LDIF - Linked Data Integration Framework

Andreas Schultz[1], Andrea Matteini[2], Robert Isele[1], Christian Bizer[1], and Christian Becker[2]

1. Web-based Systems Group, Freie Universität Berlin, Germany
a.schultz@fu-berlin.de, mail@robertisele.com, chris@bizer.de

2. MediaEvent Services GmbH & Co. KG, Germany
a.matteini@mes-info.de, c.becker@mes-info.de

Abstract. The Linked Data Integration Framework can be used within Linked Data applications to translate heterogeneous data from the Web of Linked Data into a clean local target representation while keeping track of data provenance. LDIF provides an expressive mapping language for translating data from the various vocabularies that are used on the Web into 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 user-provided matching heuristics. For provenance tracking, the LDIF framework employs the Named Graphs data model. This paper describes the architecture of the LDIF framework and presents a performance evaluation of a life science use case.

Keywords: Linked Data, Data Integration, Data Translation, Identity Resolution

1 Motivation

The Web of Linked Data grows rapidly but the development of Linked Data applications is still cumbersome due to the heterogeneity of the Web of Linked Data. Two major roadblocks for Linked Data applications are vocabulary heterogeneity and URI aliases. A fair portion of the Linked Data sources reuse terms from widely-deployed vocabularies to describe common types of entities such as people, organizations, publications and products. For more specialized, domain-specific entities, such as genes, pathways, descriptions of subway lines, statistical and scientific data, no wide-spread vocabulary agreement has evolved yet. Data sources in these domains thus use proprietary terms [1]. 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 [1].

In contrast to the heterogeneity of the Web, it is beneficial in the application context 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 any more sophisticated processing.

There are various open source tools available that help application developers with either data translation or identity resolution. But to date, there are hardly any integrated frameworks available that cover both tasks. With LDIF, we try to fill this gap and provide an open-source Linked Data integration framework that provides for data translation and identity resolution while keeping track of data provenance.

LDIF is implemented in Scala and can be downloaded from the project website[1] 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 as data access components, data transformation components and data output components. So far, we have implemented the following modules:

2.1 Data Access: N-Quads Loader

The current version of LDIF expects input data to be represented as Named Graphs and be stored in N-Quads format. The graph URI is used for provenance tracking. Provenance meta-information describing the graphs can be provided as part of the input data within a specific provenance graph. The name of this provenance graph can be set in the LDIF configuration file. LDIF does not make any assumptions about the provenance vocabulary that is used to describe the graphs, meaning that you can use your provenance vocabulary of choice.

2.2 Transformation: R2R 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

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

transformations (1-to-n and n-to-1), property value transformations such as normalizing different units of measurement or string manipulations. *Modifiers* make it possible to change the language tag or data type of a literal or the RDF node type (URI $\leftrightarrow$ literal). The R2R Mapping Language syntax is very similar to the SPARQL query language, which eases the learning curve. The expressivity of the language enabled us to deal with all requirements that we have encountered so far when translating Linked Data from the Web into a target representation [2].

2.3 Transformation: Silk Identity Resolution

LDIF employs the Silk Link Discovery Framework [3] to find different URIs 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 Link Specification Language provides a variety of data transformations to normalize the data prior to comparing it. The resulting similarity scores can be combined and weighted using various similarity aggregation functions.

Silk uses a novel blocking approach which removes definite non-duplicates early in the matching process, thereby significantly increasing its efficiency. 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 addition, it adds *owl:sameAs* links pointing at the original URIs, which makes it possible for applications to refer back to the original data sources on the Web. If the LDIF input data already contains *owl:sameAs* links, the referenced URIs are normalized accordingly.

2.4 Data Output: N-Quads Writer

The N-Quads writer dumps the final output of the integration work flow into a single N-Quads file. This file contains the translated versions of all graphs from the input graph set as well as the contents of the provenance graph.

2.5 Runtime Environment

The runtime environment manages the data flow between the modules and the caching of the intermediate results. In order to parallelize processing, data is partitioned into entities prior to supplying it to a transformation module. An entity represents a Web resource together with all data that is required by a transformation module to process this resource. Entities consist of one or more graph paths and include a provenance URI for each node. Each transformation module specifies which paths should be included into the entities it processes. By splitting the data set into fine-grained entities, LDIF is able to parallelize the workload on machines with multiple cores. In the next release, it will allow the workload to be parallelized on multiple machines using Hadoop.

3 Performance Evaluation

We evaluated the performance of LDIF using two life science data sets: KEGG GENES, a collection of gene catalogs generated from publicly available resources, and UniProt, a data set containing protein sequence, genes and functions.

We defined R2R mappings for translating genes, diseases and pathways from KEGG GENES and genes from UniProt into a proprietary target vocabulary[2]. We defined Silk linkage rules for identifying equivalent genes in both datasets. For the benchmark, we generated subsets of both data sources together amounting to 25 million, 50 million, and 100 million quads. The R2R mappings, Silk linkage rules as well as the evaluation data sets can be downloaded from the LDIF website.

We ran the performance tests on a machine with an Intel i7 950, 3.07GHz (4 cores) processor and 24GB of memory out of which we assigned 20GB to LDIF.

Table 1 summarizes the LDIF runtimes for the different data set sizes. The overall runtime is split according to the different processing steps of the integration process.

Table 1. Runtimes of the integration process for different input data set sizes.

	25M	50M	100M
Load and build entites for R2R	128.1 *sec*	297.2 *sec*	1059.7 *sec*
R2R data translation	169.9 *sec*	515.0 *sec*	1109.2 *sec*
Build entities for Silk	15.3 *sec*	36.8 *sec*	107.4 *sec*
Silk Identity Resolution	103.0 *sec*	568.5 *sec*	2954.9 *sec*
Final URI rewriting	8.1 *sec*	27.0 *sec*	65.0 *sec*
Overall execution time	7.0 *min*	24.0 *min*	88.3 *min*

References

1. Bizer, C., Jentzsch, A., Cyganiak, R.: State of the LOD Cloud. http://www4.wiwiss.fu-berlin.de/lodcloud/state/, August 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.

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