importance of publishing linked data. We reviewed the work of T2LD, a prototype system for interpreting tables and extracting entities and relations from them, and producing a linked data representation of the table's contents [32]. The author has harnessed the structure of the table to convert it into linked data. Tables are a popular format of depicting large amount of domain-specific data in a structural and concise and manner. The author tries to capture the semantics of a table by mapping header cells to classes, data cell values to entities and pair of columns to relations from a given ontology

and knowledge base. The table header and rows are provided as an input to the system. A knowledge base such as Wikitology [33] is queried. An algorithm is proposed to predict the class for the column headers. Using this technique requires a knowledge base that is queried to predict the classes. Using these results, the cell value is linked to entities. A relationship is identified between the columns based on the query results and the prediction algorithm. Finally, the linked data is generated as an output. The major drawback of this proposal is the core dependency on the knowledge base. If there are any unknown entities present in the table, which does not exist in a knowledge base, linked data will not be generated.

The short comings of the proposal is addressed in TABEL, a domain independent and extensible framework for inferring the semantics of tables and representing them as RDF Linked Data [34]. TABEL framework allows for the background knowledge bases to be adapted and changed based on the domains of the table. This feature makes TABEL domain independent.

The previous work mainly focused on converting tables to linked data. A more widespread usage of data storage is relational databases. A comprehensive work of automated mapping generation for converting databases into Linked Data is discussed by Polfliet and Ichise [35]. The authors propose a method to automatically associate attribute names to existing ontology entities in order to complete the automation of the conversion of databases. They have used libraries like WordNet [36] and other java libraries to determine string equality and query a knowledge base like DBpedia [37] to query for linking the database ontology to an existing ontology.

3.3 Semantic ETL

In this section we review the research efforts made towards building semantic ETL framework.

Datalift platform [38] is one of the earlier work done towards semantic ETL framework. The process here is lifting raw data sources to semantic interlinked data sources. The process is not a traditional ETL one. The user provides structured data sources as an input to the system. To maintain a uniform format, all data sources are converted into a raw RDF format, meaning this conversion does not involve any vocabularies or entities. Once a unique data format is ready, vocabularies will be selected to assign meaning to the lifted data. Once the vocabulary selection is done, the ontology is prepared. This ontology is now used to map the elements in the RDF file, forming a fully formal RDF file. The final step in the process aims at providing links from the newly published dataset to other datasets already published as Linked Data on the Web. The interconnection modules give the possibility to achieve this linkage [38]. Linked Open Vocabulary (LOV) is used in this platform, to provide functionalities for search and also to add the generated RDF data to the LOV dataset. Another unique feature of this system is that it provides an interlinking module. Due to this, the published datasets can be linked with other datasets on the web, using sameAs property of OWL.

A more recent work towards semantic ETL framework is a python based programmable semantic ETL framework (SETL) [39]. SETL builds on Semantic Web (SW) standards and tools and supports developers by offering a number of powerful modules, classes and methods for (dimensional and semantic) data warehouse constructs and tasks. Thus it

supports semantic-aware data sources, semantic integration, and creating a semantic data warehouse, composed of an ontology and its instances [39]. The approach used here is to extract the data, transform it to RDF data using domain based ontology and loading this transformed data to a data warehouse. Before loading onto a triple store, an external linking of the RDF data is done using a semantic web search engine, Sindice [40].

Another interesting approach for ETL, although not for semantic web purposes is [41]. The authors propose a novel on demand approach to ETL, with a framework named Lenses. Data curation is a process of selective parsing, transformation and loading into a new structure. Data analysis needs to be done on-demand so that only immediately relevant tasks can be curated. On-demand curation of data is an approach where data is curated incrementally in response to specific query requirements. The authors of this work propose a standalone data processing component that unlike a typical ETL processing stage that produces a single, deterministic output, produces a PC-Table (W,P), where W is a large collection of deterministic databases, the so called possible worlds, all sharing the same schema S, and P is a probability measure over W. This table defines the set of possible outputs, and a probability measure that approximates the likelihood that any given possible output accurately models the real world [27]. A similar on demand ETL approach can be applied to the semantic ETL framework proposed in our thesis.

The challenge in data integration, be it traditional or semantic, is that it is a domain specific process. Hence a complete automation that ranges across domains is not possible. Given this, a semi-automatic approach where an intuitive UI is available to user is imperative. We have used KARMA [4] tool for this reason. The component introduced in this thesis is the linking of the ontology with online ontologies available. We use Linked

Open Vocabulary API [42] to link domain specific user defined classes to appropriate entities present on the cloud. The authors in use Sindice [40] to link their data as an external module. This thesis proposes a method to tightly integrate linking module in the integration process. Sindice is similar to LOV API. However, it is currently not being maintained.

CHAPTER 4

DATA EXTRACTION

In this thesis, research is conducted using a case study approach wherein we have developed a Semantic-driven Flights and Airlines data analysis web application, which integrates data from three different domains that are flights, holidays and weather. Flight schedule is mostly affected either by weather or holidays. The basic idea of this project is to integrate these three datasets in order to present useful results to the end user. Our web application aims at using these datasets to analyze the flight cancellation and delay patterns of different airlines. Figure 5 depicts the relation between the data sets.

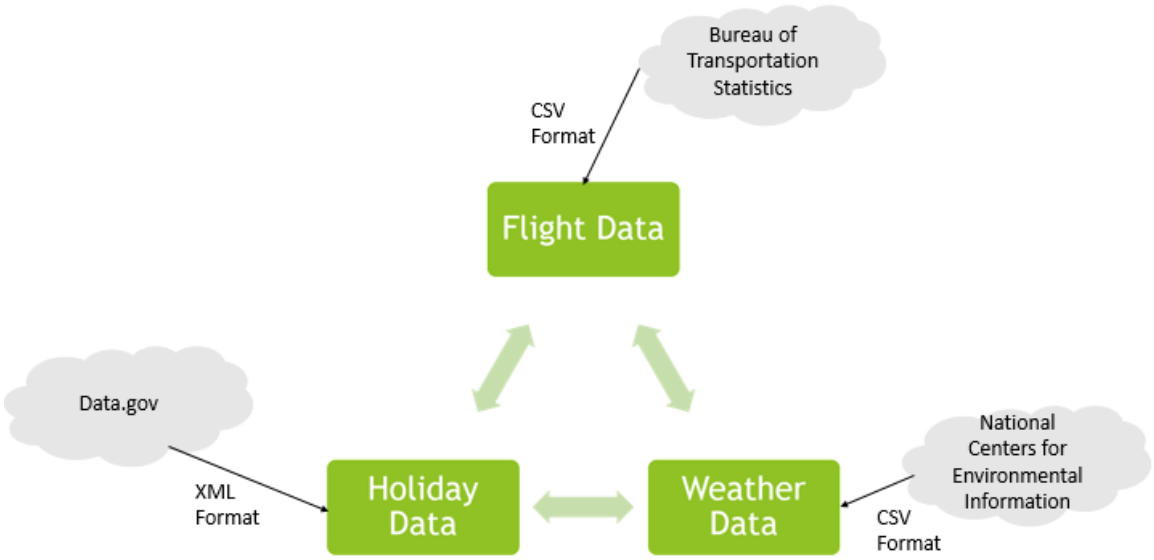


Figure 6- Relationship between datasets used and their sources

4.1 Flights Data set

The flight data is taken from the website of United States Bureau of transportation statistics [43]. The name of the dataset is “Airline On-Time performance data”. The dataset contains flight schedule data for nonstop domestic flights by major air carriers. It provides details like arrival and departure delays, origin and destination airports, as well as scheduled and actual arrival or departure times. For the scope of this project, we have limited the data to flights with origin and destination as “Phoenix” city and we have included data only for the year 2014. Even after this limitation, this dataset contains 400,000 triples for each month. The dataset is in CSV format.

The important fields used from this dataset are:

- Flight Date - date of departure of the flight
- Origin city
- Origin departure time
- Destination city
- Destination arrival time
- Arrival delay: The delay in flight arrival at destination in seconds.
- Departure delay : The delay in flight departure from origin in seconds
- Flight Number
- Cancelled: This is a Boolean value specifying whether or not the flight was cancelled.

4.2 Weather Dataset

Weather dataset is taken from the website of National Center for Environmental Information [44]. The dataset includes a monthly summary from major airport weather stations that includes a daily account of temperature extremes, degree-days, precipitation and winds. For this dataset also, we have included values only for the city of Phoenix and

for the year 2014. The size of the dataset becomes around 1500 triples. The original format of the dataset is in CSV. The fields used from this dataset are:

- Date
- Temperature
- Wind Speed

4.3 Holiday Dataset

This is a dataset listing down all the Federal datasets followed in the United States [45]. It is taken from the website of U.S. Office of Personnel Management. This dataset includes all federal holidays for the years 1997 to 2020. We have limited the scope to just the year of 2014. Hence, this dataset consists of about 40 triples. The important fields in this dataset are:

- Summary
- Start Date
- End Date

The above identified datasets were extracted from their websites in CSV format to be used in the proposed Semantic ETL framework.

CHAPTER 5

SEMANTIC DATA INTEGRATION – METHODOLOGY

5.1 Semantic Data Model

The methodology proposed in this thesis involves design and implementation of a combined semantic data model for the identified datasets. The semantic data model of the system is depicted in Figure 6. The data model has four main classes. They are: Flight, Holiday, Airline and Weather. We have also linked our ontology to the existing ontology in schema.org. As is depicted in Figure 6, all the classes are subclasses of the class "Thing". Flight is subclass of "Intangible" class, belonging to schema.org vocabulary. Object and Data properties of the classes are also depicted in the Semantic data model. Flight has an object property "hasAirline", which connects the classes Flight and Airline. Airline class is a sub-class of Organization from the schema.org vocabulary. All the other classes have many data properties. The ontology was based on this semantic data model. We have used Protégé Desktop [46] for building the OWL file. The class, object properties and data properties are encoded in OWL based on the semantic model.

