FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Comparison of LOD Frameworks

Finding evaluation of existing frameworks

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Software und Information Engineering

eingereicht von

Lukas Baronyai
Matrikelnummer 1326526

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Pretitle Forename Surname, Posttitle
Mitwirkung: Pretitle Forename Surname, Posttitle
Pretitle Forename Surname, Posttitle
Pretitle Forename Surname, Posttitle

Wien, 10. Juli 2017

Lukas Baronyai	Forename Surname

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Comparison of LOD Frameworks

Finding evaluation of existing frameworks

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software and Information Engineering

by

Lukas Baronyai
Registration Number 1326526

to the Faculty of Informatics

at the TU Wien

Advisor: Pretitle Forename Surname, Posttitle
Assistance: Pretitle Forename Surname, Posttitle
Pretitle Forename Surname, Posttitle
Pretitle Forename Surname, Posttitle

Vienna, 10th July, 2017

	Lukas Baronyai	Forename Surname

Erklärung zur Verfassung der Arbeit

Lukas Baronyai
Längenfeldgasse 28/8/5, 1120 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 10. Juli 2017

Lukas Baronyai

Acknowledgements

Enter your text here.

Abstract

Enter your text here.

Contents

CHAPTER 1

Introduction

1.1 Research Question

RQ: *How do common LOD-frameworks compare against each?*

1. **RQ1:** What are existing frameworks?
2. **RQ2:** What are criteria to compare frameworks?
3. **RQ3:** How do they compare against each other?
4. **RQ4:** What can be a solution for TU Wien?

1.2 Objective

1.3 Methodology

1.4 Structure of this Paper

CHAPTER 2

State Of The Art

CHAPTER 3

Methodology (RQ2)

3.1 About the difficulty of comparing frameworks

3.1.1 The term "framework"

In order to comparing frameworks of the field of Linked Data, a first step must be to define *what* a framework actually is, since it is a very generic term. One way to define it could be the definition by Roberts and Johnson, [1]:

> Frameworks are reusable designs of all part of a software described by a set of abstract classes and the way instances of those collaborate

Another way could be the explanation by Riehle in his PhD thesis, [7]:

> Frameworks model a specific domain or an important aspect thereof. They represent the domain as an abstract design, consisting of abstract classes (or interfaces). The abstract design is more than a set of classes, because it defines how instances of the classes are allowed to collaborate with each other at runtime. Effectively, it acts as a skeleton, or a scaffolding, that determines how framework objects relate to each other.
>
> A framework comes with reusable implementations in the form of abstract and concrete class implementations. Abstract implementations are abstract classes that implement parts of a framework abstraction (as expressed by an abstract class or interface), but leave crucial implementation decisions to subclasses. [...]

Both of them refer frameworks as tools for coding, used when writing own applications. One of the most classical examples might be the Spring Framework in the Java world. In the mentioned project, Apache Jena and RDF4J mostly apply on this definition.

But the problem is here, that the term is not always used and understand in this way, LDIF and the Silk frameworks define themselves as such, but providing in facts a set of tools without necessarily needing coding to work with them (except configuration files). Others may see tools like the Information Workbench or D2RQ as a framework for publishing.

On a higher level, the architectures proposed in section 4.1 might be seen as a high-level or meta-framework. And since the proposed tools in the other sections (partially) are using these architectures, one could argue, that they therefore are also frameworks.

3.1.2 Defining the limits

Next to the general problem about the term "framework", another problem is to set the borders of the examine topic. Since the paper aims to compare "Linked Data frameworks", the goal is to cover the whole process of publishing Linked Data, from the bottom persistence layer of accessing existing data, transforming data formats (e.g. relational to RDF), over cleaning and interlinking the data, over storing them in a triple store, up to making them available over an interface like SPARQL.

But there are not many tools/frameworks covering the whole process and supporting different data formats (e.g. relational data and CSV) at the same time. There are some tools like D2RQ only focusing on specific data formats, but providing the full stack, some tools like LDIF only focusing on a specific part of the process, without e.g. providing capabilities for SPARQL endpoints.

The best way is maybe using a stack of different tools to cover the whole workflow, combining them like Silk is integrated in LDIF. Or using the generic architecture, coding an own application and using partially the proposed tools.

But covering different areas, it is difficult to actually compare them. How to compare a persistence framework with a GUI framework?

3.2 Used Methodology Literature Study

In order to answer RQ1, a literature study was conducted, but since the scope is difficult to define as described in section 3.1, it was not a pure study. As the aim of this paper is to compare *common* frameworks and *best practices*, it would be not sufficient to review every possible paper about a LOD framework or tool, therefore another approach was chosen: deriving candidates from projects. In order to do that, the following process was used:

1. Identify & find a LOD application/project, ignoring the success of it
2. Find public documentation and/or scientific work of it
3. Analyse used technology, add as candidate if appropriate and if not disadvised

4. Classify candidates (see 3.3
5. Analyse reference work for possible input for 1.)
6. Analyse reference work of tool/framework at its documentation

Using this approach led to a variety of candidates, which will be listed in section 4. The candidates from section 4.3 were mostly excluded because of step 2.), which ensured a better base for the following comparison.

3.3 Classification

Resulting from 3.1 different classifications were introduced to find classification-based criteria and to balance out the vast variation of the results. The classifications are:

Class	Detail
Architecture	A general architecture without concrete technology. A framework/tool of this class can be used in combination of any other class.
Full-Stack	A tool/framework which covers the whole stack and therefore does not need another component. An "All-In-One" Solution
Presentation layer	A tool/framework which only covers the presentation or UI layer and therefore depends on other component .Managing how LOD can be accessed from outside and how the data are exposed.
Business Layer	A tool/framework which only covers the business layer and therefore depends on other component. Managing how LOD are processed.
Data Access Layer	A tool/framework which only covers the data access layer and therefore depends on other component. Managing how LOD are stored and accessed by the application.

The Classifications are based on the idea, that a majority of applications are using in one way or another a variation or parts of the three layer architecture style, with components responsible for either UI, Business or Data Access. This does *not* necessarily mean, that they use the full concepts of this architecture or even implementing this style. It is only assumed that a component have a responsibility mappable to one of the layers. Accordingly it is assumed, that a tool/framework can be associated with one of these responsibilities.

It is arguable, if the differentiation between "Full-Stack" and labelling a framework with the three layer class is necessary. The additional "Full-Stack" class was added to emphasize the "All-In-One" approach of such a tool, meaning that all components are provided, no further components are need. This also means, that the included components of the different responsibilities are either harmonized to each other or do not differentiate between these responsibilities. On the other side labelling a tool with the three layer classes, does not implicit this and can also mean, that the support of each of this layer can be optional.

CHAPTER 4

Overview of solutions (RQ1)

In order to compare frameworks an understanding of existing frameworks is necessary. This section will look at existing frameworks, what kind of frameworks they are, which of them can be used for this paper and which must be excluded. Furthermore, this section aims to understand how frameworks look like and will examine the architecture of them.

4.1 Architectures Of Frameworks

In this subsection the paper will look into three proposed models how frameworks (and/or implemented LD-applications) should look like. There are many other existing architectures and ongoing projects exposing data as Linked (Open) Data, this paper will use the following as representation of them.

4.1.1 Euclid Project

EUCLID

EdUcational Curriculum for the usage of LInked Data

The EUCLID project [1](EdUcational Curriculum for the usage of Linked Data) was founded under the *Seventh Framework Programme of Research and Technological Development*, a funding program of the European Union/European Commission for 2007-2013 [2,3].

