Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS

Karel Klíma

Customizable Linked Data Browser

Department of Software Engineering

Supervisor of the master thesis: doc. Mgr. Martin Nečaský, Ph.D.

Study programme: Informatics

Specialization: Software Systems

Prague 2015

I would like to thank my supervisor, doc. Mgr. Martin Nečaský, Ph.D., for the patient guidance, encouragement and advice he has provided throughout my time as his student. I have been extremely lucky to have a supervisor who cared so much about my work, and who responded to my questions and queries so promptly.

Název práce: Přizpůsobitelný prohlížeč pro Linked Data

Autor: Karel Klíma

Katedra / Ústav: Katedra softwarového inženýrství

Vedoucí diplomové práce: doc. Mgr. Martin Nečaský, Ph.D.

Abstrakt: Cílem této práce je prozkoumat a identifikovat klíčové aspekty problematiky explorace dat publikovaných v souladu s principy Linked Data a navrhnout a implementovat webovou aplikaci, která bude sloužit jako prohlížeč těchto dat, jenž bude dále zahrnovat možnosti vyhledávání a modifikace vzhledu zobrazení procházených dat. Na rozdíl od stávajících řešení budou moci uživatelé aplikace poskytnout vlastní šablony, které mohou definovat alternativní zobrazení určitých typů dat v rámci prostoru Linked Data, případně budou moci upravovat styl aplikace a prezentace dat pomocí jiných metod.

Klíčová slova: Linked Data, RDF, prohlížeč dat

Title: Customizable Linked Data Browser

Author: Karel Klíma

Department / Institute: Department of Software Engineering

Supervisor of the master thesis: doc. Mgr. Martin Nečaský, Ph.D.

Abstract: The aim of this thesis is to identify key requirements for exploring Linked Data and design and implement a web application which serves as a Linked Data browser, including search and customization features. In comparison to existing approaches it will enable users to provide templates which define a visual style for presentation of particular types of Linked Data resources. Alternatively, the application can provide other means of altering the presentation of data or the appearance of the application.

Keywords: Linked Data, RDF, data browser

Contents

1 Introduction

Since its early years, the World Wide Web has been the most invaluable source of information in the entire world. The ever growing body of interlinked documents comprising almost a billion of active websites contains unimaginable amount of information. Unfortunately, the data are not always easily accessible or recognizable, as they often lack the most important thing to understand it – the semantics.

Traditionally, the web documents served only one purpose – to present the information to the reader, or, in other words, to take raw data, enhance it with templates and styles and display it to the user in a fashionable way. However convenient this transformation is for the user, one of the gravest side effects is that the data generally lose some of their semantics, if not all of it, rendering them complicated for machine processing. The reason is that the key technologies used for building web documents were not really designed with data semantics in mind. The HTML (HyperText Markup Language) specification defines some basic content relations, for example it provides means to annotate headings, paragraphs, lists or tables, which is clearly not sufficient when managing and presenting complex data.

Over the years, numerous projects and initiatives that tried to incorporate semantics to HTML arose, the most prominent of them being the Semantic Web, [1] which is an extension of WWW through standards that promote common data formats and exchange protocols. The standards provided can be used to describe data entities in existing web documents, but the most important part is the introduction of relations between particular data entities and data sets, resulting in a *web* of data. This collection of interrelated datasets is referred to as Linked Data. The main philosophy behind Linked Data is that by interconnecting data sets the total value of the data is increased – many common things are described in multiple data sets, and by connecting these data sets with linking identifiers, it is possible to leverage the shared knowledge. In other words, exploring one particular data entity permits the user to follow interconnecting links to related entities, in theory, indefinitely.

Technically, Linked Data refers to data published on the Web in such a way that it is machine-readable, its meaning is explicitly defined, it is linked to other external data sets and can in turn be linked to from external data sets.

	Web of Documents	*Web of Data*
Analogy	A global file system	A global database
Primary objects	Documents	Things, or descriptions of things
Links between	Documents	Things, including documents
Degree of structure in objects	Low	High
Semantics of content and links	Implicit	Explicit
Designed for	Human consumption	Machines first

Table 1 Comparison of the Web of Documents and Web of Data

The Linked Data ecosystem is based on the following principles providing guidance for publishing data on the Web, introduced by Tim Berners-Lee: [3]

1. Use URIs[1] to identify things.
2. Use HTTP[2] URIs so that these things can be looked up.
3. Provide useful information about what a name identifies when it is looked up, using open standards.
4. Refer to other things using their HTTP URI-based names.