The following table lists the number of elements in the ontology

Entity	**Total Number**
Class	5
Data properties	16
Object Properties	1
Relation (owl:sameAs, rdfs:sublclassOf)	2

Table 2 Number of entities in the ontology

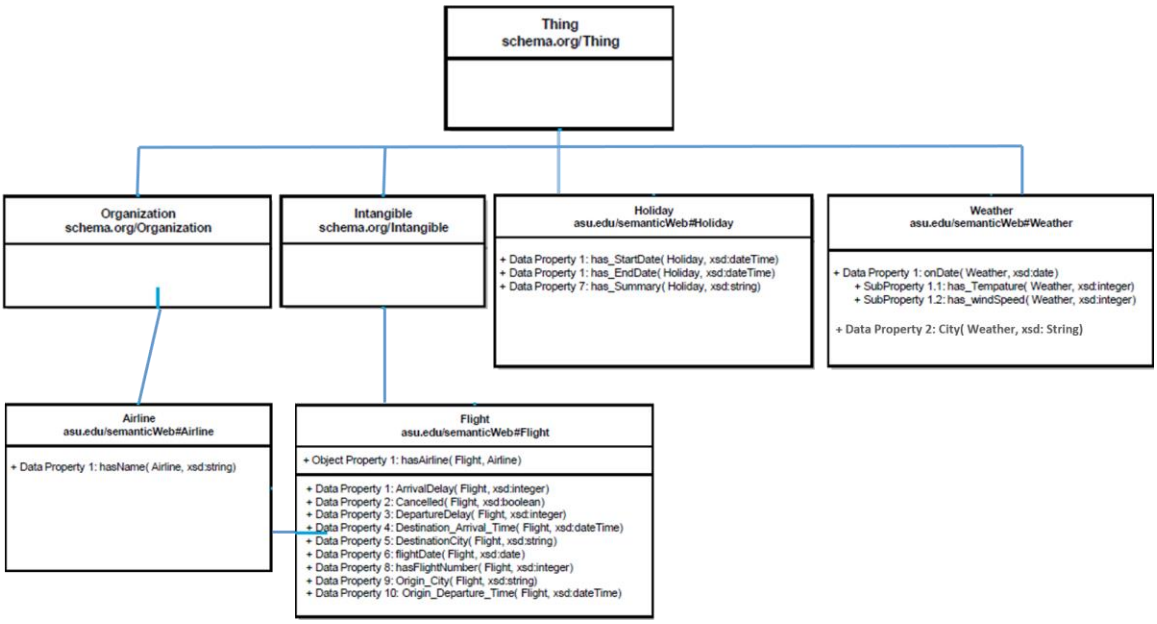


Figure 7 Semantic Data Model

5.2 Linked Data Generation

There are various tools available for generating linked data. In our work we have used Karma [4] to generate the RDF data. Karma is an intuitive tool which enables the user to build a model using user-defined ontology and the datasets to be used to generate the RDF data. The model generated is re-usable and can be used to generate RDF instances for different datasets by importing the model file. There is also a batch mode that allows user to create RDF instances in batches avoiding the manual process of repeating the same process for various subsets of data.

Another tool similar to Karma that was investigated as a part of this thesis was Google Refine [6]. Google Refine is a standalone web application with an RDF extension [48] to enable integration of data and RDF data integration. The reason Karma was chosen over Google refine was that Karma provides a better user interface to model the data as shown in Figure 7. This kind of a visualization was not provided in Google refine. The

availability of batch mode Karma allowed for automation of large flight data sets used in this thesis.

Using Karma data integration tool and OWL ontology generated using Protégé, RDF instance data was generated. The RDF generated was in turtle – Terse RDF Triple language [49] format.

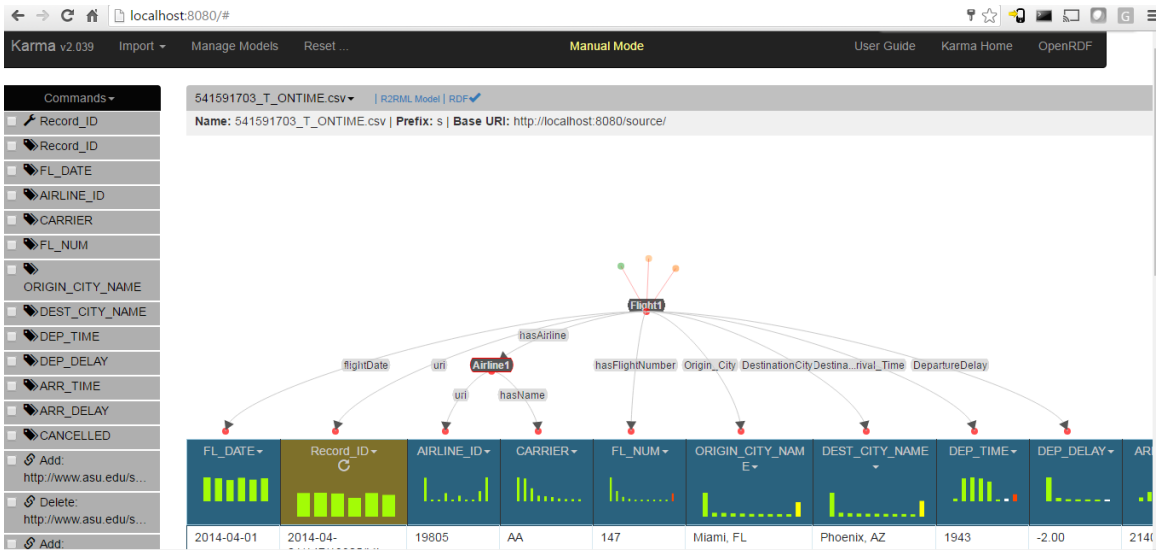


Figure 8 Data model in Karma tool

Table 3 represents the number of triples generated:

Dataset	Number of triples	Size of the dataset
Flight delays dataset	400,000*12	33 MB
Weather Dataset	1500	28 KB
Holiday Dataset	40	26 KB

Table 3 Size of Individual Datasets

Following is the snapshot of the RDF data generated for Flight data in turtle format

<http://semanticWeb/planYourTrip/date-2014-01-01-ID-AA-num-155-origin-Dallas/FortWorth_TX> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.asu.edu/semanticWeb#Flight> .

<http://semanticWeb/planYourTrip/date-2014-01-01-ID-AA-num-155-origin-Dallas/FortWorth_TX> <http://www.asu.edu/semanticWeb#Cancelled> "0.00" .

<http://semanticWeb/planYourTrip/date-2014-01-01-ID-AA-num-155-origin-Dallas/FortWorth_TX> <http://www.asu.edu/semanticWeb#flightDate> "2014-01-01" .

<http://semanticWeb/planYourTrip/19805> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.asu.edu/semanticWeb#Airline> .

<http://semanticWeb/planYourTrip/date-2014-01-01-ID-AA-num-155-origin-Dallas/FortWorth_TX> <http://www.asu.edu/semanticWeb#Destination_Arrival_Time> "1935" .

<http://semanticWeb/planYourTrip/date-2014-01-01-ID-AA-num-155-origin-Dallas/FortWorth_TX> <http://www.asu.edu/semanticWeb#ArrivalDelay> "5.00" .

<http://semanticWeb/planYourTrip/date-2014-01-01-ID-AA-num-155-origin-Dallas/FortWorth_TX> <http://www.asu.edu/semanticWeb#hasFlightNumber> "155" .

<http://semanticWeb/planYourTrip/date-2014-01-01-ID-AA-num-155-origin-Dallas/FortWorth_TX> <http://www.asu.edu/semanticWeb#Origin_City> "Dallas/Fort Worth, TX" .

<http://semanticWeb/planYourTrip/date-2014-01-01-ID-AA-num-155-origin-Dallas/FortWorth_TX> <http://www.asu.edu/semanticWeb#hasAirline> <http://semanticWeb/planYourTrip/19805> .

<http://semanticWeb/planYourTrip/date-2014-01-01-ID-AA-num-155-origin-Dallas/FortWorth_TX> <http://www.asu.edu/semanticWeb#Origin_Departure_time> "1822".

<http://semanticWeb/planYourTrip/date-2014-01-01-ID-AA-num-155-origin-Dallas/FortWorth_TX> <http://www.asu.edu/semanticWeb#DepartureDelay> "17.00" .

<http://semanticWeb/planYourTrip/date-2014-01-01-ID-AA-num-155-origin-Dallas/FortWorth_TX> <http://www.asu.edu/semanticWeb#DestinationCity> "Tucson, AZ" .

<http://semanticWeb/planYourTrip/19805> <http://www.asu.edu/semanticWeb#hasName> "AA" .

CHAPTER 6

CASE STUDY IMPLEMENTATION

6.1 High Level Design

To address the second question of our problem statement that if a richer data set can be published to the linked data cloud by use of semantic technologies in the transform phase of ETL we have designed a web application that uses three datasets from a variety of domains. Flights and Airlines data analysis app is a semantic web-based project, which aggregates data from three different domains: flights, holidays and weather. Flight schedule is mostly affected either by weather or increased number of travelers during holiday season. The basic idea of this project is to aggregate these three datasets to present consolidated results to the end user. Our web application aims at using these datasets to analyze the flight cancellation and delay patterns of different airlines. The purpose of the project is integration of these three datasets using any commonality in them and present meaningful data to the users. The 3 datasets used individually will not give much meaningful information. Hence, the goal is to use the datasets together to make a useful application.

The landing page of our system is shown in Figure 8. The main page is divided in 3 sections as the application provides three basic functionalities. The first functionality is to display flight details given the airline name and date. Figure 9 shows the result of this query on the web page. Next, the system returns the count of cancelled flights on every holiday in a year and its respective airline. The results of this query is depicted in Figure 9.

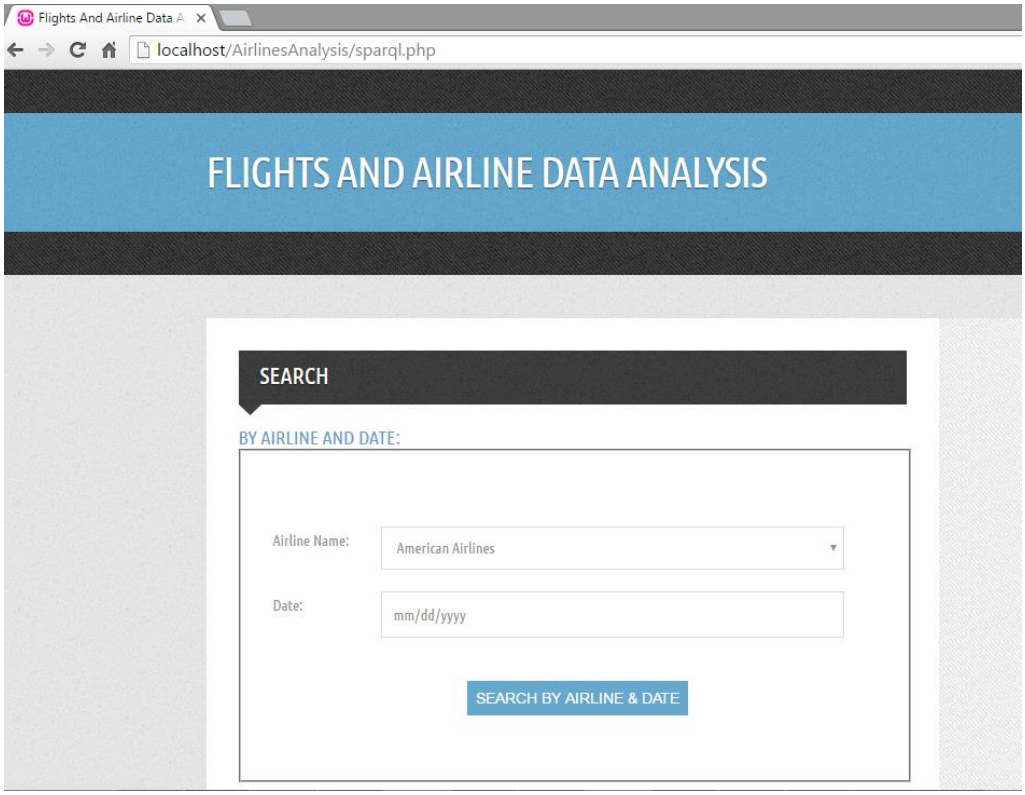
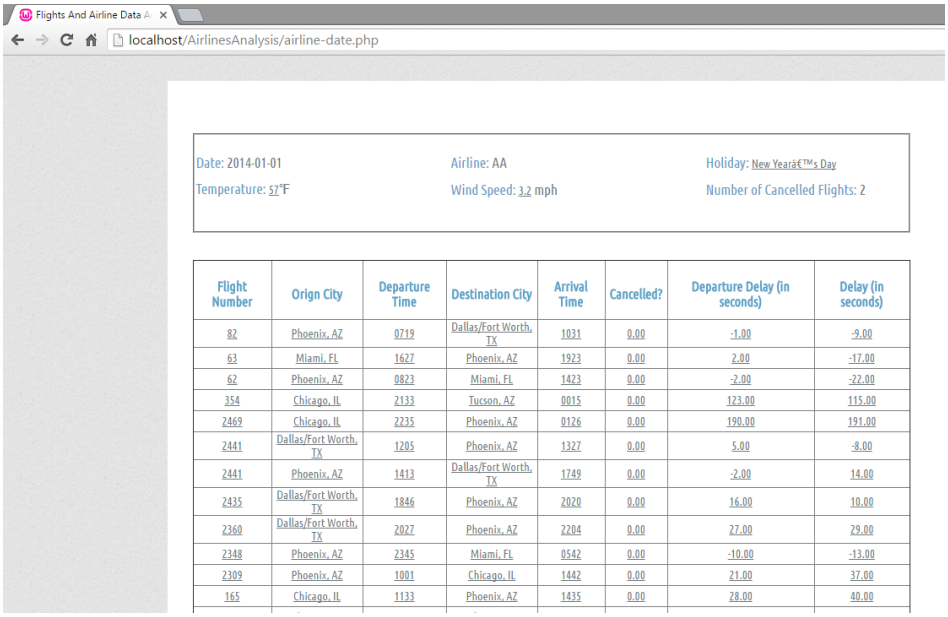