[1] [20]

[2] [21]

[3] EUCLID in the CORDIS database: `http://cordis.europa.eu/project/rcn/103709_en.html`

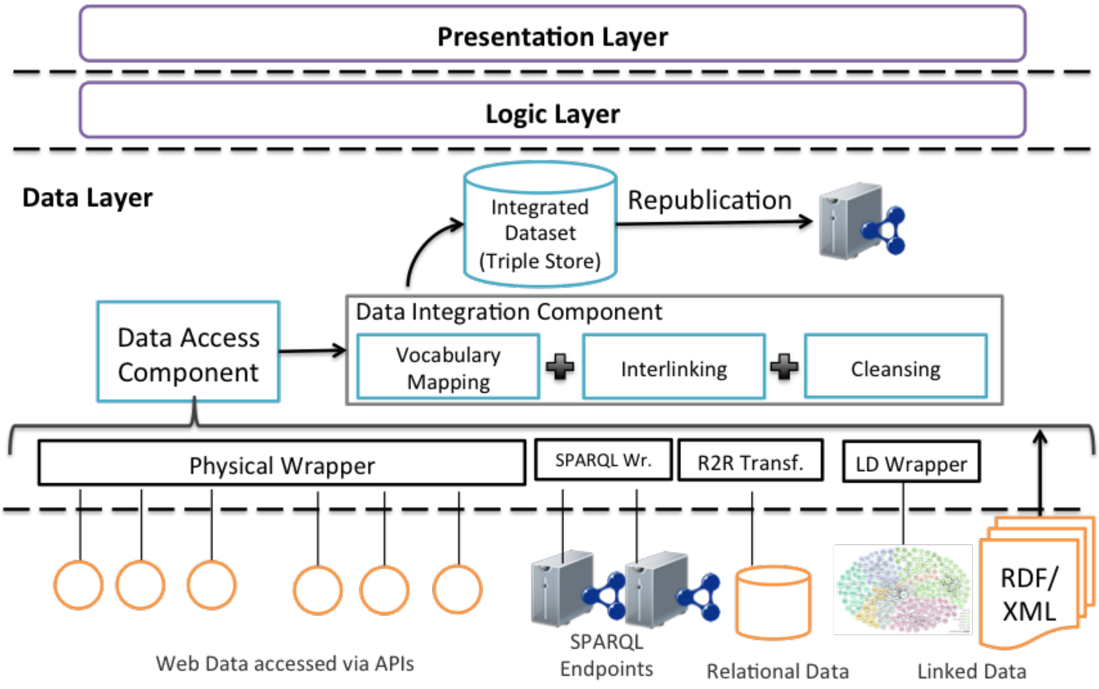

Figure 4.1: General EUCLID architecture

Aim of the project was (and still is) to gather existing knowledge and expertise of *"researchers, technology enthusiasts and early adopters in various European Member States"* and provide that accumulated as educational resources to enable the full benefit of L(O)D for European businesses. The project built upon a consortium experienced in *"over 20 LD projects with over 40 companies and public offices in more than 10 countries"* [22]

The outcome of this project is a a range of learning materials, fragmented into modules, and eLearning distribution channels. Overall there are six modules:

1. **Introduction and Application Scenarios** The introduction provides the knowledge to understand, *what* Linked Data are, the main principles, the standards and the required technologies. Further, an overview how to publish and to consume the data is given.
2. **Querying Linked Data** This chapter mainly describes SPARQL and how to use it for querying and updating.
3. **Providing Linked Data** This module deals with the production and exposure of Linked data, using the tools as R2RML (for relational databases), Open Refine (for spreadsheets), GATECloud (for natural language) and Silk (for interlinkage between datasets, see section 4.2.3 for details about this tool)

4. **Interaction with Linked Data** The projects describes in this chapter, how to explore Linked Data, using visualization tools, semantic browsers and applications, introducing search options like faceted search, concept-based search and hybrid search.

5. **Creating Linked Data Applications** This module describes how to build a Linked Data Application, which technologies to use and how to integrate common Web APIs.

6. **Scaling up** Finally this chapter examines the main issues of scalability regarding Linked Open Data and describes the relationship to Big Data.

For this paper module 3 and 5 [4] are the most interesting. Module 3 describes some useful technologies for various steps on the way of exposing L(O)D, but module 5 introduce a high level architecture and some patterns, how a L(O)D application might look like (see [23] for details). In detail, they provide a three-tier architecture (see figure 4.1 and three architecture patterns.

The architecture is very generic and consists of the classic three tiers: presentation, logic and data, each independent to the overlaying tier. Since the presentation and logic layer does not concern the actual publishing of the data, the data layer is the interesting one here. The layer consists of the *Data Access Component*, which represents the access to different data types like relational data or other Web APIs and transforms the data to RDF, the *Data Integration Component*, which does the vocabulary mapping and interlinking for the cleansing in order to e.g. identify and fix ambiguities in resource names, and finally the *Triple Store*, holding the integrated dataset for exposing it to the web.

The mentioned patterns to use for implementations are:

- **Crawling pattern** Used for loading the data in advance and storing them in a triple store, increasing the efficiency of data access. In exchange, the data might not be up to date when accessed

- **On-The-Fly Dereferencing Pattern** Meaning that the URIs are dereferenced when the application need to access the data. This pattern provides up to date data but for the cost of performance when dereference many URIs.

- **(Federated) Query Pattern** Describing the use of complex queries on a fix set of data sources, enabling to work with current data directly retrieved from the sources. The pattern offers an access up-to-date data with adequate response time in specific situations but for the cost of the complex problem to find optimal queries.

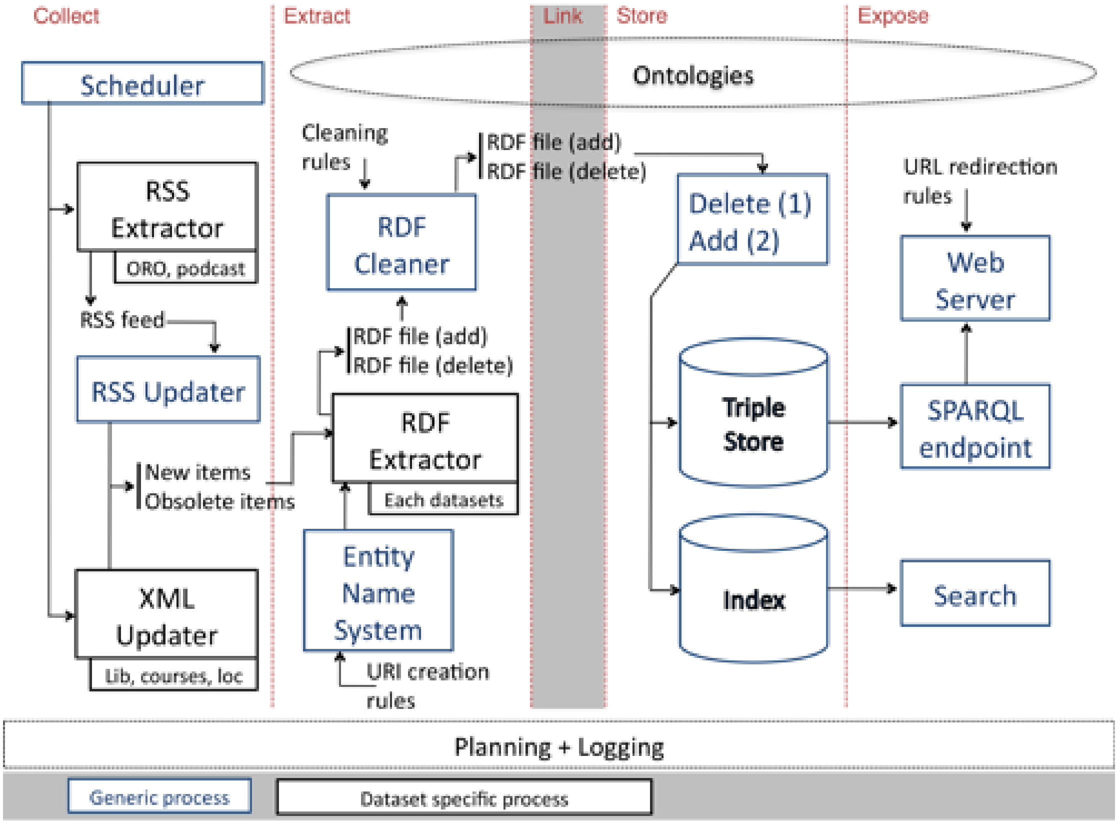

Figure 4.2: LUCERO work flow & architecture

4.1.2 LUCERO

The LUCERO project ("Linking University Content for Education and Research Online") [5] was a project at the Open University, aiming to *"scope, prototype, pilot and evaluate reusable, cost-effective solutions relying on the linked dataprinciples and technologies for exposing and connecting educational and research content"*. It was founded for one year by the JISC Information Environment 2011 Programme under the call Deposit of research outputs and Exposing digital content for education and research. [24]

The projects connected with other organizations through LinkedUniversities.org [6] to gather common issues and practices. The outcome was the first university linked data platform, `http://data.open.ac.uk/`, with a lot of impact on The Open University and the education community.