Thanks to these principles, it is possible to consider the Web of Data to be a gigantic global database, existing in parallel to the Web of Documents.

1.1 Problem statement

With significant volumes of Linked Data being published on the Web, numerous efforts are underway to research and build applications that exploit this web of data. The applications can be classified to three categories: Linked Data browsers, Linked Data search engines, and domain-specific Linked Data applications. [2] Application from each category are developed with different goals in mind – the browsers' purpose is to display generic data entities the user knows about and provide navigation to

[1] Uniform Resource Identifier – a strings that identifies a particular entity, the set of all URIs is a superset of all URLs (Uniform Resource Locator)
[2] HyperText Transfer Protocol

related entities, the search engines aim to look for data entities the user does not know and finally the domain-specific Linked Data applications usually provide means and methods to present data from particular fields.

However, for effective exploration of Linked Data, it is necessary to combine these three approaches and create a software product that leverages all the mentioned applications' features and strengths.

In comparison to other databases, Linked Data have specific features – both positive and negative – that may represent obstacles application developers have to overcome. Some of the biggest problems are these:

1. Distributed nature of Linked Data – plurality of data sources.
2. Decentralized nature of data semantics – there is no universal data specification, the publishers can provide own specifications or utilize existing ones, often combining both of these principles.
3. Incomplete or invalid data – opposed to relational databases, Linked Data entities may not abide the data specification.
4. Infinity – Linked Data entities are potentially infinite, there is not any limit on the number of relations that connect a particular data entity to others.

These problems illustrate that the world of Linked Data is a world of uncertainty, and any developer of applications leveraging the Linked Data must bear this paradigm in mind.

Due to this indeterminism of Linked Data, their exploration and adequate presentation to users is undoubtedly a non-trivial task.

1.2 Goals

The aim of this thesis is to identify key requirements for exploring Linked Data and design and implement a web application which serves as a Linked Data browser, including search and customization features.

In comparison to existing approaches it will enable users to provide templates which define a visual style for presentation of particular types of Linked Data resources. Users will be allowed to define their own templates. Before showing a chosen resource to a user, the browser will identify available templates which can be applied to the resource. If there are more templates identified, the browser will use the priorities defined by the user to choose an appropriate template.

One of the main goals of the thesis it to identify existing or develop own language for specification of templates.

1.3 Thesis outline

The structure of the thesis is organized as follows.

The second chapter provides the reader with an overview of essential Linked Data principles, technologies and related resources.

In the third chapter, existing Linked Data browsers are identified and compared, and a model of an ideal browser is presented.

The fourth chapter provides analysis of the problem and defines general requirements and gives possible solutions. Based on the analysis, the fifth chapter suggest a design of the browser and presents key design and implementation decisions.

The sixth chapter evaluates the developed application from the viewpoints of customization features of the application and performance.

Finally, the conclusion assesses achieved goas and provides an overview of possible future work.

2 Preliminaries

In this chapter basic concepts and definitions that are used throughout this thesis are introduced. Discussing these topics in-depth is beyond the scope of this thesis; only the required basic information will be described.

2.1 Resource Description Framework

The Resource Description Framework (RDF), is a W3C[3] recommendation for representation resources in the World Wide Web. RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a “triple”). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications. [4]

Using the RDF, a particular data resource is described using an unordered set of triples. Each triple is of form (*subject*, *predicate*, *object*). The triple denotes that the resource (*subject*) has a relation (*predicate*) to another resource (*object*). This linking structure forms a directed, labeled graph, where the edges represent the named link between two resources, represented by the graph nodes.

RDF can be expressed using various formats, including:

- Turtle[4] – compact and human-friendly language
- RDFa[5] – enhancing web pages by encoding special RDF attributes inside HTML source code
- JSON-LD[6] – JavaScript Object Notation for Linking Data

2.1.1 Turtle serialization

Consider the sentence: “Tim Berners-Lee is the author of the Internet.” This phrase contains two resources – Tim Berners-Lee and the Internet, (*subject* and *object*) whereas the part “is the author of” represents a relation between those resources (*predicate*). The following example shows the expression of these facts in Turtle:

[3] World Wide Web Consortium, http://www.w3.org
[4] Turtle: http://www.w3.org/TR/turtle/
[5] Resource Description Framework in Attributes: http://www.w3.org/TR/xhtml-rdfa-primer/
[6] JSON for Linking Data: http://www.w3.org/TR/json-ld/

```
<http://example.org/person/Tim_Berners-Lee>
    <http://example.org/relation/author>
    <http://example.org/thing/The_Internet> .
```

Figure 1 Example of Turtle syntax

It is necessary to emphasize that the predicate is a resource too, and is therefore identified by its own URI. Being a resource, it can be a subject or an object of relations too, even though its primary reason for existence is to express a property.