Figure 9 Landing page of web application



Flight Number	Origin City	Departure Time	Destination City	Arrival Time	Cancelled?	Departure Delay (in seconds)	Delay (in seconds)
82	Phoenix, AZ	0719	Dallas/Fort Worth, TX	1031	0.00	-1.00	-9.00
63	Miami, FL	1627	Phoenix, AZ	1923	0.00	2.00	-17.00
62	Phoenix, AZ	0823	Miami, FL	1423	0.00	-2.00	-22.00
354	Chicago, IL	2133	Tucson, AZ	0015	0.00	123.00	115.00
2469	Chicago, IL	2235	Phoenix, AZ	0126	0.00	190.00	191.00
2441	Dallas/Fort Worth, TX	1205	Phoenix, AZ	1327	0.00	5.00	-8.00
2441	Phoenix, AZ	1413	Dallas/Fort Worth, TX	1749	0.00	-2.00	14.00
2435	Dallas/Fort Worth, TX	1846	Phoenix, AZ	2020	0.00	16.00	10.00
2360	Dallas/Fort Worth, TX	2027	Phoenix, AZ	2204	0.00	27.00	29.00
2348	Phoenix, AZ	2345	Miami, FL	0542	0.00	-10.00	-13.00
2309	Phoenix, AZ	1001	Chicago, IL	1442	0.00	21.00	37.00
165	Chicago, IL	1133	Phoenix, AZ	1435	0.00	28.00	40.00

Figure 10 Result of first search

6.2 Implementation Details

The web application is developed using PHP and HTML for the front-end implementation. Easy RDF library [50] is used for accessing the SPARQL end-point and executing the SPARQL queries to present the results on the web page. Apache Jena Fuseki server [3] is the SPARQL end point that hosts the datasets in the turtle format., hence serving RDF data over HTTP. The architecture is shown in Figure 10.

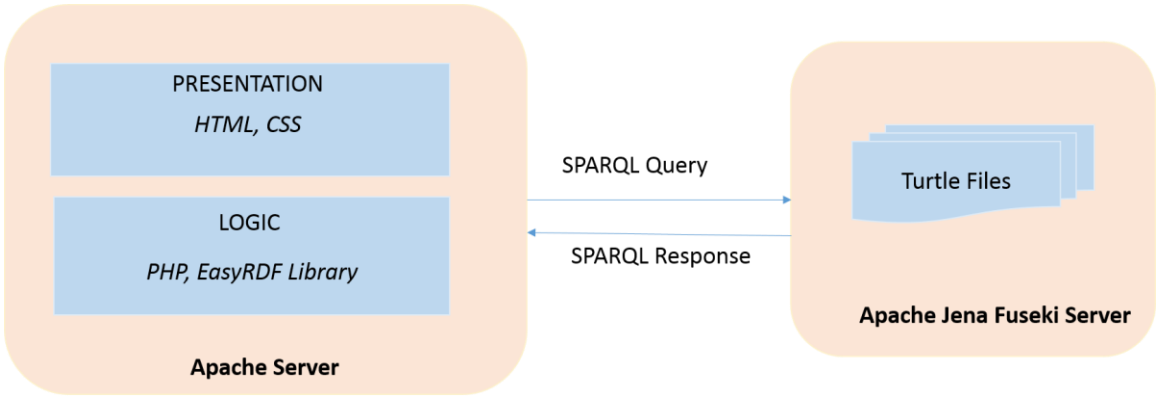


Figure 11 Architecture for Case Study

The source code for the implementation of the web application is provided in the Appendix A. Following are the SPARQL queries used in the web application.

Query 1:

The first query gets details of all the flights on a particular date on a particular airline. First, we are looking for all airlines subjects in the triples set with a flight's name equal to the airline name selected by the user "$airlineFromUser". Then we're getting all flights resources from the dataset that has the user specified airline. After we get all these flights, we are filtering the data based on the date that was entered by the user "$dateFromUser".

Finally, we are selecting the flights properties to be displayed (Origin, Destination, Delay, Cancelled, ...etc.).

```
SELECT ?delay ?depdelay ?num ?cancelled ?destTime ?originTime ?origin ?dest'.
        ' WHERE '. ' { '.
                    ' ?temp flight:ArrivalDelay ?delay;flight:DepartureDelay '.
           ' ?depdelay;flight:hasFlightNumber ?num; flight:Cancelled ?cancelled;'.
           ' flight:Destination_Arrival_Time ?destTime;flight:Origin_Departure_time '.
           ' ?originTime; flight:Origin_City ?origin; flight:DestinationCity ?dest. '.
           ' ?temp flight:flightDate "'.$dateFromUser. '" .?temp flight:hasAirline ?airline.
'.
           ' ?airline flight:hasName "'. $airlineFromUser. '";'.
        ' } '.
        ' ORDER BY DESC(?num)
```

Query 2:

The second query gets the number of flight cancellations on every holiday in a year and the respective Airline. In this query, we get all the triples from the holiday turtle file, by the data property has_EndDate. Based on the date, i.e the ?holiday object, the query fetches all the triples with predicate flightDate. Based on these triples, the cancelled "1.00" flights are selected.

Finally, the holiday's title and the count of cancelled flights and the respective airline are being displayed.

```
SELECT (COUNT(?num) AS ?count) (sample(?summary) as ?Holiday)
(sample(?airline) as ?Airline) '.
        ' WHERE '.
        ' { '.
           ' ?id flight:hasName ?airline.?temp flight:hasAirline ?id.?temp
flight:hasFlightNumber ?num. ?temp flight:Cancelled "1.00". '.
           ' ?temp flight:flightDate ?holiday.?s flight:has_Summary ?summary.?s
flight:has_EndDate ?holiday '.

        ' } '.
        ' GROUP BY ?airline ?holiday '.
        ' ORDER BY (?airline)
```

Query 3:

The third query returns the weather on a day flight was cancelled from Phoenix. Here, we are getting all the flight resources from the generated triples that were cancelled (cancelled "1.00") and all the flights subjects that have Origin city "Phoenix, AZ", along with their dates, and their respective airlines. Then from the Weather dataset, we fetch weather info (Wind Speed, and Temperature) for that particular date.

```
SELECT DISTINCT ?date ?wind ?tempr ?airline '.
        ' WHERE '.
        ' {   '.
           ' ?weather flight:has_windSpeed ?wind.?weather flight:has_Temperature
?tempr.?weather flight:onDate ?date. '.
           ' ?id flight:hasName ?airline.?temp flight:hasAirline ?id. '.
           ' ?temp flight:flightDate ?date.?temp flight:Origin_City "Phoenix, AZ".?temp
flight:Cancelled "1.00" '.
          ' } '.
         ' ORDER BY (?airline)
```

Query 4:

Following query gets the count of cancelled flights on selected holiday on selected airline

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX flight: <http://www.asu.edu/semanticWeb#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT (COUNT(?num) AS ?count)
WHERE
{
  ?temp flight:hasFlightNumber ?num. ?temp flight:Cancelled "1.00".
  ?temp flight:flightDate "2014-02-02".?temp flight:hasAirline ?airline. ?airline
flight:hasName "WN";

}ORDER BY DESC(?num)
```

Query 5:

Following query gets the number of flight cancellations on every holiday in a year

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX flight: <http://www.asu.edu/semanticWeb#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT (COUNT(?num) AS ?count) (sample(?summary) as ?title)
WHERE
{
  ?temp flight:hasFlightNumber ?num. ?temp flight:Cancelled "1.00".
  ?temp flight:flightDate ?holiday.?s flight:has_Summary ?summary.?s
flight:has_EndDate ?holiday

}GROUP BY ?holiday
```

CHAPTER 7

EVALUATION AND RESULTS

In this chapter we present our findings in terms of requirements for building a semi-automatic Semantic ETL software tool. We conduct an evaluation of the semantic ETL process and present the results.

7.1 Requirements for Semantic ETL Framework

We describe the requirements by first describing the process involved followed by the challenges and propose possible solutions to it.

7.1.1 Process

The process is mainly divided into three steps of Extract, Transform and Load.

***Extract**:*

Extract phase involves extracting data from various data sources. These data sources can be of in the form of relational tables, CSV/ XML/TXT files, data from scrapers etc. Data from these sources are extracted in a format that is conducive for transformation that is performed in the next step.

Transform:

Transform phase involves transforming the extracted data from the previous step into a format required by the user. This phase can be further divided into following steps:

Data Preparation: In this step, the data is prepared for further transformation. Since the data in extract phase can be from different data sources, processes like normalizing the

data, removing duplicates, integrity violation checks, appropriate filtering of data, sorting and grouping of data may be required.

Semantic Data Model Generation: In this phase, domain specific ontology is imported into the framework. If an ontology does not exist, a new ontology is created externally using an editor like Protégé.

Extend Linked Vocabulary: In this phase, the ontology created using Protégé editor is imported into the framework. This ontology file has to be updated with linked vocabulary existing in the linked open data cloud in order to make the RDF data as linked data. This is achieved by the use of Linked Open Vocabulary (LOV) API. A user interface is provided where user can select the most appropriate URI from top matching classes provided by the LOV API search. The ontology file is updated with classes selected by the user and linking these user-defined classes to the LOV classes by the use of owl:sameAs or rdf:subclassOf property.