Looking at the architecture in figure 4.2 comparing to the Euclid architecture seen in the

[4] [23]

[5] The code is available in the Google Code Archive: `https://code.google.com/archive/p/luceroproject/wikis/StepByStepDocumentation.wiki`

[6] `http://linkeduniversities.org/`

previous section, there are quite a lot of similarities. Both have components for accessing different kinds of data, here called *Extractors*, for cleaning the data, here called *Cleaner*, and a Triple Store, holding the data available. The lanes "Collect", "Extract", "Link" and "Store" can be seen as the data layer from the classic three-tier architecture, the "Expose" lane as the logic and presentation layer.

Both using the crawling pattern to extract, map and store the data in a Linked Data format instead of transforming them for every request.

TABLOID

One of the outcomes next to the LOD application itself was the Tabloid ("Toolkit ABout Linked Open Institutional Data"), *"a toolkit intended to help institutions and developers to both publish and consume linked data"*. It contains work-flows, documentations, examples and tools [25] trying to address different roles such as managers, developers and users. Tabloid try to help people to understand LD, what can be done with it and give advice on a technical perspective, how to publish and consume LD, providing at the same time a detailed and generic way.

4.1.3 Linked Data book

Another big effort ampng many others of describing LD in general, how to publish and consume them and how to implement applications was done by the book "Linked Data: Evolving the Web into a Global Data Space" by Heath and Bizer [2], which received a lot of attention.

The book aims in general to give a basic understanding of LD and describing publication and consumption of LD. They providing advices and best practices, including architectures approaches, identifying the right set of URIs and vocabulary and much more. They also described an architecture, to be seen in figure 4.3

Next to patterns they also provide a general workflow for LD publishing, see figure 4.3. But comparing to the introduced architectures in the previous sections, the workflow has a different approach: instead of holding the data in a Triple Store, the workflow access and transforms the raw data on-the-fly for every request.

Next to this workflow, the book also provides various "recipes" for publishing LD and one of them is also to hold the data in a triple store as shown by Euclid and LUCERO. Furthermore the book provides a guide for the D2R-Server, which will be described in section 4.2.1.

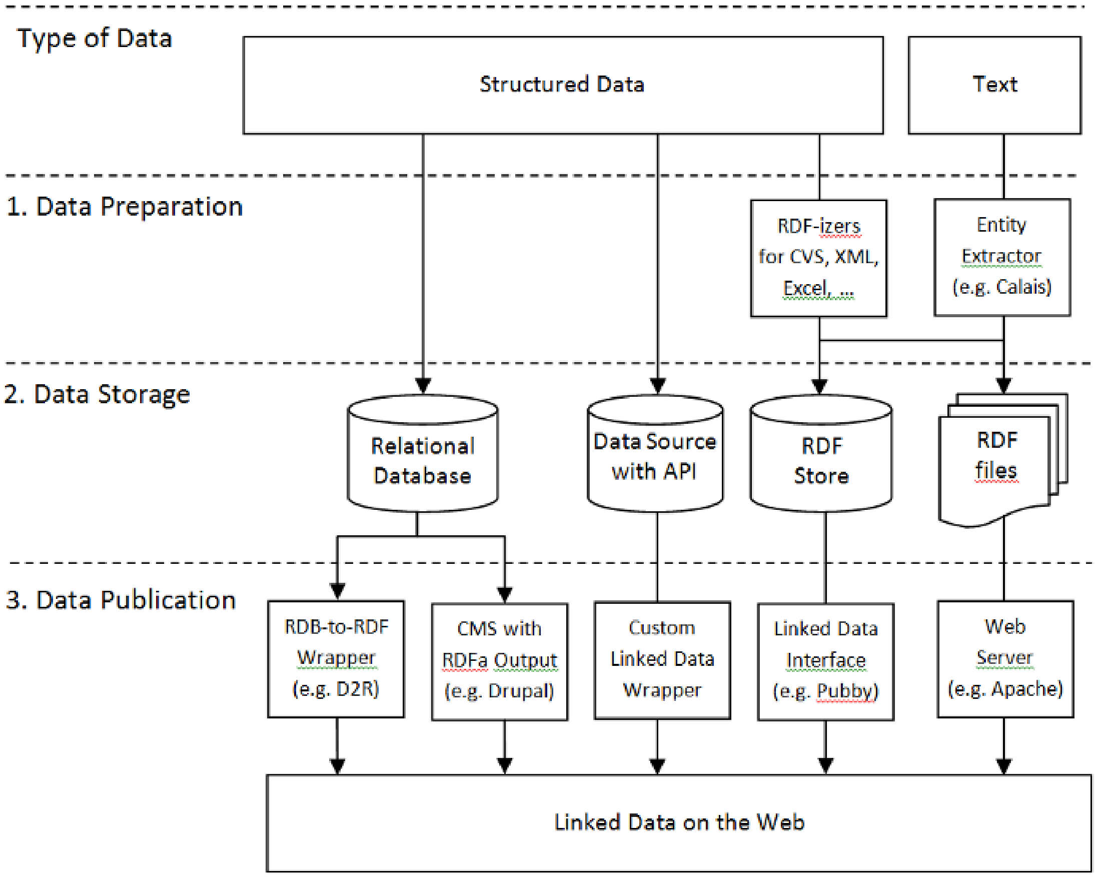
Type of Data
Structured Data
Text
1. Data Preparation
RDF-izers for CVS, XML, Excel, ...
Entity Extractor (e.g. Calais)
2. Data Storage
Relational Database
Data Source with API
RDF Store
RDF files
3. Data Publication
RDB-to-RDF Wrapper (e.g. D2R)
CMS with RDFa Output (e.g. Drupal)
Custom Linked Data Wrapper
Linked Data Interface (e.g. Pubby)
Web Server (e.g. Apache)
Linked Data on the Web

Figure 4.3: Linked Data Publishing Options and Workflows according to the LD book

4.2 Frameworks

4.2.1 D2RQ Platform

> **NOTE:** The last update on the D2RQ platform was in 2012 (version 0.8.1) and on the D2R Server in 2009 (version 0.7)

The D2RQ platform [7] was introduced by the Free University of Berlin and provides a database-to-RDF mapping. It is licensed under the terms of the GNU General Public License.

To map a relational database the platform provides a declarative mapping language, expressed in RDF, which is then be used to provide access to the database in the following, read-only, ways: [8]

[7] `http://d2rq.org`

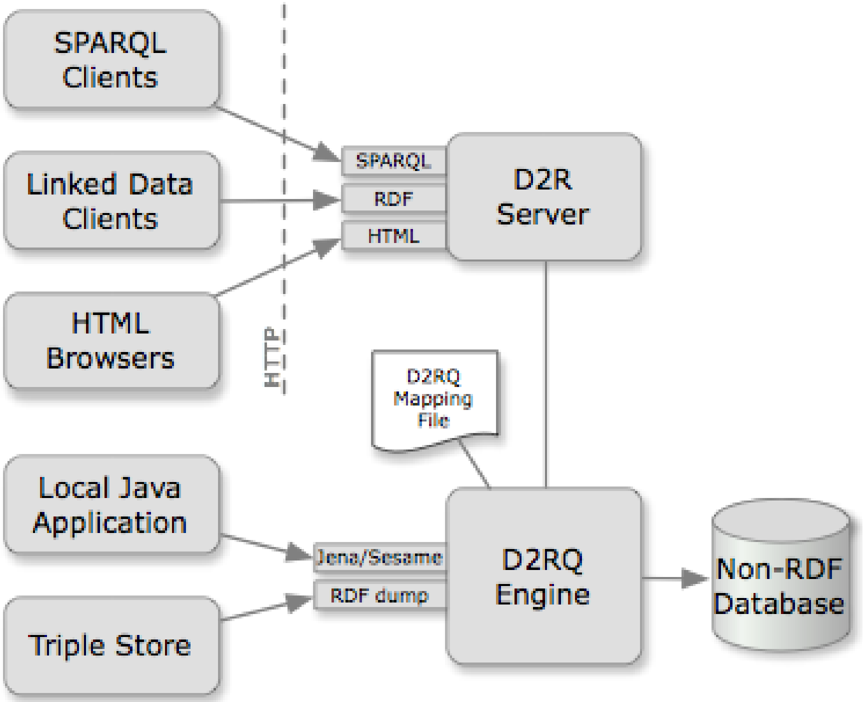

Figure 4.4: D2R Server architecture

- **RDF dumps**
- **RDF APIs**
- **SPARQL endpoint** (D2R Server)
- **Linked Data**
- **HTML view** (D2R Server)

For an overview of the framework structure see figure 4.4.

