Introduction to LOD2

Sören Auer(✉)

University of Bonn, Bonn, Germany
auer@informatik.uni-bonn.de

Abstract. In this introductory chapter we give a brief overview on the Linked Data concept, the Linked Data lifecycle as well as the LOD2 Stack – an integrated distribution of aligned tools which support the whole life cycle of Linked Data from extraction, authoring/creation via enrichment, interlinking, fusing to maintenance. The stack is designed to be versatile; for all functionality we define clear interfaces, which enable the plugging in of alternative third-party implementations. The architecture of the LOD2 Stack is based on three pillars: **(1)** Software integration and deployment using the Debian packaging system. **(2)** Use of a central SPARQL endpoint and standardized vocabularies for knowledge base access and integration between the different tools of the LOD2 Stack. **(3)** Integration of the LOD2 Stack user interfaces based on REST enabled Web Applications. These three pillars comprise the methodological and technological framework for integrating the very heterogeneous LOD2 Stack components into a consistent framework.

The Semantic Web activity has gained momentum with the widespread publishing of structured data as RDF. The Linked Data paradigm has therefore evolved from a practical research idea into a very promising candidate for addressing one of the biggest challenges in the area of intelligent information management: the exploitation of the Web as a platform for data and information integration as well as for search and querying. Just as we publish unstructured textual information on the Web as HTML pages and search such information by using keyword-based search engines, we are already able to easily publish structured information, reliably interlink this information with other data published on the Web and search the resulting data space by using more expressive querying beyond simple keyword searches. The Linked Data paradigm has evolved as a powerful enabler for the transition of the current document-oriented Web into a Web of interlinked Data and, ultimately, into the Semantic Web. The term Linked Data here refers to a set of best practices for publishing and connecting structured data on the Web. These best practices have been adopted by an increasing number of data providers over the past three years, leading to the creation of a global data space that contains many billions of assertions – the Web of Linked Data (cf. Fig. 1).

In that context LOD2 targets a number of research challenges: improve coherence and quality of data published on the Web, close the performance gap between relational and RDF data management, establish trust on the Linked

S. Auer et al. (Eds.): Linked Open Data, LNCS 8661, pp. 1–17, 2014.
DOI: 10.1007/978-3-319-09846-3_1

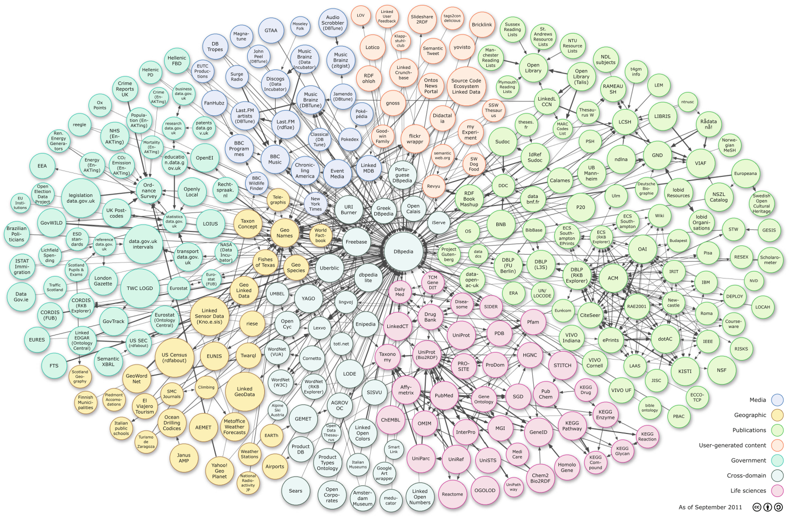


Fig. 1. Overview of some of the main Linked Data knowledge bases and their interlinks available on the Web. (This overview is published regularly at http://lod-cloud.net and generated from the Linked Data packages described at the dataset metadata repository ckan.net.)

Data Web and generally lower the entrance barrier for data publishers and users. The LOD2 project tackles these challenges by developing:

- enterprise-ready tools and methodologies for exposing and managing very large amounts of structured information on the Data Web.
- a testbed and bootstrap network of high-quality multi-domain, multi-lingual ontologies from sources such as Wikipedia and OpenStreetMap.
- algorithms based on machine learning for automatically interlinking and fusing data from the Web.
- adaptive tools for searching, browsing, and authoring of Linked Data.

The LOD2 project integrates and syndicates linked data with large-scale, existing applications and showcases the benefits in the three application scenarios publishing, corporate data intranets and Open Government Data.