Semantic Link Identification: In this phase the data is ready to be provided with semantic links. The updated ontology file generated in the previous step is used in this step. Using this ontology file, semantic types can be associated to the data and RDF instance data can be generated.

Linked Data Generation: The semantic types associated with data in the previous step is utilized and applied to each row of data. Thus RDF instance data is generated. The linked data can be generated in RDF/XML or TTL/Turtle – Terse RDF Triple Language format.

Load:

This is the final step in the process. Here, the linked data generated in the transform phase is loaded into data storage like Freebase, triple stores, graph database, or Fuseki server.

Figure 10 shows a process flow diagram for the semantic ETL framework. A complete software package should include components for all of these tasks.

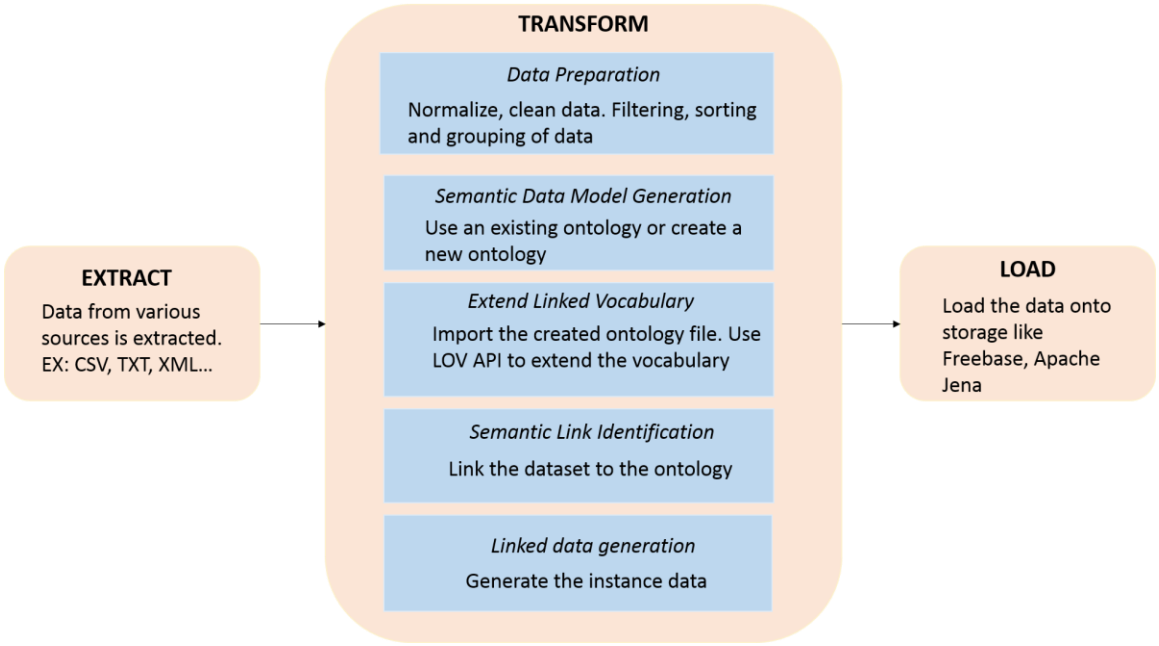


Figure 12 Semantic ETL process

7.1.2 Challenges and Proposed Solution

We have identified the following open research challenges in this process:

Domain-specific ontology for datasets under integration:

For the semantic data model generation phase, if there exists a domain-specific ontology for the datasets under integration, it can be loaded, customized, and used. In a number of cases such a domain-specific ontology may not exist at all. Our proposed solution for this

challenge is to integrate an ontology editor such as Protégé into the semantic ETL framework so that it is a one-stop tool for the entire process and provides a solution with minimal human intervention. This has not been implemented as part of this thesis and will be done as part of future work.

Extend Linked Vocabulary:

A missing component in automating the process of linked data generation is identifying and adding relationship(s) to existing classes in the Linked Open Data Cloud with the use of ontology constructs such as owl:sameAs or rdfs:subclassOf. It is the usage of these relationships that makes the generated RDF data as linked data. We propose an algorithm to automate this process. This algorithm uses Linked Open Data Vocabulary [51] API to search for terms that exist on the linked open cloud.

Following are the steps followed in this algorithm:

- Upload the ontology/owl file to be updated
- Read the owl file, recording only those lines which start with owl:class
- Read the property rdf:about to get the class name from the ontology. For example, let's consider "Flight" class from our case study.
- Use LOV API to search for term "Flight" and let the user select the most suitable related class from the results returned. The algorithm uses the URI of the selected term.
- Next, the user is asked to select the type of relationship that best fits with the identified resource from the cloud, it can be either subclassOf or sameAs.

- This process is repeated for all the classes in the domain-specific ontology. User is allowed to make selection of the most closely related terms from the LOV cloud.
- This process is repeated for all properties in the domain-specific ontology. User is allowed to make selection of the most closely related properties from the LOV Cloud and relationships such as subPropertyOf or sameAs are added.
- The ontology (OWL) file is updated followed by the instance data (RDF triples).

A proof-of-concept of this algorithm was developed as part of this thesis and is described in the next sub-section.

Creation of a Semantic ETL software tool:

One of the challenges identified through this research was the creation of a software tool that includes all components needed for the semantic ETL process. We explored the possibility of extending Clover ETL (existing popular ETL tool). The challenge faced was that though Clover ETL allows for addition of custom components to allow different kind of tasks in the various phases of ETL, it does not allow the creation of GUI-based components. A user friendly GUI is very important for the success of a semantic ETL process in order to ease burden on the user as creation of semantic data model and relationships between various terms is a highly cognitive process. So an existing ETL framework that allows extensibility through use of other GUI-based tools (for ontology editing) and libraries will have to be identified as part of future work.

7.1.3 Extend Linked Vocabulary: Proof of Concept

This algorithm is presented as a proof of concept for extending linked vocabulary by implementing it as a web application. This web application is a Java EE application that consumes LOV API into simple POJOs and displays them in the browser using Vaadin UI [52]. Wildfly server is used to deploy the web application.

The implementation process of the proof of concept can be broken down into three steps:

Model the Data

LOV API is a REST service that returns a JSON response. The JSON response returned has fairly complex data. In order to make the code more readable, a tool [53] is used to generate the model of the JSON response. The model generated is plain Java classes.

Fetch the Data

Data is fetched from the JSON response provided by the LOV API REST service. To fetch data from the web application, JAX-RS client API is used which is wrapped into a service class, JsonService.java, to keep it separate from the UI code. The only classes visible to the UI are the model classes generated in the previous step.

```
private Client client;
private WebTarget target;

   @PostConstruct
   protected void init() {
      client = ClientBuilder.newClient();
            target = client.target(
                "http://lov.okfn.org/dataset/lov/api/v2/term/search").queryParam("type",
"class");
   }
```

In the business method we only need to append the dynamic parameter and do the actual request. The dynamic parameter is the Class name selected by the user using the UI.

```
public Response getReponse(String class) {
      return target.queryParam("q", class)
            .request(MediaType.APPLICATION_JSON)
            .get(Response.class);
   }
```

In the GET method, we pass the Response class generated in the previous step. We take advantage of this class by not inspecting the response object manually. Behind the scenes, an object mapping library like Jackson is used to convert the raw JSON into the reverse engineered model generated in the previous step.

Consume the Data

Vaadin UI is used to consume the data generated from the previous step. From the UI a class can be selected. Using the REST service, the selected class is searched and the top 7 matching URIs are displayed. The selected input for all the classes are displayed in a table on the UI. The data in the table is read and updated in the ontology file appropriately.

Following figures shows the user interface of the web application. The ontology file of flights.owl generated as a part of this thesis (case study) is used in the demo.

