LDIF - A Framework for Large-Scale Linked Data Integration

Andreas Schultz
Web-based Systems Group
Freie Universität Berlin,
Germany
a.schultz@fu-berlin.de

Andrea Matteini
mes|semantics
Berlin, Germany
a.matteini@mes-info.de

Robert Isele
Web-based Systems Group
Freie Universität Berlin,
Germany
mail@robertisele.com

Pablo N. Mendes
Web-based Systems Group
Freie Universität Berlin,
Germany
pablo.mendes@fu-berlin.de

Christian Bizer
Web-based Systems Group
Freie Universität Berlin,
Germany
chris@bizer.de

Christian Becker
mes|semantics
Berlin, Germany
c.becker@mes-info.de

ABSTRACT

While the Web of Linked Data grows rapidly, the development of Linked Data applications is still cumbersome and hampered due to the lack of software libraries for accessing, integrating and cleansing Linked Data from the Web. In order to make it easier to develop Linked Data applications, we provide the LDIF - Linked Data Integration Framework. LDIF can be used as a component within Linked Data applications to gather Linked Data from the Web and to translate the gathered data into a clean local target representation while keeping track of data provenance. LDIF provides a Linked Data crawler as well as components for accessing SPARQL endpoints and remote RDF dumps. It provides an expressive mapping language for translating data from the various vocabularies that are used on the Web to a consistent, local target vocabulary. LDIF includes an identity resolution component which discovers URI aliases in the input data and replaces them with a single target URI based on flexible, user-provided matching heuristics. For provenance tracking, the LDIF framework employs the Named Graphs data model. LDIF contains a data quality assessment and a data fusion module which allow Web data to be filtered according to different data quality assessment policies and provide for fusing Web data using different conflict resolution methods. In order to deal with use cases of different sizes, we provide an in-memory implementation of the LDIF framework as well as an RDF-store-backed implementation and a Hadoop implementation that can be deployed on Amazon EC2.

1. MOTIVATION

The Linking Open Data Cloud catalog[1] currently lists 326 datasets covering domains such as geographic data, government data, media, libraries and publications, life science, retail and commerce, user-generated content. As of September 2011, this data space is estimated to contain 31 billion RDF triples and around 504 million RDF links between data sources [1]. Developing Linked Data applications that exploit this data space is still cumbersome due to the heterogeneity of the Web of Linked Data and due to the lack of toolkits to access, integrate and cleanse data from this space.

Two major roadblocks for building Linked Data applications are vocabulary heterogeneity and URI aliases. A part of the Linked Data sources reuse terms from widely-deployed vocabularies to represent parts of their content describing common types of entities such as people, organizations, publications and products. Other Linked Data sources do not [1]. For domain-specific entities such as genes, pathways, descriptions of subway lines, statistical and scientific data, no agreement on common vocabularies has evolved yet. A second problem are identity links: Some data sources set *owl:sameAs* links pointing at data about the same entity in other data sources. Many other data sources do not.

In contrast to the heterogeneity of the Web, many Linked Data applications would prefer to have all data describing one class of entities being represented using the same vocabulary. Instead of being confronted with URI aliases which refer to data that might or might not describe the same entity, Linked Data applications would prefer all triples describing the same entity to have the same subject URI as this eases many application tasks including querying, aggregation and visualization.

In order to ease using Web data in the application context, it is thus advisable to translate data to a single target vocabulary (vocabulary mapping) and to replace URI aliases with a single target URI on the client side (identity resolution), before doing more sophisticated processing.

A third roadblock for Linked Data applications is the varying quality of the published data. As the Web of Linked Data is an open medium on which everybody can publish data without central control, it is natural that data is inconsistent, some sources are outdated and other sources are

[1]http://thedatahub.org/group/lodcloud

WWW2012 Developer Track, April 18-20, 2012, Lyon, France.

intensionally misleading (SPAM). Thus, before Web data is used in the application context, its quality needs to be assessed and the application needs to decide which data to trust and how to handle data conflicts.

There are various open source tools available that help application developers with either Web data access, data translation, identity resolution or data quality assessment and fusion. But up-till-now, there are hardly any scalable, integrated frameworks that cover all five tasks. With LDIF, we try to fill this gap and provide an open-source Linked Data integration framework that provides for Web data access, data translation, identity resolution, provenance tracking, as well as data quality assessment and fusion.

Figure 1 shows the schematic architecture of a Linked Data application that implement the crawling/data warehousing pattern [1] and highlights the steps of the data integration process that are supported by LDIF.