The main result of LOD2 is the LOD2 Stack[1] – an integrated distribution of aligned tools which support the whole life cycle of Linked Data from extraction, authoring/creation via enrichment, interlinking, fusing to maintenance. The LOD2 Stack comprises new and substantially extended existing tools from the LOD2 partners and third parties. The major components of the LOD2 Stack are open-source in order to facilitate wide deployment and scale to knowledge bases with billions of triples and large numbers of concurrent users. Through

[1] After the end of the project, the stack will be called Linked Data Stack and maintained by other projects, such as GeoKnow and DIACHRON.

an agile, iterative software development approach, we aim at ensuring that the stack fulfills a broad set of user requirements and thus facilitates the transition to a Web of Data. The stack is designed to be versatile; for all functionality we define clear interfaces, which enable the plugging in of alternative third-party implementations. We also plan a stack configurer, which enables potential users to create their own personalized version of the LOD2 Stack, which contains only those functions relevant for their usage scenario. In order to fulfill these requirements, the architecture of the LOD2 Stack is based on three pillars:

- *Software integration and deployment using the Debian packaging system.* The Debian packaging system is one of the most widely used packaging and deployment infrastructures and facilitates packaging and integration as well as maintenance of dependencies between the various LOD2 Stack components. Using the Debian system also allows to facilitate the deployment of the LOD2 Stack on individual servers, cloud or virtualization infrastructures.
- *Use of a central SPARQL endpoint and standardized vocabularies for knowledge base access and integration between different tools.* All components of the LOD2 Stack access this central knowledge base repository and write their findings back to it. In order for other tools to make sense out of the output of a certain component, it is important to define vocabularies for each stage of the Linked Data life-cycle.
- *Integration of the LOD2 Stack user interfaces based on REST enabled Web Applications.* Currently, the user interfaces of the various tools are technologically and methodologically quite heterogeneous. We do not resolve this heterogeneity, since each tool's UI is specifically tailored for a certain purpose. Instead, we develop a common entry point for accessing the LOD2 Stack UI, which then forwards a user to a specific UI component provided by a certain tool in order to complete a certain task.

These three pillars comprise the methodological and technological framework for integrating the very heterogeneous LOD2 Stack components into a consistent framework. This chapter is structured as follows: After briefly introducing the linked data life-cycle in Sect. 1 and the linked data paradigm in Sect. 2, we describe these pillars in more detail (Sect. 3), and conclude in Sect. 4.

1 The Linked Data Life-Cycle

The different stages of the Linked Data life-cycle (depicted in Fig. 2) include:

Storage. RDF Data Management is still more challenging than relational Data Management. We aim to close this performance gap by employing column-store technology, dynamic query optimization, adaptive caching of joins, optimized graph processing and cluster/cloud scalability.

Authoring. LOD2 facilitates the authoring of rich semantic knowledge bases, by leveraging Semantic Wiki technology, the WYSIWYM paradigm (What You See Is What You Mean) and distributed social, semantic collaboration and networking techniques.

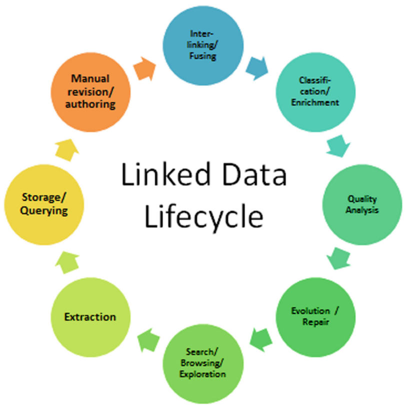


Fig. 2. Stages of the Linked Data life-cycle supported by the LOD2 Stack.

Interlinking. Creating and maintaining links in a (semi-)automated fashion is still a major challenge and crucial for establishing coherence and facilitating data integration. We seek linking approaches yielding high precision and recall, which configure themselves automatically or with end-user feedback.

Classification. Linked Data on the Web is mainly raw instance data. For data integration, fusion, search and many other applications, however, we need this raw instance data to be linked and integrated with upper level ontologies.

Quality. The quality of content on the Data Web varies, as the quality of content on the document web varies. LOD2 develops techniques for assessing quality based on characteristics such as provenance, context, coverage or structure.

Evolution/Repair. Data on the Web is dynamic. We need to facilitate the evolution of data while keeping things stable. Changes and modifications to knowledge bases, vocabularies and ontologies should be transparent and observable. LOD2 also develops methods to spot problems in knowledge bases and to automatically suggest repair strategies.

Search/Browsing/Exploration. For many users, the Data Web is still invisible below the surface. LOD2 develops search, browsing, exploration and visualization techniques for different kinds of Linked Data (i.e. spatial, temporal, statistical), which make the Data Web sensible for real users.

These life-cycle stages, however, should not be tackled in isolation, but by investigating methods which facilitate a mutual fertilization of approaches developed to solve these challenges. Examples for such mutual fertilization between approaches include:

- The detection of mappings on the schema level, for example, will directly affect instance level matching and vice versa.
- Ontology schema mismatches between knowledge bases can be compensated for by learning which concepts of one are equivalent to which concepts of another knowledge base.

- Feedback and input from end users (e.g. regarding instance or schema level mappings) can be taken as training input (i.e. as positive or negative examples) for machine learning techniques in order to perform inductive reasoning on larger knowledge bases, whose results can again be assessed by end users for iterative refinement.
- Semantically enriched knowledge bases improve the detection of inconsistencies and modelling problems, which in turn results in benefits for interlinking, fusion, and classification.
- The querying performance of RDF data management directly affects all other components, and the nature of queries issued by the components affects RDF data management.