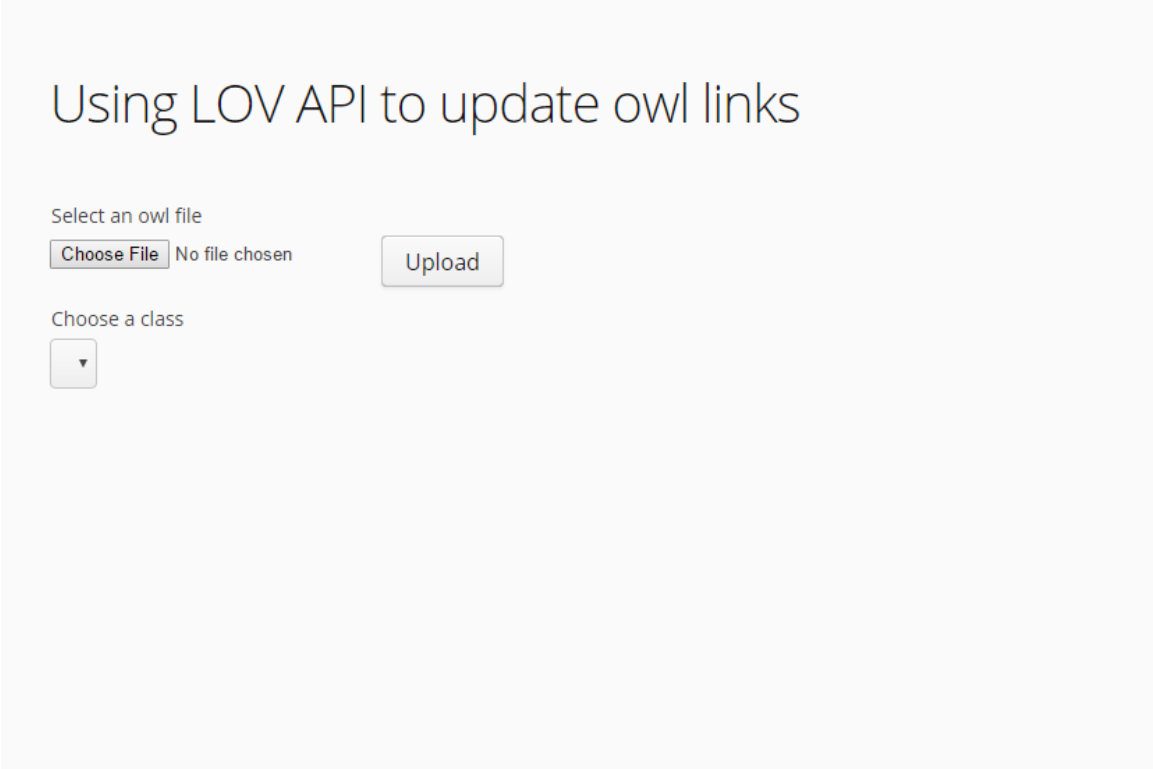


Figure 13 Landing page

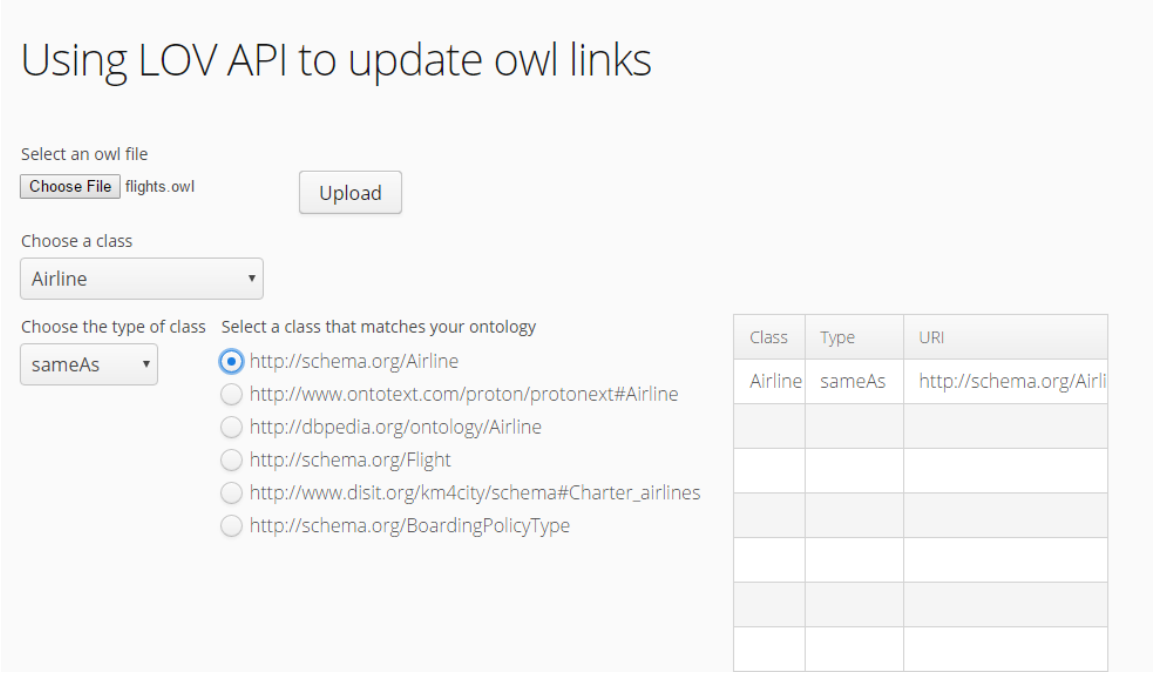


Figure 14 Page after uploading ontology file and selecting a class

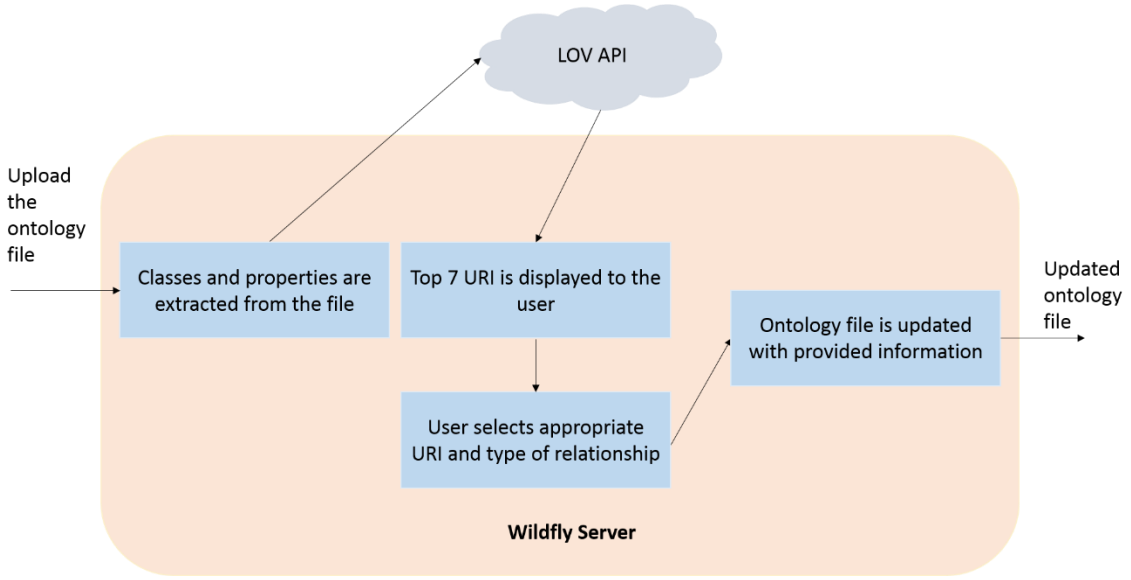


Figure 15 Extending Linked Vocabulary - Architecture

This algorithm can be used as a plug-in in any of the existing frameworks. Karma and google refine are both java-based applications. This implementation can be included in either of these frameworks to make this missing module as a tightly integrated component with the entire framework instead of a standalone one. As future work this plug-in can be included into the Semantic ETL software tool in order to have all the tasks involved in data integration packaged into one tool.

7.2 Comparison of Semantic ETL with Traditional ETL

We evaluate this thesis by revisiting and answering the two questions in the hypothesis.

RQ1: Can we introduce some level of automation in the integration, generation, and publishing of publish linked data from multiple data sources to the cloud by introducing a semantic component in the transform phase of the ETL process?

Through a case study approach we devised the requirements for a Semantic ETL framework and identified and discussed challenges in building a software tool with all components packaged into one. The case study used a manual integration process that helped us identify the challenges. Various existing tools and libraries were identified and used to ease this process. For the challenge related to identifying relationships between various classes and extending the linked open vocabulary, a proof-of-concept algorithm and web application prototype was designed and developed that can be included as a plug-in in the software tool in the future. This plug-in eliminates the manual search and creation of relationships. Hence we conclude that these findings and proof-of-concept can be used in creation of a semi-automatic software tool for semantic data integration and publishing of linked data.

RQ2: By using semantic technologies in the transform phase of ETL, can a richer data model and integrated data be published to the linked data cloud?

We address the second research question by querying the integrated data set using the semantic approach as well as traditional approach. In our case study, we linked the integrated flights, weather, and holiday data with DBPedia [54] (an existing knowledge base of RDF triples on the Linked Open Data cloud). The purpose of this to prove that by providing links in our dataset to the linked data cloud, we can achieve richer data set. SPARQL queries were written to query the integrated RDF instance data along with DBPedia already available on the LOD cloud.