D2R Server

Part of the platform is the D2R Server [8], which provides the public access to the platform over SPARQL and HTML, publishing it to the semantic web. More concrete, the server provides a dereferencing interface, for HTTP request dereferencing, and a SPARQL interface.

[8] http://d2rq.org/d2r-server

The server uses the mentioned **On-The-Fly Dereferencing Pattern** and does not provide a triple store, therefore it may be not has as good performance than tools with a triple store, although the team made a great effort to improve it.

Part of the server is also a tool which generates automatically a corresponding mapping and RDF vocabulary for an existing table structure, using table names as class names and column names as property names. The generated mapping file can then be customised. [9]

The following applications are examples using D2R-Server:

- DBLP Bibliography (University of Hannover) [9]
- DBtune (University of London) [10]
- Database of the Nobel Prize [11]

4.2.2 Information Workbench

The Information Workbench [12] is a high customisable tool to support the building of Linked Data applications, from basic data integration up to rich UI and visualisations. The tool is developed by fluidOps and is published as Community Edition free available and under an Open Source License with a limited selection of capablities and only for non-productive use (educational use, testing, development). The enterprise edition is also available but not for free.

The workbench consists of four layers (see figure 4.5 for an overview): [10] [11]

- **Persistence** Using so-called *providers*, the layers offer capabilities to integrate and convert data from different data source and stores them in a central triple store. Alternatively it also supports virtualised integration of local and public Linked Data sources using a *federation layer*.
- **Platform** On top of the persistence layer the core Platform layer a selection of modules and functionalities covering generic needs of Linked Data applications, the most important are a *Semantic Wiki & Widget Engin*e, an *User Management & Access Control*, a *Search & Analytics Engine* and a *Workflow Engine*.
- **SDK** To support customised applications the workbench provides a SDK (Solution Development Kit) for developers to build domain specific applications, including *extensible data providers*, *data management facilities*, modified *ontologies*, *templates*, *widgets* and different APIs for extensive *system configuration*, *rules* and *workflows*.

[9] http://dblp.uni-trier.de/

[10] http://dbtune.org/

[11] http://data.nobelprize.org/

[12] https://www.fluidops.com/en/products/information_workbench/

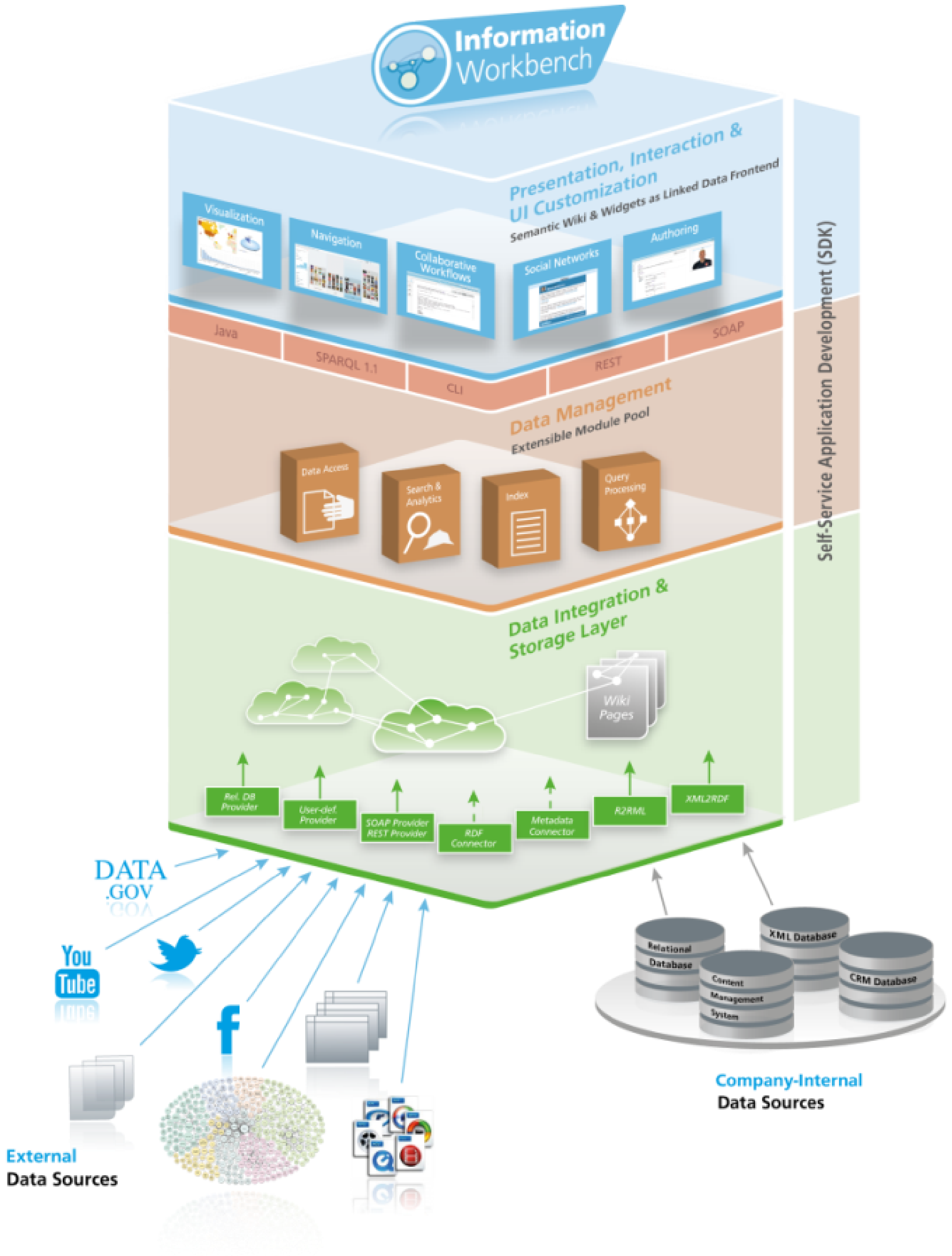

Figure 4.5: Architecture of the Information Workbench

- **Solution** On top of all layer stands the final solution, the application itself, which is either directly deployed through a RESTful API or over a zipped file for other installation approaches.

The resulting application is again customisable by widget and different views, enabling data exploration and visualisation.