2.1.2 RDFa serialization

RDFa (Resource Description Framework in Attributes) is a technique that allows adding structured data to HTML pages directly: it provides a set of markup attributes to augment the visual information on the Web with machine-readable hints. The key feature of this serialization is that it permits expressing RDF data using RDFa directly in HTML without breaking the HTML code, and is therefore perfect for enriching existing human-readable content of Web pages with machine-readable data.

```
<html>
<head> ... </head>
<body>
<div resource="http://example.com/alice/posts/trouble_with_bob">
  <h2 property="http://purl.org/dc/terms/title">The Trouble with
Bob</h2>
  <p>Date: <span
property="http://purl.org/dc/terms/created">2011-09-10</span></p>
  ...
</div>
</body>
</html>
```

Figure 2 Example of RDFa syntax

The example above is equivalent to the following Turtle markup:

```
@prefix dcterms: <http://purl.org/dc/terms/>

<http://example.com/alice/posts/trouble_with_bob>
    dcterms:title “The Trouble with Bob” ;
    dcterms:created “2011-09-10”^xsd:date .
```

Figure 3 Advanced example of Turtle syntax

2.1.3 JSON-LD serialization

JSON-LD is a lightweight Linked Data format. It is easy for humans to read and write. It is based on the already successful JSON format and provides a way to help JSON data interoperate at Web-scale. JSON-LD is an ideal data format for programming environments and REST Web services. [5] Being expressed using the syntax of JSON, it is implicitly supported by every JavaScript platform, i.e. web browsers or Node.js server side environment.

```
{
  "@context": "http://json-ld.org/contexts/person.jsonld",
  "@id": "http://dbpedia.org/resource/John_Lennon",
  "name": "John Lennon",
  "born": "1940-10-09",
  "spouse": "http://dbpedia.org/resource/Cynthia_Lennon"
}
```

Figure 4 Example of JSON-LD syntax

2.2 Triple store

A triple store is a database designed to store and retrieve identities that are constructed from triplex collections of strings (sequences of letters). These triplex collections represent a subject-predicate-object relationship that more or less corresponds to the definition put forth by the RDF standard. [6]

Triple stores can be both primary and secondary sources of data. In addition to serving as a traditional original data layer, they can be used for storing RDF data originally provided for example as RDFa enhancement of HTML Web pages. Mirroring the data to the triple store permits users to query the data, which is otherwise hard at best, considering the data can be scattered across multiple Web documents.

2.3 SPARQL Query Language

SPARQL[7] is an RDF query language, that is, a semantic query language for databases, able to retrieve and manipulate data stored in RDF format. It was made a standard by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium, and is recognized as one of the key technologies of the semantic web. [7]

In case of queries that read data from the database, SPARQL language specification defined four different query types that serve different purposes.

2.3.1 SELECT query

Probably the most common SPARQL query type is the SELECT query. Similar to the SQL SELECT queries in relational databases, it too returns a table-formatted collection of values, one row for every tuple that corresponds to the possible partial result of the query, i.e. the specified conditions. The following example shows a query to get name and email values of every person in the data set.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
  ?person foaf:mbox ?email.
}
```

Figure 5 Example of a SELECT SPARQL query

A result of a particular SELECT query can be further modified, restricted or aggregated by parameters common in the world of relation databases, for example GROUP BY, ORDER BY, LIMIT or OFFSET.

2.3.2 CONSTRUCT query

A CONSTRUCT query is used to extract information from a SPARQL endpoint and transform the results into valid RDF – a graph of triples. This is useful for further serialization of the result that persists the RDF properties, for example to JSON-LD

[7] SPARQL Protocol and RDF Query Language. http://www.w3.org/TR/sparql11-overview/

notation, which represent an ideal transmission format for web based Linked Data applications, as the JSON is already widely used in this field.

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
CONSTRUCT {
  ?X vCard:FN ?name .
  ?X vCard:TITLE ?title .
}
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
  OPTIONAL { ?X foaf:name ?name . FILTER isLiteral(?name) . }
  OPTIONAL { ?X foaf:title ?title . FILTER isLiteral(?title) . }
}
```