We have used Apache Jena Fuseki to store the integrated RDF data and serves as a SPARQL endpoint. The following query fetches flight cancellations of for cities with a particular population or higher.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX flight: <http://www.asu.edu/semanticWeb#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX  dbpedia-owl: <http://dbpedia.org/ontology/>
SELECT (COUNT(?num) AS ?count)
WHERE {
  ?temp flight:hasFlightNumber ?num. ?temp flight:Cancelled "1.00".
  ?temp flight:flightDate "2014-02-02".?temp flight:Origin_City ?city.
  ?city rdf:type dbpedia-owl:City.
  ?city rdfs:label ?citylabel.
  ?city dbpedia-owl:country ?country.
  ?country rdfs:label ?countrylabel.
  ?city dbpedia-owl:populationTotal ?pop .
  FILTER ( lang(?countrylabel) = 'en' and lang(?citylabel) = 'en' and ?pop>10000 and
?country = 'USA')
}ORDER BY DESC(?num)
```

Such linking of data between linked data on the cloud with user-generated data cannot be achieved by traditional ETL systems like CloverETL. We prove this comparing the performance of SPARQL queries run on integrated dataset through semantic ETL and SQL queries run on integrated dataset from traditional ETL.

Traditional ETL implementation:

The database schema for the datasets discussed previously is simulated for a relational database like Postgres is shown in Figure 13. The SPARQL queries discussed in Chapter 7 is replicated in a relational database schema.

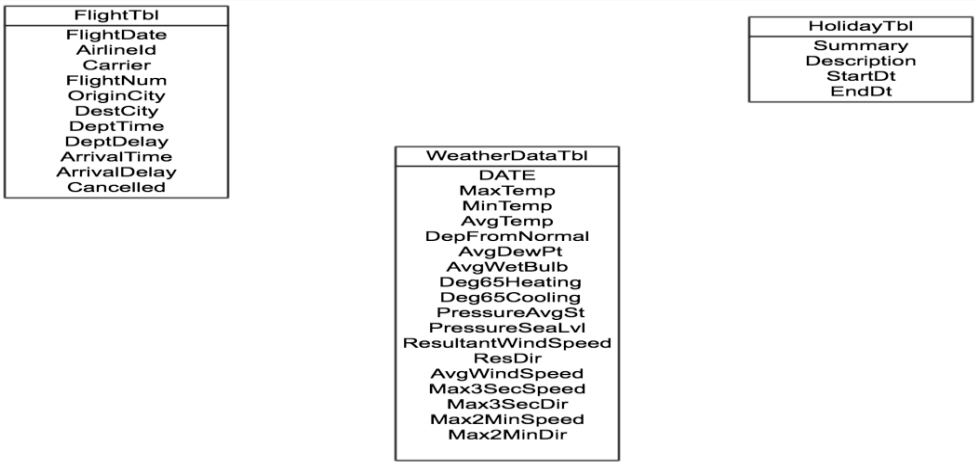


Figure 16 Database Schema

Number	Queries	Semantic ETL (in ms)	ETL (in ms)
Q1	Get details by flight name and date	179	175
Q2	Get the count of cancelled flights on selected holiday on selected airline	42	150
Q3	Number of flight cancellations on every holiday in a year	92	295
Q4	Number of flight cancellations on every holiday in a year and the respective Airline	35	284
Q5	Returns the weather on a day flight was cancelled from Phoenix. The corresponding Airline is shown	283	366

Number	Queries	Semantic ETL (in ms)	ETL (in ms)
Q6	Fetch the total number of flight cancellations for a city(in DBPedia ontology) whose population is greater than given number	387	Not possible

Table 4 Query execution time traditional vs semantic ETL

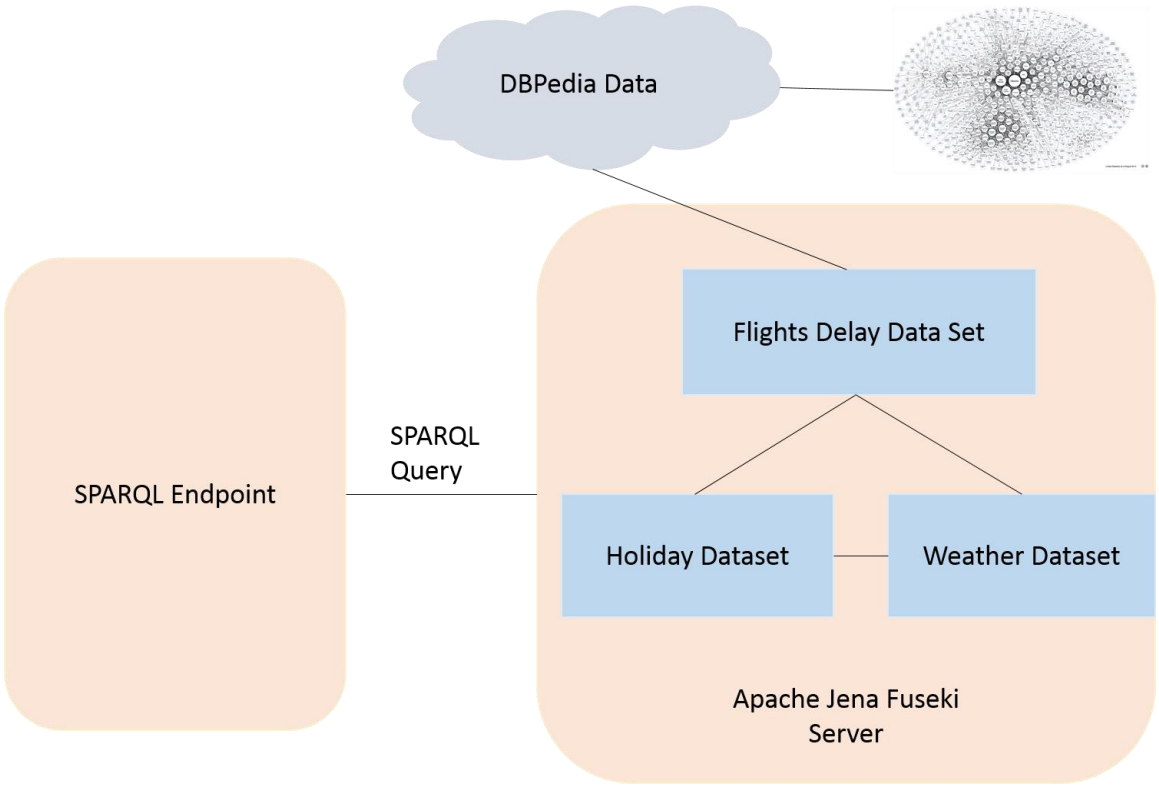


Figure 17 Data integration and linking using Semantic ETL

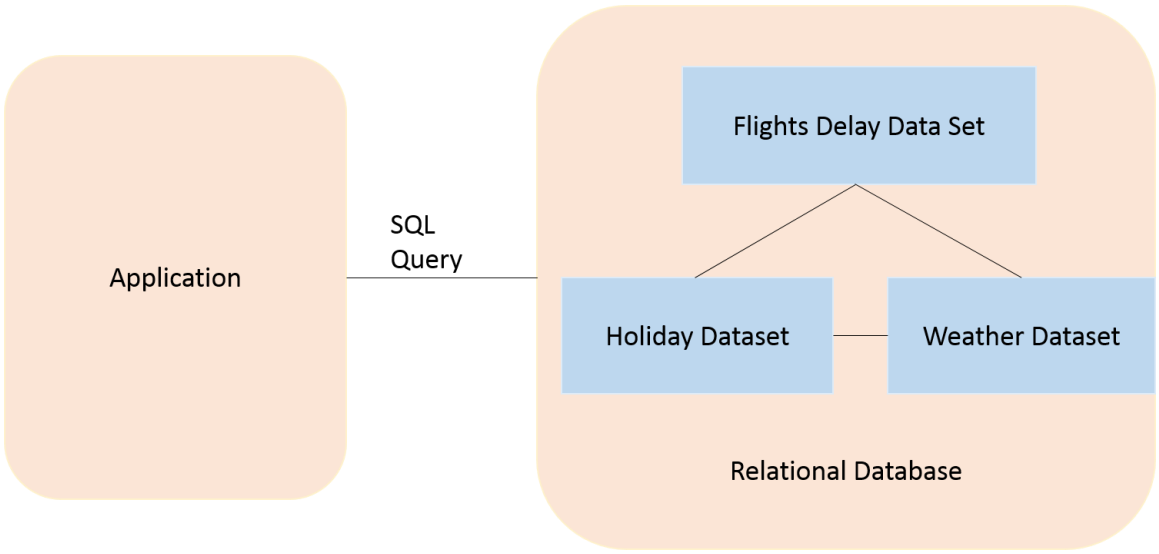


Figure 18 Data integration and linking using traditional ETL

Table 4 shows the performance of queries using traditional and semantic ETL approaches. Analysis of these results shows that not all questions are answered through the SQL queries using the traditional ETL approach. Q6 is answered through semantic ETL approach because the resultant integrated data is linked to the LOD cloud thereby providing a richer data model and dataset. Also for all queries except Q1, semantic ETL out performs the traditional ETL. For Q1 the performance of both approaches is close. These indicate that the usage of a semantic ETL framework generated a richer dataset. It allows for fetching answers from a much larger body of knowledge as opposed to just the 3 datasets that were under integration because of the Linked data principles. This is not possible with traditional ETL without manually connecting each source of data.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Conclusion

After a thorough analysis of the research work done in the area of data integration and Semantic ETL and a keen observation of the process followed in the generation of linked data, we have identified that the missing components for semantic data integration and linked data generation. They are i) extending semantic data model with linked open data vocabulary; ii) linking the generated RDF data to the linked open data cloud; iii) reducing human-loop in semantic data model generation; iv) software tool for semantic ETL with minimal manual intervention. We have presented a case study with proof-of-concept algorithms that utilizes an existing API and makes the process of linking and publishing linked data automated and tightly integrated with the entire ETL process. The solution that we have proposed is unique to the best of our knowledge.

One of the major advantages of using semantic technologies over existing techniques is that the integrated data can be published as open data on the web that is reachable for querying as opposed to traditional data stores supporting SQL that is not publicly accessible. Our hypothesis that a richer data set can be generated using semantic ETL process holds.

8.2 Future Work

The solution discussed in this thesis is a proof-of-concept with prototype implementation. However, they can easily be extended to create a complete software tool for Semantic

ETL with ability to generate semantic data model and publish linked data with minimal human intervention.

The web application has a variety of applications and a lot can be added to it to make it more useful. As of now, we have just used temperature and wind speed values from the weather dataset. As flight schedule does depend a lot on weather, more information about weather can be incorporated. Also, data corresponding to severe weather conditions can also be integrated, as severe weather leads to a lot of delays and cancellations in flight schedules. Currently the system displays cancelled flights during holiday season. This can be extended to display delayed flights as well. Also, a tabular comparison between different airlines carriers can be done on the basis of cancelled or delayed flights during holidays and severe weather conditions.

The semantic ETL framework can be extended to on-demand ETL like the one discussed in [41]. The above mentioned challenges will be addressed as part of future work.

REFERENCES

[1] D. Werner, "ETL yesterday, today and tomorrow: Something borrowed, something green," *LinkedIn Pulse*, 19-Jun-2015. [Online]. Available: https://www.linkedin.com/pulse/etl-yesterday-today-tomorrow-something-borrowed-green-dennis-werner. [Accessed: 06-Apr-2016].

[2] "The Linking Open Data cloud diagram." [Online]. Available: http://lod-cloud.net/. [Accessed: 05-May-2016].

[3] "Apache Jena - Fuseki: serving RDF data over HTTP." [Online]. Available: https://jena.apache.org/documentation/serving_data/. [Accessed: 17-Jun-2016].

[4] "Karma: A Data Integration Tool." [Online]. Available: http://usc-isi-i2.github.io/karma/. [Accessed: 17-Jun-2016].

[5] V. V. Mulwad, "TABEL–A Domain Independent and Extensible Framework for Inferring the Semantics of Tables," University of Maryland, 2015.

[6] "OpenRefine." [Online]. Available: http://openrefine.org/download.html. [Accessed: 17-Jun-2016].

[7] "fbelleau/talend4sw," *GitHub*. [Online]. Available: https://github.com/fbelleau/talend4sw. [Accessed: 06-May-2016].

[8] "tarql/tarql," *GitHub*. [Online]. Available: https://github.com/tarql/tarql. [Accessed: 23-Jun-2016].

[9] "CloverETL Rapid Data Integration." [Online]. Available: http://www.cloveretl.com/. [Accessed: 23-Jun-2016].

[10] "Talend Real-Time Open Source Data Integration Software." [Online]. Available: http://www.talend.com/. [Accessed: 23-Jun-2016].

[11] "Semantic Web - W3C." .

[12] "RDF - Semantic Web Standards." [Online]. Available: https://www.w3.org/2001/sw/wiki/RDF. [Accessed: 23-Jun-2016].

[13] "Uniform Resource Identifier - semanticweb.org.edu." [Online]. Available: http://semanticweb.org/wiki/Uniform_Resource_Identifier. [Accessed: 23-Jun-2016].

[14] "Semantic Web - W3C." [Online]. Available: https://www.w3.org/standards/semanticweb/. [Accessed: 20-Apr-2016].

[15] davidshotton, "Libraries and linked data #2: A rough guide to Turtle," *Semantic Publishing*, 01-Mar-2013. .

[16] "OWL Web Ontology Language Reference." [Online]. Available: https://www.w3.org/TR/owl-ref/. [Accessed: 23-Jun-2016].

[17] "Introducing RDFS & OWL." [Online]. Available: http://www.linkeddatatools.com/introducing-rdfs-owl. [Accessed: 21-Apr-2016].

[18] "SPARQL 1.1 Query Language." [Online]. Available: https://www.w3.org/TR/sparql11-query/. [Accessed: 23-Jun-2016].

[19] "Protege4Features - Protege Wiki." .

[20] "Data Integration | Pentaho Data Integration Platform." [Online]. Available: http://www.pentaho.com/product/data-integration. [Accessed: 23-Jun-2016].

[21] "Warehouse Builder 11gR2: Home Page on OTN." [Online]. Available: http://www.oracle.com/technetwork/developer-tools/warehouse/overview/introduction/index.html. [Accessed: 23-Jun-2016].

[22] "Apache Any23 - Apache Any23 - Introduction." [Online]. Available: https://any23.apache.org/index.html. [Accessed: 23-Jun-2016].

[23] "Linked Data - The Story So Far - bizer-heath-berners-lee-ijswis-linked-data.pdf." .

[24] “Linked Data | Linked Data - Connect Distributed Data across the Web.” [Online]. Available: http://linkeddata.org/. [Accessed: 05-May-2016].

[25] “Linked Data - Design Issues.” [Online]. Available: https://www.w3.org/DesignIssues/LinkedData.html. [Accessed: 05-May-2016].

[26] A. Calì, “Reasoning in data integration systems: why lav and gav are siblings,” in *Foundations of Intelligent Systems*, Springer, 2003, pp. 562–571.

[27] M. S. Kettouch, C. Luca, M. Hobbs, and A. Fatima, “Data integration approach for semi-structured and structured data (Linked Data),” in *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, 2015, pp. 820–825.

[28] C. A. Knoblock, P. Szekely, J. L. Ambite, A. Goel, S. Gupta, K. Lerman, M. Muslea, M. Taheriyan, and P. Mallick, “Semi-automatically mapping structured sources into the semantic web,” in *The Semantic Web: Research and Applications*, 2012, pp. 375–390.

[29] B. Alexe, M. Roth, and W.-C. Tan, “Preference-aware integration of temporal data,” *Proc. VLDB Endow.*, vol. 8, no. 4, pp. 365–376, 2014.

[30] M. Poess, T. Rabl, H.-A. Jacobsen, and B. Caufield, “TPC-DI: the first industry benchmark for data integration,” *Proc. VLDB Endow.*, vol. 7, no. 13, pp. 1367–1378, 2014.

[31] J. S. Erickson, E. Rozell, Y. Shi, J. Zheng, L. Ding, and J. A. Hendler, “TWC International Open Government Dataset Catalog,” in *Proceedings of the 7th International Conference on Semantic Systems*, New York, NY, USA, 2011, pp. 227–229.

[32] “T2LD - Table to Linked Data.pdf.” .

[33] “Wikitology: Wikipedia as an ontology.” [Online]. Available: http://ebiquity.umbc.edu/resource/html/id/245/Wikitology-Wikipedia-as-an-ontology. [Accessed: 05-May-2016].

[34] V. V. Mulwad, “TABEL -- A Domain Independent and Extensible Framework for Inferring the Semantics of Tables,” Ph.D., University of Maryland, Baltimore County, United States -- Maryland, 2015.

[35] S. Polfliet and R. Ichise, “Automated mapping generation for converting databases into linked data,” in *Proceedings of the 2010 International Conference on Posters & Demonstrations Track-Volume 658*, 2010, pp. 173–176.

[36] “About WordNet - WordNet - About WordNet.” [Online]. Available: http://wordnet.princeton.edu/. [Accessed: 06-May-2016].

[37] “DBpedia.” [Online]. Available: http://wiki.dbpedia.org/. [Accessed: 06-May-2016].

[38] F. Scharffe and S. Eurecom, “Enabling Linked Data Publication with the Datalift Platform,” *Semantics Scholar*. [Online]. Available: https://www.semanticscholar.org/paper/Enabling-Linked-Data-Publication-with-the-Datalift-Scharffe-Eurecom/0cb9f594c103bb0516ec21f4f8a76ec9cf991541. [Accessed: 21-Apr-2016].

[39] R. P. Deb Nath, K. Hose, and T. B. Pedersen, “Towards a Programmable Semantic Extract-Transform-Load Framework for Semantic Data Warehouses,” in *Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP*, New York, NY, USA, 2015, pp. 15–24.

[40] “Sindice - Semantic Web Standards.” [Online]. Available: https://www.w3.org/2001/sw/wiki/Sindice. [Accessed: 06-May-2016].

[41] Y. Yang, N. Meneghetti, R. Fehling, Z. H. Liu, and O. Kennedy, “Lenses: an on-demand approach to ETL,” *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1578–1589, 2015.

[42] “Linked Open Vocabularies (LOV).” [Online]. Available: http://lov.okfn.org/dataset/lov/api. [Accessed: 25-Jun-2016].

[43] “RITA | BTS | Transtats.” [Online]. Available: http://transtats.bts.gov/Tables.asp?DB_ID=120&DB_Name=Airline%20On-Time%20Performance%20Data&DB_Short_Name=On-Time. [Accessed: 06-May-2016].

[44] “Local Climatological Data Publication | IPS | National Climatic Data Center (NCDC).” [Online]. Available: http://www.ncdc.noaa.gov/IPS/lcd/lcd.html. [Accessed: 06-May-2016].

[45] “Federal Holidays - Data.gov.” [Online]. Available: https://catalog.data.gov/dataset/federal-holidays. [Accessed: 06-May-2016].

[46] “protégé.” [Online]. Available: http://protege.stanford.edu/products.php. [Accessed: 07-Jun-2016].

[47] R. Liepinš, M. Grasmanis, and U. Bojars, “OWLGrEd ontology visualizer,” in *Proceedings of the 2014 International Conference on Developers-Volume 1268*, 2014, pp. 37–42.

[48] “dbcls/bh11,” *GitHub*. [Online]. Available: https://github.com/dbcls/bh11. [Accessed: 17-Jun-2016].

[49] “Turtle - Terse RDF Triple Language.” [Online]. Available: https://www.w3.org/TeamSubmission/turtle/. [Accessed: 17-Jun-2016].

[50] N. Humfrey, “EasyRdf - Documentation - Getting Started.” [Online]. Available: http://www.easyrdf.org/docs/getting-started. [Accessed: 17-Jun-2016].

[51] “swj974.pdf.” .

[52] “Vaadin – User interface components for web apps,” *vaadin.com*. [Online]. Available: https://vaadin.com/home. [Accessed: 21-Jun-2016].

[53] “jsonschema2pojo.” [Online]. Available: http://www.jsonschema2pojo.org/. [Accessed: 21-Jun-2016].

[54] “DBpedia.” [Online]. Available: http://wiki.dbpedia.org/. [Accessed: 17-Jun-2016].

APPENDIX A

WEB APPLICATION SOURCE CODE

Airline-date.php – The landing page of the web application