As a result of such interdependence, we should pursue the establishment of an improvement cycle for knowledge bases on the Data Web. The improvement of a knowledge base with regard to one aspect (e.g. a new alignment with another interlinking hub) triggers a number of possible further improvements (e.g. additional instance matches).

The challenge is to develop techniques which allow exploitation of these mutual fertilizations in the distributed medium Web of Data. One possibility is that various algorithms make use of shared vocabularies for publishing results of mapping, merging, repair or enrichment steps. After one service published its new findings in one of these commonly understood vocabularies, notification mechanisms (such as *Semantic Pingback* [11]) can notify relevant other services (which subscribed to updates for this particular data domain), or the original data publisher, that new improvement suggestions are available. Given proper management of provenance information, improvement suggestions can later (after acceptance by the publisher) become part of the original dataset.

The use of Linked Data offers a number of significant benefits:

- *Uniformity.* All datasets published as Linked Data share a uniform data model, the RDF statement data model. With this data model all information is represented in facts expressed as triples consisting of a subject, predicate and object. The elements used in subject, predicate or object positions are mainly globally unique identifiers (IRI/URI). Literals, i.e., typed data values, can be used at the object position.
- *De-referencability.* URIs are not just used for identifying entities, but since they can be used in the same way as URLs they also enable locating and retrieving resources describing and representing these entities on the Web.
- *Coherence.* When an RDF triple contains URIs from different namespaces in subject and object position, this triple basically establishes a link between the entity identified by the subject (and described in the source dataset using namespace A) with the entity identified by the object (described in the target dataset using namespace B). Through the typed RDF links, data items are effectively interlinked.
- *Integrability.* Since all Linked Data sources share the RDF data model, which is based on a single mechanism for representing information, it is very easy to attain a syntactic and simple semantic integration of different Linked Data

sets. A higher level semantic integration can be achieved by employing schema and instance matching techniques and expressing found matches again as alignments of RDF vocabularies and ontologies in terms of additional triple facts.

- *Timeliness.* Publishing and updating Linked Data is relatively simple thus facilitating a timely availability. In addition, once a Linked Data source is updated it is straightforward to access and use the updated data source, since time consuming and error prune extraction, transformation and loading is not required.

Table 1. Juxtaposition of the concepts Linked Data, Linked Open Data and Open Data.

Representation\degree of openness	Possibly closed	Open (cf. opendefinition.org)
Structured data model (i.e. XML, CSV, SQL etc.)	**Data**	**Open Data**
RDF data model (published as Linked Data)	**Linked Data** (LD)	**Linked Open Data** (LOD)

The development of research approaches, standards, technology and tools for supporting the Linked Data lifecycle data is one of the main challenges. Developing adequate and pragmatic solutions to these problems can have a substantial impact on science, economy, culture and society in general. The publishing, integration and aggregation of statistical and economic data, for example, can help to obtain a more precise and timely picture of the state of our economy. In the domain of health care and life sciences making sense of the wealth of structured information already available on the Web can help to improve medical information systems and thus make health care more adequate and efficient. For the media and news industry, using structured background information from the Data Web for enriching and repurposing the quality content can facilitate the creation of new publishing products and services. Linked Data technologies can help to increase the flexibility, adaptability and efficiency of information management in organizations, be it companies, governments and public administrations or online communities. For end-users and society in general, the Data Web will help to obtain and integrate required information more efficiently and thus successfully manage the transition towards a knowledge-based economy and an information society (Table 1).

2 The Linked Data Paradigm

We briefly introduce the basic principles of Linked Data (cf. Sect. 2 from [4]). The term Linked Data refers to a set of best practices for publishing and interlinking structured data on the Web. These best practices were introduced by

Tim Berners-Lee in his Web architecture note Linked Data[2] and have become known as the Linked Data principles. These principles are:

- 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.

The basic idea of Linked Data is to apply the general architecture of the World Wide Web [6] to the task of sharing structured data on global scale. The Document Web is built on the idea of setting hyperlinks between Web documents that may reside on different Web servers. It is built on a small set of simple standards: Uniform Resource Identifiers (URIs) and their extension Internationalized Resource Identifiers (IRIs) as globally unique identification mechanism [2], the Hypertext Transfer Protocol (HTTP) as universal access mechanism [3], and the Hypertext Markup Language (HTML) as a widely used content format [5]. Linked Data builds directly on Web architecture and applies this architecture to the task of sharing data on global scale.

2.1 Resource Identification with IRIs

To publish data on the Web, the data items in a domain of interest must first be identified. These are the things whose properties and relationships will be described in the data, and may include Web documents as well as real-world entities and abstract concepts. As Linked Data builds directly on Web architecture, the Web architecture term *resource* is used to refer to these *things of interest*, which are in turn identified by HTTP URIs. Linked Data uses only HTTP URIs, avoiding other URI schemes such as URNs [8] and DOIs[3]. The structure of HTTP URIs looks as follows:

```
[scheme:][//authority][path][?query][#fragment]
```

A URI for identifying Shakespeare's 'Othello', for example, could look as follows:

```
http://de.wikipedia.org/wiki/Othello#id
```

HTTP URIs provide a simple way to create globally unique names in a decentralized fashion, as every owner of a domain name or delegate of the domain name owner may create new URI references. They serve not just as a name but also as a means of accessing information describing the identified entity.