4.2.3 LDIF – Linked Data Integration Framework

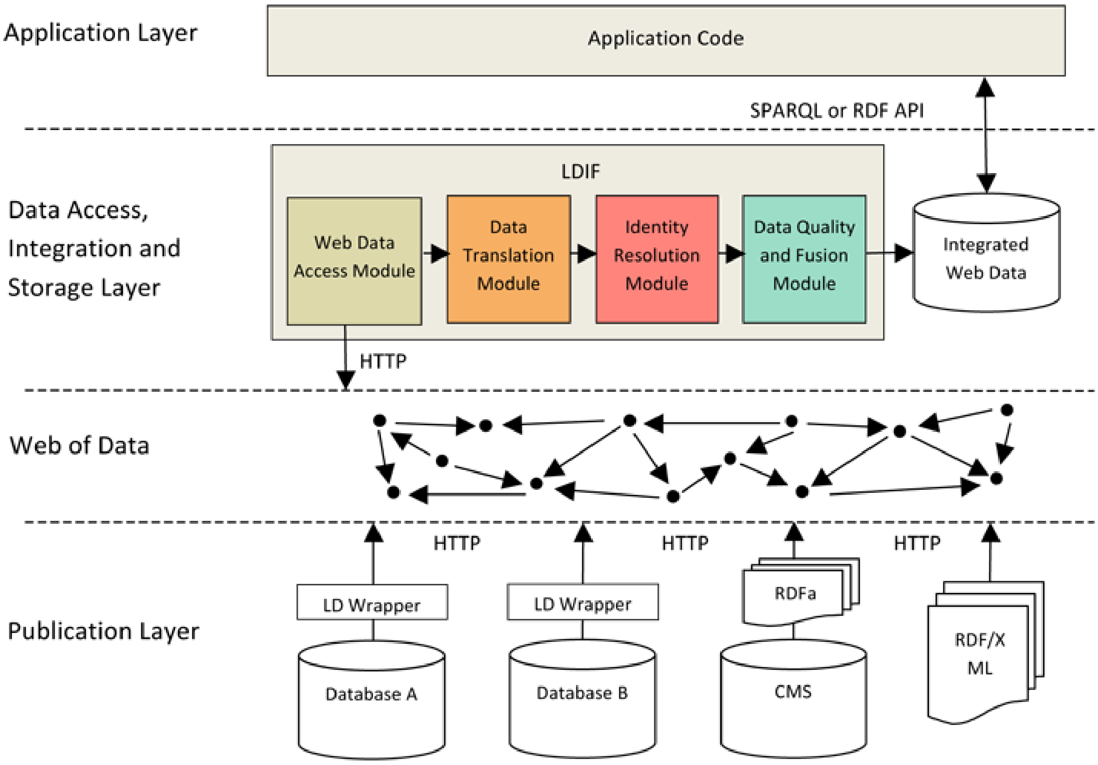

Figure 4.6: LDIF in the context of a LD application

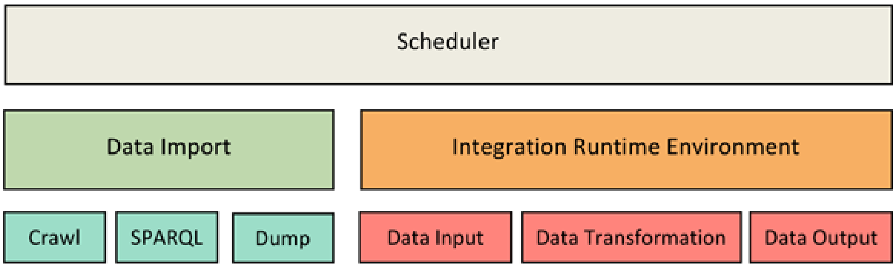

Figure 4.7: Components of LDIF

LDIF [13] was developed by the University of Mannheim and is published under the terms of the Apache Software License. It is implemented in Scala and aims to translate *"heterogeneous Linked Data from the Web into a clean, local target representation while keeping track of data provenance."*.

From a component perspective, LDIF consists of pluggable modules and a runtime environment, managing the data flows between them. The modules are: [12] [13]

- **Data Access Modules & Scheduler** For accessing the data to transform, LDIF provides several ways to import them. These import jobs are managed by a scheduler, which frequently fills a local cache. The module supports Triple/Quade Dump (for RDF/XML, N-Triples, N-Quads and Turtle formats), Crawler (using LDSpider [14]) and SPARQL imports.
- **Data Translation** For translating Web data using different vocabularies into a single target vocabulary, LDIF uses the R2R Mapping Language [15].
- **Identity Resolution** To find different URIs in different data pointing to the same entity, LDIF employs the Silk Link Discovery Framework with the Silk - Link Specification Language (Silk-LSL).
- **Data Quality Assessment and Fusion** For quality assessment, LDIF uses Sieve Data Quality Assessment and Data Fusion Framework [16].
- **Data Output** In the final step, LDIF write the cleaned data together with the provenance information in a single N-Quads file or without the meta-information in a N-Triples file.
- **Runtime Environment** As mentioned, the runtime environment manage the data flow between each module, providing an in-memory (fast, but limited scalable), a RDF store (using Apache Jena TDB and SPARQL queries, better scalable for the price of performance) and a Hadoop version.

Silk

Silk [17] is *"an open source framework for integrating heterogeneous data sources."* using the declarative Silk - Link Specification Language (Silk-LSL). It generates RDF links between data sets by custom link specifications. There are three different variations: [14]

- **Silk Single Machine** generates RDF links between two data items on a single machine.

[13] http://ldif.wbsg.de/

[14] https://github.com/ldspider/ldspider

[15] http://wifo5-03.informatik.uni-mannheim.de/bizer/r2r/spec/

[16] http://wifo5-03.informatik.uni-mannheim.de/bizer/sieve/

[17] http://silkframework.org/

- **Silk MapReduce** is for big scale datasets, using Hadoop and distributes to multiple machines.
- **Silk Server** is intended be used as an identity resolution component of as Linked Data consuming application. It provides a REST interface and runs as an HTTP server.

For details about the Link Discovery Framework see [3] and [15], for the server version consult [14].

4.2.4 Eclipse RDF4J (formerly Sesame)

Eclipse RDF4J [18] (formerly known as Sesame) is *a powerful Java framework for processing and handling RDF data. This includes creating, parsing, scalable storage, reasoning and querying with RDF and Linked Data. It offers an easy-to-use API that can be connected to all leading RDF database solutions.* It can be used as an embedded part of an application or as a stand-alone server.

Originally developed as Sesame by Aduna as part of the "On-To-Knowledge" project (1999-2002), it was official forked into RDF4J. It is licensed under a BSD-style license.

The frameworks comes with many components, like Alibaba, an API for mapping Java classes onto ontologies. The RDF database API is unlikely similar solutions, it consists of stackable interfaces for adding functionality. Next to the intern abstract storage engine (SAIL, Storage and Inference Layer), many other triplestores are supported, like Ontotext GraphDB, Mulgara, and AllegroGraph

4.2.5 Apache Jena

Apache Jena [19] is *a free and open source Java framework for building Semantic Web and Linked Data applications.* It was originally developed by HP Laboratories and now maintained by the Apache Software Foundation and is licensed under the Apache License 2.0.

The framework provides an API to extract data from and write to RDF, supporting relational databases, RDF/XML, Turtle and Notation 3. In contrast to RDF4J it also supports OWL.

More concrete, Jena can be used to manipulate RDF data, storing them in a triple store and publish it as a SPARQL access point. This HTTP interface is called *Fuseki*, which is in fact a sub-project of Jena and can be also run as stand-alone server using the Jetty web server.

[18] http://rdf4j.org/

[19] https://jena.apache.org/

4.3 Excluded Tools and Projects

4.3.1 LD-Patterns

The Linked Data Patterns book by Dodds and Davis (see [4]) tried to give an overview of existing design pattern regarding LD. But they don't give concrete architectures or architecture relating informations, so this paper will not use its content. But it is suggested, that this design pattern catalogue is used additionally when creating an application.

4.3.2 LOD2 Stack

The LOD2 stack, introduced by Auer et. al., is *is 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.* [16] For this paper the proposed stack of technology was too generic to compare it with other frameworks and the website of the project [20] was at point of writing this paper offline, therefore it was excluded of this paper.

4.3.3 LODUM

Another interesting project is the LODUM project (Linked Open Data University of Münster), the Open Data initiative of the university, hosted at the Institute for Geoinformatics' Semantic Interoperability Lab (MUSIL). The project team has co-initiated both LinkedUniversities.org and LinkedScience.org.

It was excluded for this paper because the project don't provide public documentation of their architecture or any other part of their technical details `http://lodum.de/`

4.3.4 Synth and SHDM

Synth [21] is a development environment for building SHDM [22] (Semantic Hypermedia Design Method) modelled applications, providing a set of modules, receiving SHDM generated models. Synth comes with a web browser GUI for adding and editing these models. A conceptual view of the architecture can be seen in figure 4.8, where the dashed boxes are modules and the whites boxes insides the module components. [5] The authors de Souza Bomfim and Schwabe describe in two papers, how a Linked Data application can be build with the environment: [5] and [17].

Since their description is very abstract and there are no further documentations of the tool, it was excluded for this paper.