```
<?php
   /**
    * Making a SPARQL SELECT query
    *
    * This example creates a new SPARQL client, pointing at a
    * local endpoint created using Apache Fuseki. It then makes a SELECT query that
    * returns data about flights and how are they effected by holidays and weather..
    *
    * @package    EasyRdf
    * @copyright  Copyright (c) 2009-2013 Nicholas J Humfrey
    * @license    http://unlicense.org/
    */
   set_include_path(get_include_path() . PATH_SEPARATOR . '../lib/');
   require_once "EasyRdf.php";
   require_once "html_tag_helpers.php";

   // Setup some additional prefixes
   EasyRdf_Namespace::set('rdfs', 'http://www.w3.org/2000/01/rdf-schema#');
   EasyRdf_Namespace::set('flight', 'http://www.asu.edu/semanticWeb#');
   EasyRdf_Namespace::set('owl', 'http://www.w3.org/2002/07/owl#');
   $sparql = new EasyRdf_Sparql_Client('http://localhost:3030/flights-weather-
holiday/query');

   $airlineFromUser = $_POST['airlineMenu'];
   $dateFromUser = $_POST['dateInput'];

         $quer = 'SELECT ?delay ?depdelay ?num ?cancelled ?destTime ?originTime
?origin ?dest'.
         ' WHERE '.
         ' { '.
            ' ?temp flight:ArrivalDelay ?delay;flight:DepartureDelay '.
            ' ?depdelay;flight:hasFlightNumber ?num; flight:Cancelled ?cancelled;'.
            ' flight:Destination_Arrival_Time ?destTime;flight:Origin_Departure_time '.
            ' ?originTime; flight:Origin_City ?origin; flight:DestinationCity ?dest. '.
            ' ?temp flight:flightDate "'.$dateFromUser. '" .?temp flight:hasAirline ?airline.
'.
            ' ?airline flight:hasName "'. $airlineFromUser. '";'.
         ' } '.
         ' ORDER BY DESC(?num)';

   $result = $sparql->query($quer);

   $count_cancel_query =  ' SELECT ?num '.
```

```
            ' WHERE '.
            ' { '.
              ' ?temp flight:hasFlightNumber ?num. ?temp flight:Cancelled "1.00". '.
              ' ?temp flight:flightDate "'.$dateFromUser.'".?temp flight:hasAirline
?airline. '.
              ' ?airline flight:hasName "'.$airlineFromUser.'"; '.

            ' } '.
            ' ORDER BY DESC(?num)';


   $count_cancel_result = $sparql->query($count_cancel_query);


   $count_query =  ' SELECT ?num '.
            ' WHERE '.
            ' { '.
              ' ?temp flight:hasFlightNumber ?num. ?temp flight:Cancelled "0.00". '.
              ' ?temp flight:flightDate "'.$dateFromUser.'".?temp flight:hasAirline
?airline. '.
              ' ?airline flight:hasName "'.$airlineFromUser.'"; '.

            ' } '.
            ' ORDER BY DESC(?num)';

   $count_result = $sparql->query($count_query);


   $weather_query = ' SELECT DISTINCT ?wind ?tempr '.
            ' WHERE '.
            ' {   '.
              ' ?weather flight:has_windSpeed ?wind.?weather flight:has_Temperature '.
              ' ?tempr.?weather flight:onDate "2014-01-01" '.
            ' } ';

   $wather_result = $sparql->query($weather_query);


   $holiday_query = ' SELECT DISTINCT ?summary '.
            ' WHERE '.
            ' {   '.
              ' ?holiday flight:has_Summary ?summary.?holiday flight:has_StartDate
"'.$dateFromUser.'"'.
            ' }';
```

```
    $holiday_result = $sparql->query($holiday_query);
?>

<html>
    <title>Flights And Airline Data Analysis</title>
    <head>
        <link href="css/styles.css" rel="stylesheet" type="text/css" />
        <link href='http://fonts.googleapis.com/css?family=Ubuntu+Condensed'
rel='stylesheet' type='text/css' />
    </head>
   <body>
    <div class="header-wrapper">


  <div class="logo-search-wrapper">
    <div class="logo-search-container">
      <div class="logo">Flights And Airline Data Analysis</div>

    </div>
  </div>
  </div>
<div class="white-rec">
  <div class="left-column">
    <div class="left-col-content2">

      <br><br>

        <fieldset>
        <br>
        <table border="0">
            <tr border="0">
                <td width="300" height="30">
        <p style="font-size:120%; color:#5fa1c7;display:inline">Date: </p>
        <p style="display:inline;font-size:120%"> <? echo $dateFromUser ?> </p>
                </td>

                <td width="300">
        <p style="font-size:120%; color:#5fa1c7;display:inline">Airline: </p>
        <p style="display:inline;font-size:120%"> <? echo $airlineFromUser ?> </p>
                </td>
            <?php
                if($holiday_result->numRows() == 0){
                    echo '<td>';
                    echo '<p style="font-size:120%; color:#5fa1c7;display:inline">Holiday:
</p> ';
```

```
                echo '<p style="display:inline;font-size:120%">Not a holiday!</p>';
                echo '</td> ';
                echo '</tr>';
            }

            foreach ($holiday_result as $h_row) {
                echo '<td>';
                echo '<p style="font-size:120%; color:#5fa1c7;display:inline">Holiday:
</p> ';
                echo '<p style="display:inline;font-size:120%">'.link_to($h_row-
>summary).'</p>';
                echo '</td> ';
                echo '</tr>';
            }
            echo '<tr>';

            foreach ($wather_result as $w_row) {
                echo '<td height="30">';
                echo '<p style="font-size:120%; color:#5fa1c7;display:inline">Temperature:
</p> ';
                echo ' <p style="display:inline;font-size:120%">'.link_to($w_row-
>tempr).'&#8457 </p>';
                echo '</td>';

                echo '<td>';
                echo '<p style="font-size:120%; color:#5fa1c7;display:inline">Wind Speed:
</p> ';
                echo '<p style="display:inline;font-size:120%">'.link_to($w_row->wind).  '
mph </p>';
                echo '</td>';
            }?>

          </tr>

            </table>

            <br><br>

      </fieldset>

      <br><br>

    <table style="width:100%; border-collapse: collapse;" border="1">
                <tr border="1">
                <th height="70"><h1>Flight Number</h1></th>
```

```
<th><h1>Orign City</h1></th>
        <th><h1>Departure Time</h1></th>
        <th><h1>Destination City</h1></th>
<th><h1>Arrival Time</h1></th>
<th><h1>Cancelled?</h1></th>
<th><h1>Departure Delay (in seconds)</h1></th>
<th><h1>Delay (in seconds)</h1></th>
  </tr>