[2] http://www.w3.org/DesignIssues/LinkedData.html
[3] http://www.doi.org/hb.html

2.2 De-referencability

Any HTTP URI should be de-referencable, meaning that HTTP clients can look up the URI using the HTTP protocol and retrieve a description of the resource that is identified by the URI. This applies to URIs that are used to identify classic HTML documents, as well as URIs that are used in the Linked Data context to identify real-world objects and abstract concepts. Descriptions of resources are embodied in the form of Web documents. Descriptions that are intended to be read by humans are often represented as HTML. Descriptions that are intended for consumption by machines are represented as RDF data. Where URIs identify real-world objects, it is essential to not confuse the objects themselves with the Web documents that describe them. It is therefore common practice to use different URIs to identify the real-world object and the document that describes it, in order to be unambiguous. This practice allows separate statements to be made about an object and about a document that describes that object. For example, the creation year of a painting may be rather different to the creation year of an article about this painting. Being able to distinguish the two through use of different URIs is critical to the consistency of the Web of Data.

There are two different strategies to make URIs that identify real-world objects de-referencable [10]. In the *303 URI strategy*, instead of sending the object itself over the network, the server responds to the client with the HTTP response code `303 See Other` and the URI of a Web document which describes the real-world object (*303 redirect*). In a second step, the client de-references this new URI and retrieves a Web document describing the real-world object. The *hash URI strategy* builds on the characteristic that URIs may contain a special part that is separated from the base part of the URI by a hash symbol (#), called the fragment identifier. When a client wants to retrieve a hash URI the HTTP protocol requires the fragment part to be stripped off before requesting the URI from the server. This means a URI that includes a hash cannot be retrieved directly, and therefore does not necessarily identify a Web document. This enables such URIs to be used to identify real-world objects and abstract concepts, without creating ambiguity [10].

Both approaches have their advantages and disadvantages [10]: Hash URIs have the advantage of reducing the number of necessary HTTP round-trips, which in turn reduces access latency. The downside of the hash URI approach is that the descriptions of all resources that share the same non-fragment URI part are always returned to the client together, irrespective of whether the client is interested in only one URI or all. If these descriptions consist of a large number of triples, the hash URI approach can lead to large amounts of data being unnecessarily transmitted to the client. 303 URIs, on the other hand, are very flexible because the redirection target can be configured separately for each resource. There could be one describing document for each resource, or one large document for all of them, or any combination in between. It is also possible to change the policy later on.

2.3 RDF Data Model

The RDF data model [7] represents information as sets of statements, which can be visualized as node-and-arc-labeled directed graphs. The data model is designed for the integrated representation of information that originates from multiple sources, is heterogeneously structured, and is represented using different schemata. RDF can be viewed as a *lingua franca*, capable of moderating between other data models that are used on the Web.

In RDF, information is represented in statements, called RDF triples. The three parts of each triple are called its subject, predicate, and object. A triple mimics the basic structure of a simple sentence, such as for example:

```
Burkhard Jung   is the mayor of   Leipzig
  (subject)       (predicate)     (object)
```

The following is the formal definition of RDF triples as it can be found in the W3C RDF standard [7].

Definition 1 (RDF Triple). Assume there are pairwise disjoint infinite sets I, B, and L representing IRIs, blank nodes, and RDF literals, respectively. A triple $(v_1, v_2, v_3) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an RDF *triple*. In this tuple, v_1 is the *subject*, v_2 the *predicate* and v_3 the *object*. We call $T = I \cup B \cup L$ the set of RDF terms.

The main idea is to use IRIs as identifiers for entities in the subject, predicate and object positions in a triple. Data values can be represented in the object position as literals. Furthermore, the RDF data model also allows in subject and object positions the use of identifiers for unnamed entities (called blank nodes), which are not globally unique and can thus only be referenced locally. However, the use of blank nodes is discouraged in the Linked Data context. Our example fact sentence about Leipzig's mayor would now look as follows:

```
<http://leipzig.de/id>
              <http://example.org/p/hasMayor>
                                          <http://Burkhard-Jung.de/id> .
     (subject)          (predicate)                (object)
```

This example shows that IRIs used within a triple can originate from different namespaces thus effectively facilitating the mixing and mashing of different RDF vocabularies and entities from different Linked Data knowledge bases. A triple having identifiers from different knowledge bases at subject and object position can be also viewed as an typed link between the entities identified by subject and object. The predicate then identifies the type of link. If we combine different triples we obtain an RDF graph.

Definition 2 (RDF Graph). A finite set of RDF triples is called RDF graph. The RDF graph itself represents an resource, which is located at a certain location on the Web and thus has an associated IRI, the graph IRI.