[20] `http://stack.linkeddata.org/lod2//`

[21] `http://www.tecweb.inf.puc-rio.br/synth`

[22] `https://www.w3.org/2005/Incubator/model-based-ui/wiki/SHDM_-_Semantic_Hypermedia_Design_Method`

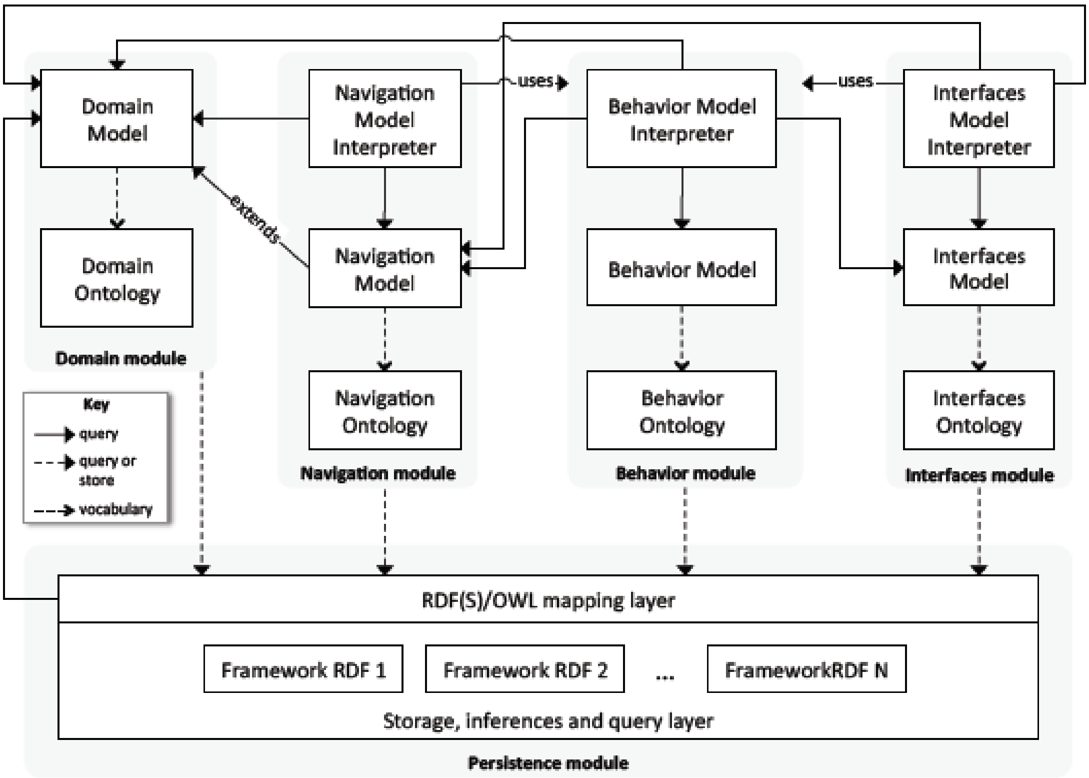

Figure 4.8: Concept of the Synth Architecture

CHAPTER 5

Criteria (RQ2)

Since there are a wide variety of solutions, it is hard to find a set of criteria which can be applied to all in the same way in order to compare them. Therefore this paper will use 4 different criteria groups to ensure higher cover of them, even in case some of the criteria might not be applicable. Otherwise a comparison will be not possible.

5.1 Criteria Group 1: Criteria from previous study

Criteria	**Scala**	**Explanation**
Maintainability	+/-	How much effort needs the maintenance?
Data Freshness	yes/ no	Can it deal with new data?
Flexibility	yes/ no	Can it deal with heterogenous and/or legacy data? Can it deal with changes in the ontology?

Table 5.1: Criteria group 1

From a previous study [18] conducted by the author, users at TU Wien are expecting and requesting from a LOD application: **clear data ownership**, management of **data freshness** and **data quality** and **maintenance**. Since data ownership is a concern of organisation and cannot be clarified by a tool, data freshness, data quality and maintainability are introduced as criteria. Since data quality is a very generic term, it will be used as criteria category. Additionally a concern of the stakeholders from the paper was how to deal with legacy data, therefore flexibility is included to Data quality.

The resulting criteria can be seen in table 5.1.

Criteria	Scala	Explanation
Effectiveness	+/-	How well do the users achieve their goals using the system? (good/bad)
Efficiency	+/-	What resources are consumed in order to achieve their goals? (less/much)
Satisfaction	+/-	How do the users feel about their use of the system? (good/bad)
Security	+/-	How well is security ensured? (good/bad)
Learnability	+/-	How much time is needed to learn the system? (less/much)

Table 5.2: Criteria Group 2: Usability

5.2 Criteria Group 2: Usability

In every software application and especially solutions designed for end-users, usability is a huge and very important point these days. There are many definitions and measurements, ISO and other proposed models, trying to classify and define usability. This paper will use an enhanced ISO model, proposed by Abran et.al. [6]. Since the goal of this paper is not a complete usability analysis of the tools/frameworks and the variety of the chosen tools is too wide, the analysis of these aspect will be more of a general type. Abran et.al. are proposing various measurements for the different categories of their model, the interested reader might use them for a detailed analysis. For this paper, these measurements will only be used as a guideline to estimate an assessment for the tools.

The model can be seen in table 5.2

5.3 Criteria Group 3: Data formats

An important aspect for the final decision for one of the tools can be the supported data format. It might be an (external) requirement or resulting from the fact of existing data. Since this highly depends on the context and the use case, this criteria will not be rated in any way, this paper will only line out the supported data formats.

5.4 Criteria Group 4: Linked Data Publishing Checklist

Since the whole paper is about Linked Data, it is important to analyse not only the tools itself but also the resulting LDs. In order to do that, the Linked Data Publishing Checklist by Heath et.al. [2] will be used. Another alternative could be the the LOD defintion itself by Tim Berners-Lee [19], but this paper will presume, that a L(O)D tool will produce valid L(O)D. As one can expect from a checklist, the rating for this criteria will be only fulfilled/not fulfilled.

Criteria
Does your data set links to other data sets?
Do you provide provenance metadata?
Do you provide licensing metadata?
Do you use terms from widely deployed vocabularies?
Are the URIs of proprietary vocabulary terms dereferenceable?
Do you map proprietary vocabulary terms to other vocabularies?
Do you provide data set-level metadata?
Do you refer to additional access methods?

Table 5.3: Criteria Group 4: Linked Data Publishing Checklist

The checklist can be seen in table 5.3

5.5 End result

Since this paper is designed to be used as a help for decisions, the aim can not be to find the "best" solution, therefore there will not be such a thing like an end result. E.g. for some situation a solution with bad usability can be more appropriate because of the supported data formats than a solution with a higher overall rating.

Criteria group	Criteria
Group 1	Maintainability
	Data Freshness
	Flexibility
Group 2: Usability	Effectiveness
	Efficiency
	Satisfaction
	Security
	Learnability
Group 3: Data formats	Data formats
Group 4: LD Publishing Checklist	LD Publishing Checklist

Table 5.4: Complete Criteria Catalogue

CHAPTER 6

Comparison (RQ3)

It this section the comparison itself will be done. In order to do that, first in section 6.1 the classification introduced in section 3.3 will be applied the found solution. Then in section 6 the criteria defined in section 5 will be applied for each of them, divided in the defined groups. The summary in section 6.6 will then give in overview of the done comparison.

An overview of the solutions to compare can be seen as recap in table 6.1, in order to simplify the following tables, each of the solutions is given an ID to refer.

ID	Framework
1	Euclid Project
2	LUCERO
3	Linked Data book
4	D2RQ Platform
5	Information Workbench
6	Linked Data Integration Framework
7	Eclipse RDF4J
8	Apache Jena

Table 6.1: Overview of the solutions

6.1 Classification

ID	Architecture	Full-Stack	Presentation layer	Business Layer	Data Access Layer	Note
1	x			x	x	
2	x			x	x	
3	x			x	x	
4			x	x	x	D2R Server includes HTML view and SPARQL endpoints
5		x				
6				x	x	
7				x	x	
8			x	x	x	Provides optionally SPARQL endpoints and stand-alone server with Jetty

Table 6.2: Classification

ID	Maintain-ability	Data Freshness	Flexibility	Note
1	+	yes*	yes*	*depending on implementation
2	?*	yes	yes*	*further investigations required
3	+	yes*	yes*	*depending on implementation
4	+	yes	yes*	*depending on configuration
5	?*	yes*	yes*	*depending on configuration
6	+	yes	yes*	
7	∼*	?*	yes*	*depending on implementation
8	∼*	?*	yes*	*depending on implementation

Table 6.3: Comparison Criteria Group 1

6.2 Criteria Group 1: Criteria from previous study

1: Euclid Project

Since the Euclid Project is very generic, the Data Freshness and Flexibility totally depends on how the concepts are implemented in an application. If the concepts are correctly implemented, Maintainability is probably well supported, since the three-layered architecture is widely recognised for it. For Data Freshness and Flexibility, the implementation needs to be done focused on it, in order to achieve it, since the project does it not explicitly.

2: LUCERO

The project supports Data Freshness by its Updaters and Schedulers, and can react to ontology changes by its Entity Name System. But it is unclear, how well it supports Maintainability, further investigations are required, which exceed the scope of this paper.