            <?php
            foreach ($result as $row) {
                echo '<tr border="1" align="center">';

    echo '<td height="25">';
    echo link_to($row->num);
    echo '</td>';

    echo '<td>' ;
            echo link_to($row->origin);
            echo '</td>';

    echo '<td>';
            echo link_to($row->originTime);
            echo '</td>';

    echo '<td>';
            echo link_to($row->dest);
            echo '</td>';

    echo '<td>';
    echo link_to($row->destTime);
    echo '</td>';

    echo '<td>';
    echo link_to($row->cancelled);
    echo '</td>';

    echo '<td>';
    echo link_to($row->depdelay);
    echo '</td>';

    echo '<td>';
    echo link_to($row->delay);
    echo '</td>';
```

```
                echo '</tr>';
                        }

                        ?>

                    </table>

        <br><br><br>
        <h2>Total Number of Flights: <?= $result->numRows() ?></h2>
        <!-- <h2>Total Number of Non-Cancelled Flights: <?= $count_result->numRows();
?></h2> -->
        <!-- <h2>Total Number of Cancelled Flights: <?= $count_cancel_result-
>numRows() ?></h2> -->


        <br> <br>
    </body>
</html>
```

Cancellations.php - returns data about flights and how are they effected by holidays and weather

```
<?php
    /**
     * Making a SPARQL SELECT query
     *
     * This example creates a new SPARQL client, pointing at a
     * local endpoint created using Apache Fuseki. It then makes a SELECT query that
     * returns data about flights and how are they effected by holidays and weather..
     *
     * @package    EasyRdf
     * @copyright  Copyright (c) 2009-2013 Nicholas J Humfrey
     * @license    http://unlicense.org/
     */
    set_include_path(get_include_path() . PATH_SEPARATOR . '../lib/');
    require_once "EasyRdf.php";
    require_once "html_tag_helpers.php";
    // Setup some additional prefixes
    EasyRdf_Namespace::set('rdfs', 'http://www.w3.org/2000/01/rdf-schema#');
    EasyRdf_Namespace::set('flight', 'http://www.asu.edu/semanticWeb#');
```

```
   EasyRdf_Namespace::set('owl', 'http://www.w3.org/2002/07/owl#');
   $sparql = new EasyRdf_Sparql_Client('http://localhost:3030/flights-weather-
holiday/query');
         $airlineFromUser = $_POST['airlineMenu'];
   $dateFromUser = $_POST['dateInput'];

         $quer = ' SELECT (COUNT(?num) AS ?count) (sample(?summary) as ?Holiday)
(sample(?airline) as ?Airline) '.
         ' WHERE '.
         ' { '.
            ' ?id flight:hasName ?airline.?temp flight:hasAirline ?id.?temp
flight:hasFlightNumber ?num. '.
            ' ?temp flight:Cancelled "1.00". '.
            ' ?temp flight:flightDate ?holiday.?s flight:has_Summary ?summary.?s
flight:has_EndDate ?holiday '.

         ' } '.
         ' GROUP BY ?airline ?holiday '.
         ' ORDER BY (?airline)';

   $result = $sparql->query($quer);

?>

<html>
   <title>Flights And Airline Data Analysis</title>
   <head>
      <link href="css/styles.css" rel="stylesheet" type="text/css" />
      <link href='http://fonts.googleapis.com/css?family=Ubuntu+Condensed'
rel='stylesheet' type='text/css' />
   </head>
  <body>
   <div class="header-wrapper">


  <div class="logo-search-wrapper">
   <div class="logo-search-container">
     <div class="logo">Flights And Airline Data Analysis</div>

   </div>
  </div>
  </div>
<div class="white-rec">
  <div class="left-column">
   <div class="left-col-content2">
```

```
    <br><br>

   <table style="width:100%; border-collapse: collapse;" border="1">
                <tr border="1">
                <th height="70"><h1>Airline</h1></th>
       <th><h1>Holiday</h1></th>
                <th><h1>Number of Cancelled Flights</h1></th>
          </tr>

                        <?php
          $same_count = 1;
          $airline_code = '';
                        foreach ($result as $row) {
                                echo '<tr border="1" align="center">';

             echo '<td height=30>' ;
             echo link_to($row->Airline);
             echo '</td>' ;

             echo '<td>' ;
                        echo link_to($row->Holiday);
                        echo '</td>';

             echo '<td>';
                        echo link_to($row->count);
                        echo '</td>';

             echo '</tr>';
             $airline_code = link_to($row->Airline);
                        }

                        ?>

                </table>



    <br> <br>
  </body>
</html>
```

Weather-info.php

```
<?php
   /**
    * Making a SPARQL SELECT query
    *
    * This example creates a new SPARQL client, pointing at a
    * local endpoint created using Apache Fuseki. It then makes a SELECT query that
    * returns data about flights and how are they effected by holidays and weather..
    *
    * @package    EasyRdf
    * @copyright  Copyright (c) 2009-2013 Nicholas J Humfrey
    * @license    http://unlicense.org/
    */
   set_include_path(get_include_path() . PATH_SEPARATOR . '../lib/');
   require_once "EasyRdf.php";
   require_once "html_tag_helpers.php";
   // Setup some additional prefixes
   EasyRdf_Namespace::set('rdfs', 'http://www.w3.org/2000/01/rdf-schema#');
   EasyRdf_Namespace::set('flight', 'http://www.asu.edu/semanticWeb#');
   EasyRdf_Namespace::set('owl', 'http://www.w3.org/2002/07/owl#');
   $sparql = new EasyRdf_Sparql_Client('http://localhost:3030/flights-weather-
holiday/query');
         $airlineFromUser = $_POST['airlineMenu'];
   $dateFromUser = $_POST['dateInput'];

         $quer = ' SELECT DISTINCT ?date ?wind ?tempr ?airline '.
        ' WHERE '.
        ' {   '.
          ' ?weather flight:has_windSpeed ?wind.?weather flight:has_Temperature '.
          ' ?tempr.?weather flight:onDate ?date. '.
          ' ?id flight:hasName ?airline.?temp flight:hasAirline ?id. '.
          ' ?temp flight:flightDate ?date. '.
          ' ?temp flight:Origin_City "Phoenix, AZ".?temp flight:Cancelled "1.00" '.
         ' } '.
         ' ORDER BY (?airline) ';

   $result = $sparql->query($quer);

?>

<html>
   <title>Flights And Airline Data Analysis</title>
   <head>
      <link href="css/styles.css" rel="stylesheet" type="text/css" />
```

```
      <link href='http://fonts.googleapis.com/css?family=Ubuntu+Condensed'
rel='stylesheet' type='text/css' />
   </head>
  <body>
   <div class="header-wrapper">


  <div class="logo-search-wrapper">
   <div class="logo-search-container">
    <div class="logo"> Flights And Airline Data Analysis </div>

   </div>
  </div>
  </div>
<div class="white-rec">
  <div class="left-column">
   <div class="left-col-content2">

      <br><br>

    <table style="width:100%; border-collapse: collapse;" border="1">
                <tr border="1">
                <th height="70"><h1>Date</h1></th>
        <th><h1>Wind Speed</h1></th>
                    <th><h1>Temperature (&#8457)</h1></th>
        <th><h1>Airline</h1></th>
             </tr>

                        <?php
            $same_count = 1;
            $airline_code = '';
                        foreach ($result as $row) {
                                echo '<tr border="1" align="center">';

               echo '<td height=30>' ;
               echo link_to($row->date);
               echo '</td>' ;

               echo '<td>' ;
                            echo link_to($row->wind);
                            echo '</td>';

               echo '<td>';
                            echo link_to($row->tempr);
                            echo '</td>';
```

```
            echo '<td>' ;
            echo link_to($row->airline);
            echo '</td>' ;

            echo '</tr>';
                    }

                    ?>

                </table>



    <br> <br>
  </body>
</html>
```

APPENDIX B

PROOF OF CONCEPT SOURCE CODE

VaadinUI.java – Class responsible for the UI of the proof of concept

```
package org.example;

import com.vaadin.annotations.Theme;
import com.vaadin.cdi.CDIUI;
import com.vaadin.data.Property;
import com.vaadin.server.VaadinRequest;
import com.vaadin.ui.NativeSelect;
import com.vaadin.ui.*;
import javax.inject.Inject;
import org.vaadin.viritin.label.Header;
import org.vaadin.viritin.layouts.MVerticalLayout;
import java.io.*;
import java.io.FileOutputStream;
import java.io.OutputStream;
import org.vaadin.viritin.label.RichText;
import java.util.*;

/**
 * A simple example how to consume REST apis with JAX-RS and display that in
 * a Vaadin UI.
 */
@CDIUI("")
@Theme("valo")
public class VaadinUI extends UI {

    @Inject
    JsonService service;
    ForecastDisplay display = new ForecastDisplay();
    NativeSelect classSelector = new NativeSelect("Choose a class");
    int count = 0;



    @Override
    protected void init(VaadinRequest request) {


        classSelector.addValueChangeListener(this::changeClass);



        Upload upload = new Upload(
```

```
          "Select an owl file",
           new Upload.Receiver() {
             @Override
             public OutputStream receiveUpload(String filename,
                String mimeType) {
               FileOutputStream output = null;


               try {
                 output = new FileOutputStream("C:\\Users\\Aparna\\"
                    + filename);
               } catch (FileNotFoundException e) {
                 e.printStackTrace();
               }
               return output;
             }
           });

           upload.addListener(new Upload.FinishedListener() {
             @Override
             public void uploadFinished(Upload.FinishedEvent finishedEvent) {
               BufferedReader br = null;

               try {

                       String sCurrentLine;

                       br = new BufferedReader(new
FileReader("C:\\Users\\Aparna\\flights.owl"));
                       ArrayList<String> a = new ArrayList<String>();

                       while ((sCurrentLine = br.readLine()) != null) {
                         if(sCurrentLine.trim().startsWith("<owl:Class")){
                           String temp= sCurrentLine.split("#")[1];

classSelector.addItems(temp.split("\"")[0]);
                           a.add(temp.split("\"")[0]);
                           count++;
                         }
                       }
                       display.setCount(count);
                       display.setArray(a);
                     } catch (IOException e) {
                       e.printStackTrace();
                     } finally {
```

```
                        try {
                          if (br != null)br.close();
                        } catch (IOException ex) {
                          ex.printStackTrace();
                        }
                    }
                }
            });
            setContent(new MVerticalLayout(
                        new Header("Using LOV API to update owl links"),
                        upload,
                        classSelector,
                        display

                    )
            );
    }


    public void changeClass(Property.ValueChangeEvent e) {

        if(service != null){
          display.setForecast(service.getForecast(classSelector.getValue().toString()));
          display.setselectedClass(classSelector.getValue().toString());
        }

    }
}
```

JsonService.java

```
package org.example;

import javax.annotation.PostConstruct;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;
import org.apache.deltaspike.core.api.config.ConfigProperty;
import org.example.domain.Response;


@ApplicationScoped
public class JsonService {

   @Inject
   @ConfigProperty(name = "weathermap.apikey")
   private String apikey;

   private Client client;
   private WebTarget target;

   @PostConstruct
   protected void init() {
      client = ClientBuilder.newClient();

            target = client.target(
                  "http://lov.okfn.org/dataset/lov/api/v2/term/search").queryParam("type",
"class");
   }

   public Response getForecast(String place) {
      return target.queryParam("q", place)
            .request(MediaType.APPLICATION_JSON)
            .get(Response.class);
   }
}
```