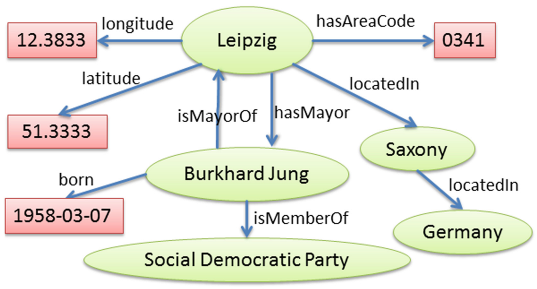


Fig. 3. Example RDF graph describing the city of Leipzig and its mayor.

An example of an RDF graph is depicted in Fig. 3. Each unique subject or object contained in the graph is visualized as a node (i.e. oval for resources and rectangle for literals). Predicates are visualized as labeled arcs connecting the respective nodes. There are a number of synonyms being used for RDF graphs, all meaning essentially the same but stressing different aspects of an RDF graph, such as *RDF document* (file perspective), *knowledge base* (collection of facts), *vocabulary* (shared terminology), *ontology* (shared logical conceptualization).

2.4 RDF Serializations

The initial official W3C RDF standard [7] comprised a serialization of the RDF data model in XML called *RDF/XML*. Its rationale was to integrate RDF with the existing XML standard, so it could be used smoothly in conjunction with the existing XML technology landscape. However, RDF/XML turned out to be difficult to understand for the majority of potential users because it requires to be familiar with two data models (i.e., the tree-oriented XML data model as well as the statement oriented RDF datamodel) and interactions between them, since RDF statements are represented in XML. As a consequence, with *N-Triples*, *Turtle* and *N3* a family of alternative text-based RDF serializations was developed, whose members have the same origin, but balance differently between readability for humans and machines. Later in 2009, *RDFa* (RDF Annotations, [1]) was standardized by the W3C in order to simplify the integration of HTML and RDF and to allow the joint representation of structured and unstructured content within a single source HTML document. Another RDF serialization, which is particularly beneficial in the context of JavaScript web applications and mashups is the serialization of RDF in JSON. Figure 4 presents an example serialized in the most popular serializations.

3 Integrating Heterogeneous Tools into the LOD2 Stack

The LOD2 Stack serves two main purposes. Firstly, the aim is to ease the distribution and installation of tools and software components that support the Linked

Data publication cycle. As a distribution platform, we have chosen the well established Debian packaging format. The second aim is to smoothen the information flow between the different components to enhance the end-user experience by a more harmonized look-and-feel.

3.1 Deployment Management Leveraging Debian Packaging

In the *Debian package management system* [9], software is distributed in architecture-specific binary packages and architecture-independent source code packages. A Debian software package comprises two types of content: **(1)** control information (incl. metadata) of that package, and **(2)** the software itself.

The control information of a Debian package will be indexed and merged together with all other control information from other packages available for the system. This information consists of descriptions and attributes for:

(a) The software itself (e.g. licenses, repository links, name, tagline, . . .),
(b) Its relation to other packages (dependencies and recommendations),
(c) The authors of the software (name, email, home pages), and
(d) The deployment process (where to install, pre and post install instructions).

The most important part of this control information is its relations to other software. This allows the deployment of a complete stack of software with one action. The following dependency relations are commonly used in the control information:

Depends: This declares an absolute dependency. A package will not be configured unless all of the packages listed in its Depends field have been correctly configured. The Depends field should be used if the depended-on package is required for the depending package to provide a significant amount of functionality. The Depends field should also be used if the install instructions require the package to be present in order to run.

Recommends: This declares a strong, but not absolute, dependency. The Recommends field should list packages that would be found together with this one in all but unusual installations.

Suggests: This is used to declare that one package may be more useful with one or more others. Using this field tells the packaging system and the user that the listed packages are related to this one and can perhaps enhance its usefulness, but that installing this one without them is perfectly reasonable.

Enhances: This field is similar to Suggests but works in the opposite direction. It is used to declare that a package can enhance the functionality of another package.

Conflicts: When one binary package declares a conflict with another using a Conflicts field, dpkg will refuse to allow them to be installed on the system at the same time. If one package is to be installed, the other must be removed first.

N-Triples

```
<http://dbpedia.org/resource/Leipzig> <http://dbpedia.org/property/hasMayor>
      <http://dbpedia.org/resource/Burkhard_Jung> .
<http://dbpedia.org/resource/Leipzig> <http://www.w3.org/2000/01/rdf-schema#label>
      "Leipzig"@de .
<http://dbpedia.org/resource/Leipzig> <http://www.w3.org/2003/01/geo/wgs84_pos#lat>
      "51.333332"^^<http://www.w3.org/2001/XMLSchema#float> .
```

Turtle

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs="http://www.w3.org/2000/01/rdf-schema#> .
@prefix dbp="http://dbpedia.org/resource/> .
@prefix dbpp="http://dbpedia.org/property/> .
@prefix geo="http://www.w3.org/2003/01/geo/wgs84_pos#> .