3: Linked Data book

Similar to the Euclid Project, the Data Freshness and Flexibility is possible well supported, but depends on the implementation. It is also not an explicit focus. Maintainability again is inherit of the concept.

4: D2RQ Platform

The D2RQ Mapping Language can flexibly handle heterogeneous data as well as legacy data, the quality of Data Freshness depends on the configuration but is itself inherit of the concept.

5: Information Workbench

Since the Workbench is designed to easily integrate different datasources, it can handle legacy data and ontology changes relatively good, depending on custom configurations. It might be a challenge to find the correct configurations, but the provided documentation well done.

6: LDIF

LDIF is explicit designed to handle various data sources and vocabularies, mapping them to an uniform vocabulary. The framework further provides scheduler to keep the data as fresh as needed (running hourly, daily etc.)

7: Eclipse RDF4J

All three criteria are highly depending on the implementation of the framework and therefore how maintainability, freshness and flexibility are handled. The framework itself does only interact with a repository (read/write), which can potential be filled by another application which handles freshness and flexibility. Maintainability could be a bit tricky in this situation, if the other application is out of control of the RDF4J implementation.

8: Apache Jena

Jena is very similar to RDF4J, all statements there are valid for Jena too.

6.2.1 Summary Group 1

Most of the solutions are supporting all data freshness and flexibility although most of them depending on the concrete implementation or configuration of the application. Only LDIF have a clear support of especially data freshness and flexibility. Maintainability is a critical point for most of the solutions.

6.3 Criteria Group 2: Usability

ID	Effectiveness	Efficiency	Satisfaction	Security	Learnability	Note
1	N.A.	N.A.	N.A.	-	+	Not applicable, good documentation
2	N.A.	N.A.	N.A.	-	-	No UI, no Security, bad documentation
3	N.A.	N.A.	N.A.	-	+	Not applicable, good documentation
4	+	+	+	-	+	Good documentation and UI, no security
5	+	+	+*	+	+*	* "easy to learn, hard to master"
6	N.A.	N.A.	N.A.	-	+	Not applicable, good documentation
7	N.A.	N.A.	N.A.	~	+	Not applicable, good documentation
8	N.A.	N.A.	N.A.	+	+	Not applicable, good documentation

Table 6.4: Comparison Criteria Group 2: Usability

1: Euclid Project

Since the Usability totally depends on the implemented application, no general statements can be given here. Security is not explicit. The documentation is very good.

2: LUCERO

The project does not support an explicit UI, therefore Usability is hard to evaluate. It also does not have any integrated security. The documentation is old and possible outdated, therefore learnability of the system is relatively bad.

3: Linked Data book

Again, similar to Euclid, the Usability strongly depends on the implementation and is therefore not applicable. Security is not explicit.

4: D2RQ Platform

The D2RQ server provides a basic but good readable UI. The documentation of the whole platform is very good. Security is not part of the project.

5: Information Workbench

The workbench has a very well designed UI, supporting the user to achieve their tasks relatively simple. The system is very good readable, enabling the user to explore the functionality and fulfilling their task. The according documentation is well done for "standard" users, but might be not enough for advanced development. Security is per default activated.

6: LDIF

LDIF does not have an explicit UI, therefore there can be no assessment of it done. Security is not part of the LDIF concept. The documentation is well done, but covers only simple topics, which could be problematic when running into problems while using the framework.

7: Eclipse RDF4J

The framework provides a simple UI for the RDF4J server and workbench for managing the repositories, but using RDF4 means writing Java Code. Therefore UI can not be topic of examination here. The RDF4J repositories have simple user management, but are handled per default by plain text cookies in the browser. The documentation is detailed and in good condition.

8: Apache Jena

Again, all statements from RDF4J are valid for Jena too. But in contrast to RDF4J Jena uses Apache Shiro for security. The server for Jena is called Fuseki.

6.3.1 Summary Group 2

Most of the solutions does not support an UI, except D2RQ and the information workbench. Security is only explicit mentioned in the information workbench and Apache Jena, RDF4J's default security is very basic. All solutions except LUCERO provide a good documentation.

6.4 Criteria Group 3: Data formats

1: Euclid Project

The Euclid Project does theoretically supports every kind of formats, since it requires the developer to write a consumer for each data source, additionally enhanced by a wrapper

Solution	Data formats	Note
1	Potential every possible format	Every format needs its own wrapper/consumer
2	Default RSS & XML	Additional data formats need custom extractors
3	Recipes for RDF, XML, HTML, relational databases, Wrapper	
4	Only relational databases	
5	Table-base (csv, excel, groovy, jdbc, rest, tsv, sparql etc), tree-based (xml, json, etc), RDF	Each data source needs a configured data provider which provides R2RML (tabular datasources) and XML mappings to RDF (see)
6	N-Quads dumps, RDF/XML, N-Triples, Turtle dumps, dereferenced URIs, SPARQL	Using LDSpider for URI crawl import
7	Only RDF	
8	Only RDF & OWL	

Table 6.5: Comparison Criteria Group 3: Data formats

to transform the data to RDF.

2: LUCERO

Per default, the project supports only RSS and XML, but it provides also mechanism to integrate additional data extractors to the system, to enable addition data formats.

3: Linked Data book

The book provides recipes for RDF, XML, HTML, relational databases and wrapper for existing applications and Web APIs. Additional data formats can probably integrated by adopting the recipes.

4: D2RQ Platform

The scope of the platform are only the integration of relational databases.

5: Information Workbench

The workbench does support a wide range of data format. For each data source, a data provider has to be configured. For table- and tree-based data formats are mappings available to transform the data to RDF.

6: LDIF

LDIF provides four types of import jobs: Quad (import N-Quads dumps), Triple (import RDF/XML, N-Triples or Turtle dumps), Crawl (import by dereferencing URIs as RDF data, using the LDSpider Web Crawling Framework) and SPARQL Import Job (import by querying a SPARQL endpoint)

7: Eclipse RDF4J

The framework is mainly meant for working with data from a RDF repository and not explicit for putting/creating data in it in the first place.

8: Apache Jena

Jena is also developed with the focus of accessing data rather than creating the data store, therefore only RDF and additionally OWL are supported.

6.4.1 Summary Group 3

The solutions offer a wide range of data types in total, the Euclid Project and the LD book provide recipes for various data types, the Information Workbench does this out-of-the-box. D2RQ, RDF4J and Jena are specialised solutions, focusing only on single data types.

6.5 Criteria Group 4: Linked Data Publishing Checklist

ID	1	2	3	4	5	6	7	8
Does your data set links to other data sets?	x	x	x	x	x	x	?	?
Do you provide provenance metadata?	x	?	x	x	x	x	x	x
Do you provide licensing metadata?	x	?	x	x	x	x	x	x
Do you use terms from widely deployed vocabularies?	x	x	x	x	x	x	?	?
Are the URIs of proprietary vocabulary terms dereferenceable?	x	x	x	x	x	x	?	?
Do you map proprietary vocabulary terms to other vocabularies?	x	x	x	x	x	x	-	-
Do you provide data set-level metadata?	x	?	x	x	x	x	x	x
Do you refer to additional access methods?	x	-	x	x	x	x	?	?

Table 6.6: Comparison Criteria Group 4: Linked Data Publishing Checklist

1: Euclid Project

Since of its generic nature, possible every point of the checklist can be fulfilled depending on the implementation. But the architecture does not require any of the points.

2: LUCERO

LUCERO supports interlinking of data and through its Entity Name System different vocabularies. Since its bad documentation, it is unclear, if any metadata are provided (assumably not).

3: Linked Data book

Since Heath et al. propose the checklist in this book, it is also implicit and explicit integrated into the described solutions.

4: D2RQ Platform

The D2RQ server provides comprehensive support for metadata, easy customizable by templates. The D2RQ Mapping language is very powerful in handling and mapping various kinds of vocabulary.

5: Information Workbench

The workbench does support metadata, licensing metadata can be provided by custom implementations. Interlinking is as well supported as different kinds of vocabularies.

6: LDIF

The framework implicit provides provenance next to the triple store, links to other data sets and maps proprietary vocabulary terms. The LD book explicit mentions LDIF as good example, therefore it can easily be assumed, that the checklist is fulfilled.