Figure 6 Example of a CONSTRUCT SPARQL query

The example presented in Figure 6 describes a method of translating data from one vocabulary to another. A CONSTRUCT query is obviously versatile and can be utilized for various use cases, the most prominent one being transformation of any of the bound variables (prefixed by a question mark, for example the *?name* variable) to valid RDF data. For example, using this method it is possible to convert table-formatted data projected by the SELECT query into RDF triples.

```
PREFIX : <http://custom.vocabulary/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT {
    <http://example.com/People/Karel> :property ?property
}

WHERE {
    SELECT DISTINCT ?property
    WHERE {
        <http://example.com/People/Karel> ?property [] .
    }
    LIMIT 5
}
```

Figure 7 Advanced example of a CONSTRUCT SPARQL query

The previous example also illustrates the possibility to leverage SELECT specific properties[8] like LIMIT or ORDER BY while maintaining the RDF-compliant output provided by CONSTRUCT expression by introducing a custom resource that acts as a property.

2.3.3 ASK query

Applications can use the ASK form to test whether or not a query pattern has a solution. No information is returned about the possible query solutions, just whether or not a solution exists. [8] Thus, the returning value is either *true* or *false*.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
ASK  { ?x foaf:name  "Alice" }
```

Figure 8 Example of an ASK SPARQL query

2.3.4 DESCRIBE query

The DESCRIBE form returns a single result RDF graph containing RDF data about resources. This data is not prescribed by a SPARQL query, where the query client would need to know the structure of the RDF in the data source, but, instead, is determined by the SPARQL query processor. [8] The DESCRIBE form takes each of the resources identified in a solution and assembles a single RDF graph by taking whatever information is available about the particular resources, including the target RDF Dataset.

The fact that the DESCRIBE query is dependent on the configuration of the query service, and therefore not standardized, renders it rather difficult to be used in Linked Data applications, as the resulting graph of such a query may differ on various platforms, even if the underlying data were identical.

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
DESCRIBE ?x WHERE { ?x foaf:mbox <mailto:alice@org> }
```

Figure 9 Example of a DESCRIBE SPARQL query

[8] It is actually possible to use LIMIT, ORDER BY and other clauses in a SPARQL CONSTRUCT query, but it behaves differently from the SELECT query, for example the LIMIT clause limits the total number of fetched triples, which can cause the resulting graph or array of graphs to be incomplete. In addition, the output of the CONSTRUCT query is a graph, which is by definition unordered.

2.4 SPARQL Query Service

A SPARQL Query Service (often referred to by the informal term SPARQL endpoint) is a service occupying a specific URI to which SPARQL queries can be sent using the HTTP protocol and which returns results to those queries as a response. Generally, it is a service associated with a particular graph store, usually performing queries on existing local datasets. From the users' point of view it can be seen as a Web interface of a hidden RDF graph store.

It is common for a SPARQL endpoint to show a HTML form and enable the user to input the query directly if no query is submitted to the endpoint's URI as a parameter of a GET or POST HTTP request.

With the SPARQL 1.1 update the notion of a SPARQL Query Services becomes more general since it is possible to send Updates as well as Queries using SPARQL or to use HTTP operation in a RESTful manner. [9]

2.5 Representational State Transfer

Representational State Transfer (REST) is a software architecture style for building scalable web services. [10] So called RESTful systems typically communicate over the HTTP protocol with the same HTTP instructions (GET, POST, PUT, DELETE) that conventional Web browsers use to communicate with remote servers while retrieving or sending data.

A RESTful interface consists of a collection of resources, upon which the particular requests are performed, each resource typically corresponds with a particular URI. Generally, the interface is used for CRUD operations *crate, read, update, delete* that correspond to the aforementioned HTTP instructions.

As the name suggest, the RESTful services are stateless, the client context is not stored on the server between requests and must therefore be included in each and every request the client makes, should the state be maintained.

3 Related work

This chapter examines and compares existing solutions for browsing Linked Data. The chapter is organized into three parts. Firstly, general requirements for exploring Linked Data are presented and a model example of an ideal Linked Data browser is suggested. The second part provides comparison of existing browsers from the perspective of properties of the model ideal solution. Lastly, each existing product is briefly described and its fundamental and distinctive features are emphasized.

3.1 Fundamental features of an ideal Linked Data browser

Before compiling a list of basic properties of an ideal Linked Data browser, it is desirable to take a step back and think about the heart of the matter – the purpose of existence of such a browser. There is no need to reinvent the wheel. Tim Berners-Lee, the father of the Internet and author of Linked Data principles already described this matter in the proposal of the Tabulator project:

"*A goal of a generic semantic browser is serendipitous re-use: while searching for a solution to a problem, I discover and successfully use data, which I hadn't thought of using. I may not have known it existed, or I may not have realized that it was connected to the problem at hand. On the other hand, I may have known it existed, but not realized that it was now on the semantic web of linked data. A similar goal is that while idly browsing with a general interest in mind, I discover an interesting possibility which had not occurred before. The goal then is that, as with the HTML web, the value is the re-use of information in ways that are unforeseen by the publisher, and often unexpected by the consumer.*" [15]

With this goal in mind, we can now discuss desired properties of an ideal generic Linked Data browser further.