dbp:Leipzig  dbpp:hasMayor  dbp:Burkhard_Jung ;
             rdfs:label     "Leipzig"@de ;
             geo:lat        "51.333332"^^xsd:float .
```

RDF/XML

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                                 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:dbpp="http://dbpedia.org/property/"
                                 xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#">
 <rdf:Description rdf:about="http://dbpedia.org/resource/Leipzig">
  <property:hasMayor rdf:resource="http://dbpedia.org/resource/Burkhard_Jung" />
  <rdfs:label xml:lang="de">Leipzig</rdfs:label>
  <geo:lat rdf:datatype="http://www.w3.org/2001/XMLSchema#float">51.3333</geo:lat>
 </rdf:Description>
</rdf:RDF>
```

RDFa

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+RDFa 1.0//EN"
    "http://www.w3.org/MarkUp/DTD/xhtml-rdfa-1.dtd">
<html version="XHTML+RDFa 1.0" xml:lang="en" xmlns="http://www.w3.org/1999/xhtml"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:dbpp="http://dbpedia.org/property/"
                        xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#">
  <head><title>Leipzig</title></head>
  <body about="http://dbpedia.org/resource/Leipzig">
    <h1 property="rdfs:label" xml:lang="de">Leipzig</h1>
    <p>Leipzig is a city in Germany. Leipzig's mayor is
           <a href="Burkhard_Jung" rel="dbpp:hasMayor">Burkhard Jung</a>. It is located
                 at latitude <span property="geo:lat" datatype="xsd:float">51.3333</span>.</p>
  </body>
</html>
```

JSON-LD