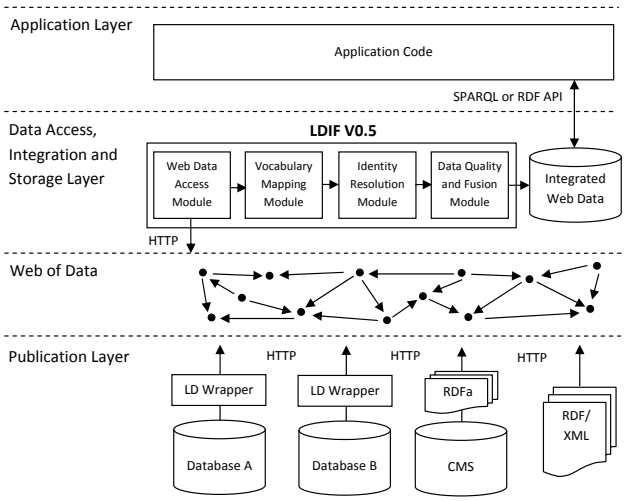

Figure 1: Role of LDIF within the architecture of a Linked Data application.

The LDIF framework is implemented in Scala and can be downloaded from the project website[2] under the terms of the Apache Software License. In the following, we explain the architecture of the LDIF framework and present a performance evaluation along the example of a life science use case.

2. ARCHITECTURE

The LDIF framework consists of a runtime environment and a set of pluggable modules. The runtime environment manages the data flows between the modules. The pluggable modules are organized in data access modules, data transformation modules and data output modules. Currently, we have implemented the following modules:

2.1 Data Access Modules and Scheduler

LDIF provides access modules for replicating Web data locally via file download, crawling or SPARQL.

[2]http://www4.wiwiss.fu-berlin.de/bizer/ldif/

These different types of import jobs all generate provenance metadata, which is passed on throughout the complete integration process. Import jobs are managed by a scheduler that can be configured to refresh the local cache hourly, daily or weekly for each source.

Triple/Quad Dump Import: The Triple/Quad Dump Importer is used to locally replicate dataset dumps from the Web. The importer supports the following RDF serialization formats: RDF/XML, N-Triples, N-Quads and Turtle.

Crawler Import: For gathering data by following RDF links, we have integrated LDSpider into LDIF. LDspider can be configured to perform crawls of different width and depth. The retrieved data from each crawled URI is put into a separate Named Graph for provenance tracking.

SPARQL Import: Data sources that can be accessed via SPARQL are replicated by LDIF's SPARQL access module. The data is accessed using SPARQL construct queries. Data from each SPARQL import job gets tracked by its own Named Graph.

The data access modules stores the gathered data in N-Quads format in a local directory. The graph URIs are used for provenance tracking. Provenance meta-information describing the origin of each graph is collected within specific provenance graph.

2.2 Data Translation

LDIF employs the R2R Framework [2] to translate Web data that is represented using terms from different vocabularies into a single target vocabulary. Vocabulary mappings are expressed using the R2R Mapping Language. The language provides for simple transformations as well as for more complex structural transformations (1-to-n and n-to-1). The language also provides for property value transformations such as normalizing different units of measurement or complex string manipulations. It also allows the user to define modifiers which make it possible to change the language or data type of a literal or the node type (URI $\leftrightarrow$ literal) of RDF nodes. The syntax of the R2R Mapping Language is very similar to the query language SPARQL, which eases the learning curve.

2.3 Identity Resolution

LDIF employs the Silk Link Discovery Framework [3] to find different URIs that are used within different data sources which identify the same real-world entity. Silk is a flexible identity resolution framework that allows the user to specify identity resolution heuristics using the declarative *Silk - Link Specification Language* (Silk-LSL). In order to specify the condition which must hold true for two entities to be considered a duplicate, the user may apply different similarity metrics, such as string, date or URI comparison methods, to multiple property values of an entity or related entities. The resulting similarity scores can be combined and weighted using various similarity aggregation functions. Silk uses a sophisticated blocking technique which removes definite non-duplicates early in the matching process increasing its efficiency significantly. For each set of duplicates which have been identified by Silk, LDIF replaces all URI aliases with a single target URI within the output data. In addi-

tion, it adds *owl:sameAs* links pointing at the original URIs, which makes it possible for applications to refer back to the data sources on the Web. If the LDIF input data already contains *owl:sameAs* links, the referenced URIs are normalized accordingly.