7: Eclipse RDF4J

Again, same as the Euclid Project, nearly every point of the checklist could be fulfilled by an implementation, especially the points about meta data. The schema/vocabulary/structure can be independent of the application and managed by another one, therefore it is out of the scope of RDF4J.

8: Apache Jena

For Jena are the same arguments than for RDF4J valid: meta data are depended on the implementation, schema/vocabulary/structure can be out of the scope.

6.5.1 Summary Group 4

The checklist is overall well supported although it, again, is depending on the implementation of some of the solutions like Euclid, RDF4J or Jena. The last two are not managing the schema/vocabulary/structure by themselve, therefore the checklist is only partially applicable.

6.6 Summary

Overall the concept of having multiple criteria worked, since some solution could not be evaluated in some categories like Usability. But in the overall view, the criteria helped to find a standardised way to describe the solutions and compare them. In the following section, the evaluation of each solution will be summarized to have a better overview.

1: Euclid Project

Since the Euclid Project does only provide a generic architecture, most of the criteria are not directly applicable and are depending on the concrete implementation. Correctly applied it does however supports directly or indirectly maintainability, data freshness, flexibility, various kinds of data formats and the complete LD checklist. The architecture has no focus, neither explicit nor implicit, on security, therefore it needs to be done additionally if required. The documentation of the architecture is outstanding and very good done. Comparing to the other solutions, Euclid provides structures how other solutions or custom implementation can interact with each other. It can also be used as blueprint when combining other solutions.

2: LUCERO

LUCERO is on one hand overall bad documented and outdated. On the other hand, it does support data freshness and flexibility per default due its mechanisms and can support various data formats by custom extractors. But due the first facts, it is not recommended any more to use LUCERO in a real application.

3: Linked Data book

The summary for the LD book architecture is similar structured to the one of the Euclid project: good documentation but overall very generic. It can support maintainability, data freshness and flexibility if well implemented. The documentation is good and does include various recipes of data formats. A possible use case for it is the same as for Euclid: as blue print for combining other solutions.

4: D2RQ Platform

The D2RQ is as mentioned a specialised tool for relational databases. According to it, it has only limited data format supports. But due its simple structure and focused usage, data freshness, maintainability and flexibility can be assured. D2RQ includes a good UI and documentation. This solution is ideal for a very specific use case, requiring only a relational database to publish as L(O)D, but solving it as an all-in-one solution without the need of further tool integration.

5: Information Workbench

The Information Workbench is an all-in-solution, rich with functions and with a wide range of supported data formats. It can be used as a full stack solution with UI, integrating different data repositories. The documentation is well done but simple written, resulting maybe into problems handling more complex problems. The Workbench is ideal for an use case involving multiple data sources. For smaller use cases, it could be an overloaded solution.

6: LDIF

7: Eclipse RDF4J

8: Apache Jena

CHAPTER 7

Usage at TU Wien (RQ4)

7.1 General

7.2 Situation 1: Specialised Single Solution

7.3 Situation 2: Function-rich Platform

7.4 Situation 2: Complete Controlled Platform

CHAPTER 8

Summary and future work

Enter your text here.

List of Figures

List of Tables

References to refereed scientific work

[1] D. Roberts and R. Johnson, “Evolving frameworks”, *Pattern languages of program design*, vol. 3, 1996.

[2] T. Heath and C. Bizer, “Linked data: Evolving the web into a global data space”, *Synthesis lectures on the semantic web: theory and technology*, vol. 1, no. 1, pp. 1–136, 2011.

[3] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov, “Silk-A Link Discovery Framework for the Web of Data.”, *LDOW*, vol. 538, 2009.

[4] L. Dodds and I. Davis, “Linked data patterns”, *Online: `http://patterns.dataincubator.org/book`*, 2011.

[5] M. H. de Souza Bomfim and D. Schwabe, “Synth–Linked Data Application Implementation Environment”,

[6] A. Abran, A. Khelifi, W. Suryn, and A. Seffah, “Usability meanings and interpretations in ISO standards”, *Software Quality Journal*, vol. 11, no. 4, pp. 325–338, 2003.

References to non-refereed work

[7] D. Riehle, “Framework design”, PhD thesis, 2000.

[8] C. Bizer and R. Cyganiak, “D2rq-lessons learned”, in *W3C Workshop on RDF Access to Relational Databases*, 2007, p. 35. [Online]. Available: `https://www.w3.org/2007/03/RdfRDB/papers/d2rq-positionpaper/`.

[9] ——, “D2r server-publishing relational databases on the semantic web”, in *Poster at the 5th international semantic web conference*, vol. 175, 2006.

[10] P. Haase, M. Schmidt, and A. Schwarte, “The information workbench as a self-service platform for linked data applications”, in *Proceedings of the Second International Conference on Consuming Linked Data-Volume 782*, CEUR-WS. org, 2011, pp. 119–124.

[11] A. Gossena, P. Haasea, C. Hüttera, M. Meiera, A. Nikolova, C. Pinkela, M. Schmidta, A. Schwartea, and J. Tramea, “The Information Workbench–A Platform for Linked Data Applications”,

[12] A. Schultz, A. Matteini, R. Isele, C. Bizer, and C. Becker, “Ldif-linked data integration framework”, in *Proceedings of the Second International Conference on Consuming Linked Data-Volume 782*, CEUR-WS. org, 2011, pp. 125–130.

[13] A. Schultz, A. Matteini, R. Isele, P. N. Mendes, C. Bizer, and C. Becker, “LDIF-a framework for large-scale Linked Data integration”, in *21st International World Wide Web Conference (WWW 2012), Developers Track, Lyon, France*, 2012.

[14] R. Isele, A. Jentzsch, and C. Bizer, “Silk server-adding missing links while consuming linked data”, in *Proceedings of the First International Conference on Consuming Linked Data-Volume 665*, CEUR-WS. org, 2010, pp. 85–96.

[15] A. Jentzsch, R. Isele, and C. Bizer, “Silk-generating rdf links while publishing or consuming linked data”, in *Proceedings of the 2010 International Conference on Posters & Demonstrations Track-Volume 658*, CEUR-WS. org, 2010, pp. 53–56.

[16] S. Auer, L. Bühmann, C. Dirschl, O. Erling, M. Hausenblas, R. Isele, J. Lehmann, M. Martin, P. N. Mendes, B. Van Nuffelen, *et al.*, “Managing the life-cycle of linked data with the LOD2 stack”, in *International semantic Web conference*, Springer, 2012, pp. 1–16.

[17] M. H. de Souza Bomfim and D. Schwabe, “Design and implementation of linked data applications using SHDM and synth”, in *International Conference on Web Engineering*, Springer, 2011, pp. 121–136.

[18] L. Baronyai, K. Haller, and S. Gamerith, “Linked Open Data at the Vienna University of Technology – A case study about research data”, 2016.

[19] T. Berners-Lee, *Linked data, 2006*, 2006. [Online]. Available: `https://www.w3.org/DesignIssues/LinkedData.html`.

References to websites

[20] EUCLID, *EUCLID — EdUcational Curriculum for the usage of LInked Data*, [Online; accessed 5-September-2016], 2012-2014. [Online]. Available: http://euclid-project.eu/index.html.

[21] E. Union, *Framework Programmes for Research and Technological Development*, [Online; accessed 8-September-2016], 2012-2014. [Online]. Available: https://ec.europa.eu/research/fp7/index_en.cfm.

[22] EUCLID, *About Euclid*, [Online; accessed 5-September-2016], 2012-2014. [Online]. Available: http://euclid-project.eu/about/project-description.html.

[23] ——, *EUCLID — Chapter 5: Building Linked Data Applications*, [Online; accessed 5-September-2016], 2012-2014. [Online]. Available: http://euclid-project.eu/modules/chapter5.html.

[24] M. d'Aquin, F. Zablith, E. Motta, O. Stephens, S. Brown, S. Elahi, and R. Nurse, *The LUCERO project – About*, [Online; accessed 15-September-2016], 11 Jun 2010. [Online]. Available: http://lucero-project.info/lb/about/index.html.

[25] ——, *The LUCERO project – Tabloid*, [Online; accessed 15-September-2016], 1Jun 2011. [Online]. Available: http://lucero-project.info/lb/tabloid/index.html.

[19] T. Berners-Lee, *Linked data, 2006*, 2006. [Online]. Available: https://www.w3.org/DesignIssues/LinkedData.html.