```
{
 "@context": {
   "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
   "hasMayor": { "@id": "http://dbpedia.org/property/hasMayor", "@type": "@id" },
   "Person": "http://xmlns.com/foaf/0.1/Person",
         "lat": "http://www.w3.org/2003/01/geo/wgs84_pos#lat"
 },
 "@id": "http://dbpedia.org/resource/Leipzig",
 "rdfs:label": "Leipzig",
 "hasMayor": "http://dbpedia.org/resource/Burkhard_Jung",
 "lat": { "@value": "51.3333", "@type": "http://www.w3.org/2001/XMLSchema#float"
}
```

Fig. 4. Different RDF serializations of three triples from Fig. 3

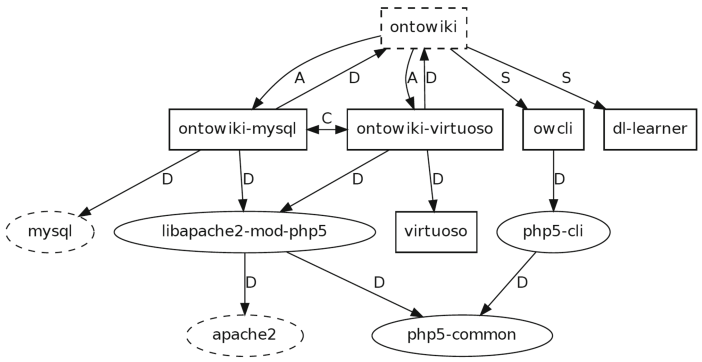


Fig. 5. Example DEB-package dependency tree (OntoWiki). Some explanation: Boxes are part of the LOD2 Stack, Ellipses are part of the Debian/Ubuntu base system, Dashed forms are meta-packages, Relations: Depends (D), Depends alternative list (A), Conflicts (C) and Suggests (S).

All of these relations may restrict their applicability to particular versions of each named package (the relations allowed are <<, <=, =, >= and >>). This is useful in forcing the upgrade of a complete software stack. In addition to this, dependency relations can be set to a list of alternative packages. In such a case, if any one of the alternative packages is installed, that part of the dependency is considered to be satisfied. This is useful if the software depends on a specific functionality on the system instead of a concrete package (e.g. a mail server or a web server). Another use case of alternative lists are meta-packages. A meta-package is a package which does not contain any files or data to be installed. Instead, it has dependencies on other (lists of) packages.

Example of meta-packaging: OntoWiki.
To build an appropriate package structure, the first step is to inspect the manual deployment of the software, its variants and the dependencies of these variants. *OntoWiki* is a browser-based collaboration and exploration tool as well as an application for linked data publication. There are two clusters of dependencies: the runtime environment and the backend. Since OntoWiki is developed in the scripting language *PHP*, it's architecture-independent but needs a web server running PHP. More specifically, OntoWiki needs PHP5 running as an Apache 2 module. OntoWiki currently supports two different back-ends which can be used to store and query RDF data: *Virtuoso* and MySQL. Virtuoso is also part of the LOD2 Stack while *MySQL* is a standard package in all Debian-based systems. In addition to OntoWiki, the user can use the OntoWiki command line client *owcli* and the *DL-Learner* from the LOD2 Stack to enhance its functionality.

The dependency tree (depicted in Fig. 5) is far from being complete, since every component also depends on libraries and additional software which is omitted here. Given this background information, we can start to plan the packaging. We assume that users either use MySQL or Virtuoso as a backend on

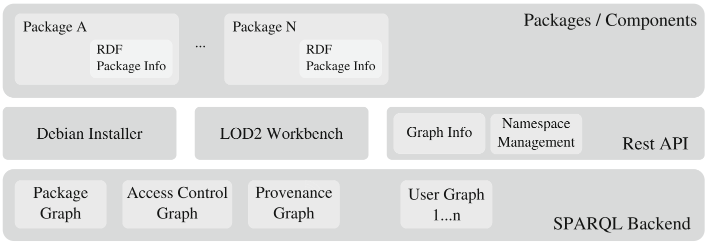


Fig. 6. Basic architecture of a local LOD2 Stack.

a server, so the first decision is to split this functionality into two packages: `ontowiki-mysql` and `ontowiki-virtuoso`. These two packages are abstracted by the meta-package `ontowiki`, which requires either `ontowiki-mysql` or `ontowiki-virtuoso`, and which can be used by other LOD2 Stack packages to require OntoWiki. Since both the MySQL backend and the Virtuoso backend version use the same system resources, we need to declare them as conflicting packages.

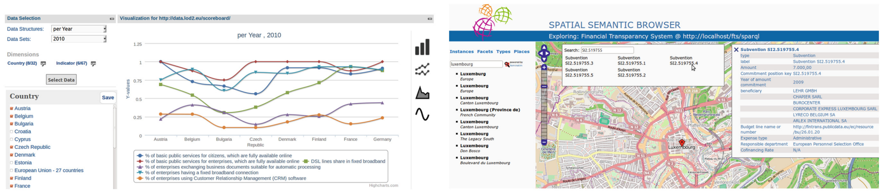
3.2 Data Integration Based on SPARQL, WebID and Vocabularies

The basic architecture of a local LOD2 Stack installation is depicted in Fig. 6. All components in the LOD2 Stack act upon RDF data and are able to communicate via SPARQL with the central system-wide RDF quad store (i.e. SPARQL backend). This quad store (Openlink Virtuoso) manages user graphs (knowledge bases) as well as a set of specific system graphs where the behaviour and status of the overall system is described. The following system graphs are currently used:

Package Graph:
In addition to the standard Debian package content, each LOD2 Stack package consists of a RDF package info which contains:

- The basic package description, e.g. labels, dates, maintainer info (this is basically DOAP data and redundant to the classic Debian control file)
- Pointers to the place where the application is available (e.g. the menu entry in the LOD2 Stack workbench)
- A list of capabilities of the packed software (e.g. resource linking, RDB extraction). These capabilities are part of a controlled vocabulary. The terms are used as pointers for provenance logging, access control definition and a future capability browser of the LOD2 workbench.

Upon installation, the package info is automatically added to the package graph to allow the workbench / demonstrator to query which applications are available and what is the user able to do with them.



Data Structures: per Year
Data Sets: 2010

Country: Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, European Union - 27 countries, Finland, France

Austria, Belgium, Bulgaria, Czech Republic, Denmark, Finland, France, Germany

- % of basic public services for citizens, which are fully available online
- % of basic public services for enterprises, which are fully available online
- DSL lines share in fixed broadband
- % of enterprises exchanging business documents suitable for automatic processing
- % of enterprises having a fixed broadband connection
- % of enterprises using Customer Relationship Management (CRM) software

Exploring: Financial Transparancy System @ http://localhost/fts/sparql

luxembourg

- Luxemburg *Europe*
- Luxembourg *Europe*
- Luxembourg *Canton Luxembourg*
- Luxembourg (Province de) *French Community*
- Luxembourg *Canton Luxembourg*
- Luxembourg *The Legacy South*
- Luxembourg *Don Bosco*
- Luxembourg *Boulevard du Luxembourg*

Search: SI2.519755

Subvention SI2.519755.3, Subvention SI2.519755.1, Subvention SI2.519755.4, Subvention SI2.519755.5, Subvention SI2.519755.2

Subvention SI2.519755.4

type	Subvention
label	Subvention SI2.519755.4
Amount	7.000,00
Commitment position key	SI2.519755.4
Year of amount commitment	2009
beneficiary	LEHR GMBH
	CHAPIER SARL
	BUROCENTER
	CORPORATE EXPRESS LUXEMBOURG SARL
	LYRECO BELGIUM SA
	ARLEX INTERNATIONAL SA
Budget line name or number	http://fintrans.publicdata.eu/ec/resource/bu/26.01.20
Expense type	Administrative