2.4 Data Quality Assessment and Fusion

LDIF includes the Sieve [4] Data Quality Assessment and Data Fusion Framework. The Sieve data quality assessment module assigns each Named Graph within the processed data one or several quality scores based on user-configurable quality assessment policies. These policies combine an assessment function with the definition of the quality-related meta-information which should be used in the assessment process. The Sieve data fusion module takes the quality scores as input and resolves data conflicts based on the assessment scores. The applied fusion functions can be configured on property level. A basic set of quality assessment functions and fusion functions are provided, as well as an open interface for the implementation of additional domain-specific functions.

2.5 Data Output

At the end of the integration queue, LDIF outputs the cleansed data together with the provenance information in the form of a single N-Quads file. This file contains the translated versions of all graphs that have been gathered from the Web, the content of the provenance graph as well as the quality scores for all graphs. For applications that do not require provenance and quality meta-information, LDIF can also output the data as N-Triples without meta-information.

2.6 Runtime Environment

The runtime environment manages the data flow between the various stages and the caching of the intermediate results. In order to parallelize the data processing, the data is partitioned into entity descriptions prior to supplying it to a transformation module. An entity description represents a Web resource together with all data that is required by a transformation module to process this resource. Entity descriptions consist of graph paths and include a provenance URI for each vertice. Splitting the work into fine-granular entities, allows LDIF to parallelize the work using multiple threads on a single machine as well as to parallelize processing on a cluster using Hadoop.

LDIF provides three implementations of the Runtime Environment: 1. the in-memory version, 2. the RDF store version and 3. the Hadoop version. Depending of the size of your dataset and the available computing resources, you can choose the runtime environment that best fits your use case.

Single machine/In-memory: The in-memory implementation keeps all intermediate results in memory. It is fast but its scalability is limited by the amount of available memory. For instance, integrating 25 million triples required 5 GB memory within one of our experiments. Parallelization is achieved by distributing the work to multiple threads.

Single machine/RDF Store: This implementation of the runtime environment uses Apache Jena TDB[3] to store intermediate results. The communication between the RDF store and the runtime environment is realized in the form of SPARQL queries. This runtime environment allows you to process datasets that do not fit into memory. The downside is that the RDF Store implementation is slower than the in-memory implementation.

Cluster/Hadoop: This implementation of the runtime environment allows you to parallelize the work onto multiple machines using Hadoop. Each phase in the integration flow has been ported to be executable on a Hadoop cluster. The Hadoop implementation allows you to integrate arbitrary amounts of data given that you have access to enough machines. We have tested the Hadoop implementation on a local cluster as well as on Amazon EC2.

3. PERFORMANCE EVALUATION

We evaluated the performance of LDIF using datasets ranging from 25 million quads to 3.6 billion quads. The experiments were run on local machines with a Intel i7 950, 3.07GHz (4 cores) processor and 24GB memory as well as on Amazon EC2 using between 8 and 32 c1.medium instances as worker nodes. The results of the evaluation can be found at `http://www.assembla.com/spaces/ldif/wiki/Benchmark`.

The evaluation showed that the performance of the Hadoop implementation scales linearly with the number of machines. For integrating 25 million triples, the in-memory implementation required 6.5 minutes, while the triple store implementation required 29.7 minutes. Integrating 100 million triples took the in-memory implementation 75 minutes, while the triple store implementation required 220 minutes. 300 million triples were integrated on Amazon EC2 in 75 minutes using 32 worker nodes.

4. ACKNOWLEDGMENTS

This work was supported by Vulcan Inc. as part of Project Halo (www.projecthalo.com) and by the EU FP7 project LOD2 - Creating Knowledge out of Interlinked Data (http://lod2.eu/, Ref. No. 257943).

5. REFERENCES

[1] Heath, T., Bizer C.: Linked Data: Evolving the Web into a Global Data Space. Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool Publishers, ISBN 978160845431, 2011.

[2] Bizer, C., Schultz, A.: The R2R Framework: Publishing and Discovering Mappings on the Web. 1st International Workshop on Consuming Linked Data (COLD 2010), Shanghai, November 2010.

[3] Isele, R., Jentzsch, A., Bizer, B.: Silk Server - Adding missing Links while consuming Linked Data. 1st International Workshop on Consuming Linked Data (COLD 2010), Shanghai, November 2010.

[4] Mendes, P., Mühleisen, H., Bizer, C.: Sieve - Linked Data Quality Assessment and Fusion. 2nd International Workshop on Linked Web Data Management (LWDM 2012), Berlin, March 2012.

[3] http://incubator.apache.org/jena/documentation/tdb/