Responsible department	European Personnel Selection Office
Cofinancing Rate	N/A

Fig. 7. The visualization widgets CubeViz (statistic) and SemMap (spatial data).

Access Control Graph:
This system graph is related to WebID[4] authentication and describes which users are able to use which capabilities and have access to which graphs. The default state of this graph contains no restrictions, but could be used to restrict certain WebIDs to specific capabilities. Currently, only OntoWiki takes this graph into account and the access control definition is based on the WebAccessControl schema[5].

Provenance Graph:
Each software package is able to log system wide provenance information to reflect the evolution of a certain knowledge base. Different ontologies are developed for that use-case. To keep the context of the LOD2 Stack, we use the controlled capability vocabulary as reference points.

In addition to the SPARQL protocol endpoint, application packages can use a set of APIs which allow queries and manipulation currently not available with SPARQL alone (e.g. fetching graph information and manipulating namespaces). Two authorized administration tools are allowed to manipulate the package and access control graphs:

- The Debian system installer application automatically adds and removes package descriptions during install / upgrade and remove operations.
- The LOD2 Workbench (Demonstrator) is able to manipulate the access control graph.

All other packages are able to use the APIs as well as to create, update and delete knowledge bases. Chapter 5 gives an comprehensive overview on the LOD2 Stack components.

3.3 REST Integration of User Interfaces

Many of the components come with their own user interface. For example, the Silk Workbench is a user interface for the Silk linking engine. This workbench supports the creation of linking specifications, executing them and improving them using the feedback from the user on the created links. With the OntoWiki

[4] http://www.w3.org/wiki/WebID
[5] http://www.w3.org/wiki/WebAccessControl

linked data browsing and authoring tool, a user can browse and update information in a knowledge base. By using both tools together, the user gains the ability to study the input sources' content structure and to create links between them.

Many stack components request similar information from the user. For example, selecting the graph of interest. To provide the end-user the feeling of a harmonized single application, we develop supportive REST-based WebAPIs. These APIs offer a common application view of the LOD2 Stack. The more tools support this API, the more harmonized and integrated the end-user experience gets. Currently, the LOD2 Stack WebAPI consists of:

- *Graph management*: The set of graphs is not easy to maintain. SPARQL does not support retrieval of all graphs. The only possible query which selects all graphs that have at least one triple is performance wise quite costly: `SELECT DISTINCT ?g WHERE GRAPH ?g ?s ?p ?o` The WebAPI also standardizes some meta information like *being a system graph*. When LOD2 Stack components use this common graph management WebAPI, the end-user obtains a uniform look-and-feel with respect to graph management.
- *Prefix management*: To make RDF resources more readable, prefixes are used to abbreviate URI namespaces. Typically, each application manages its own namespace mapping. Using this REST API, a central namespace mapping is maintained, thus producing consistency among stack components. The end-user is freed from updating the individual component mappings. Moreover, an update in one component is immediately available to another.

In addition to creating supportive REST-based APIs, the LOD2 Stack encourages component owners to open up their components using REST based WebAPIs. For example, the semantic-spatial browser, a UI tool that visualizes RDF data containing geospatial information on a map, is entirely configurable by parameters encoded within its invocation URL. Similarly other visualization and exploration widgets (such as the CubeViz statistical data visualization) can directly interact with the SPARQL endpoint (cf. Fig. 7). This makes it easy to integrate into (third party) applications into the stack.

4 Conclusion and Outlook

In this chapter we gave a brief introduction to Linked Data its management life-cycle on the Web and the LOD2 Stack, the result of a large-scale effort to provide technological support for the life-cycle of Linked Data. We deem this a first step in a larger research and development agenda, where derivatives of the LOD2 Stack are employed to create corporate enterprise knowledge hubs withing the Intranets of large companies. The overall stack architecture and guidelines can also serve as a blue-print for similar software stacks in other areas.

References

1. Adida, B., Birbeck, M., McCarron, S., Pemberton, S.: RDFa in XHTML: Syntax and processing - a collection of attributes and processing rules for extending XHTML to support RDF. W3C Recommendation, October 2008. http://www.w3.org/TR/rdfa-syntax/
2. Berners-Lee, T., Fielding, R.T., Masinter, L.: Uniform resource identifiers (URI): Generic syntax. Internet RFC 2396, August 1998
3. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext transfer protocol - http/1.1 (rfc 2616). Request for Comments. http://www.ietf.org/rfc/rfc2616.txt (1999). Accessed 7 July 2006
4. Heath, T., Bizer, C.: Linked data - evolving the web into a global data space. In: Hendler, J., van Harmelen, F. (eds.) Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool, San Rafael (2011)
5. HTML 5: A vocabulary and associated APIs for HTML and XHTML. W3C Working Draft, August 2009. http://www.w3.org/TR/2009/WD-html5-20090825/
6. Jacobs, I., Walsh, N.: Architecture of the world wide web, volume one. World Wide Web Consortium, Recommendation REC-webarch-20041215, December 2004
7. Klyne, G., Carroll, J.J.: Resource description framework (RDF): concepts and abstract syntax. Technical report W3C, 2 (2004)
8. Moats, R.: Urn syntax. Internet RFC 2141, May 1997
9. Murdock, I.: The Debian Manifesto (1994). http://www.debian.org/doc/manuals/project-history/ap-manifesto.en.html
10. Sauermann, L., Cyganiak, R.: Cool URIs for the semantic web. W3C Interest Group Note, December 2008
11. Tramp, S., Frischmuth, P., Ermilov, T., Auer, S.: Weaving a social data web with semantic pingback. In: Cimiano, P., Pinto, H.S. (eds.) EKAW 2010. LNCS (LNAI), vol. 6317, pp. 135–149. Springer, Heidelberg (2010)