3.1.1 Target platform

Considering the fact that the Web of Data is a part of the Internet, it makes perfect sense to access it using a web browser, a software that is already commonly used to access documents on the web. It is therefore more than suitable for our model Linked Data browser to be a web-based application.

One can argue that platform-specific applications provide better user experience, which is an essential component of any good software product. This proposition, however right it may have been in the past, has been negated by an array of technologies that have risen in recent years, especially with a connection to new kinds of devices accessing the web such as mobile phones or tablets. With the ascent of those devices a new user interface design methodology was invented – a responsive web design. It is an approach aimed at crafting web sites to provide an optimal viewing and interaction experience – easy reading and navigation with a minimum of resizing, panning, and scrolling – across a wide range of devices, especially with a regard to a display size. Today, it is absolutely necessary to implement such a design when developing a web application, as the Internet traffic from the non-traditional devices is ever increasing.

3.1.2 Source of the data

Generally, there are two major approaches to extract information from the Web of Data: the first is direct querying of a SPARQL endpoint, the second is dereferencing a URI containing either RDF data or a HTML web page enhanced with RDF annotations. In the early years of the Linked Data phenomenon existence, most browsers accessed the semantic data directly using the second approach, for example applications such as Tabulator, [11] Marbles, [12] Disco[9], Zitgist data viewer or OpenLink Data Explorer.[10] Unfortunately, most of these browsers are not maintained anymore and are superseded by modern solutions leveraging SPARQL endpoint access.

Firstly, accessing Linked Data through a SPARQL query service is arguably easier and much more effective than parsing web pages with each user request. Secondly, websites that publish data in RDF format are presumably backed by an RDF data store and can therefore expose their data through a SPARQL endpoint. Even it is not the case and the RDF data published on a website are indeed the original source of the information, it is suitable for that information to be extracted only once using data mining techniques and to be stored in a separate database. In fact, chances are that if there is any information in the form of RDF on a website, it is likely to be already mirrored on some of the existing public data stores. Finally, it may be easier to build

[9] Disco. http://www4.wiwiss.fu-berlin.de/bizer/ng4j/disco/
[10] OpenLink Data Explorer. http://www.w3.org/wiki/OpenLinkDataExplorer

a quality data scraper[11] and a separate data browser than an application combining both functionalities.

To summarize, our model of an ideal Linked Data browser should embrace the SPARQL access when obtaining the data, as the data mining tasks can and should be preferable performed by other tools. Besides, there are already far more superior applications tailored specifically to explore the data embedded in web sites and providing the best user experience possible – the web browsers.

3.1.3 Exploration capabilities

An ideal Linked Data browser should provide users with exploratory features not only to browse known data, but first and foremost to discover new information and connections between data. Basic search functionality should be available as well, for users generally do not know a URI of a resource they look for, and even if they do, it might be convenient to type some key words and reach the desired resource, than typing its full URI.

Browsing of Linked Data essentially means traversing from one resource to another, which requires presenting all the possible relations for each resource that is displayed to the user. Considering the fact that in the world of Linked Data the relations are in fact edges of an oriented graph, it is necessary to examine both “outgoing” and “incoming” connections, that is, relations where a particular resource figures both as a subject and as an object.

Also, it is common that there are multiple data sets contained in separate graphs present in one data store that is served by a SPARQL endpoint. Should such circumstances appear, the browser ought to display the array of graphs to the user and enable the user to select one or more of them to restrict the data domain to browse or search in.

3.1.4 Big data management

As was already explained in the introduction of this thesis, one of the properties of Linked Data is an unrestricted number of connection between resources. That means that one resource may contain thousands or millions of connections. The browser

[11] A tool used to extract data from web resources like web pages.

should assume such cases and provide users with means to explore and understand the relations.

3.1.5 Customization options

The browser should enable the data presentation to be customized, that is, to provide means of overriding default display of data for particular data types. If plausible, some form of customization should be available to end users that do not necessarily possess extensive knowledge of RDF of Linked Data phenomena.

3.2 Comparison of existing Linked Data browsers

The table below shows a comparison of identified major existing solutions. The scope of the comparison is based on the features of an ideal Linked Data browser described in previous paragraphs.

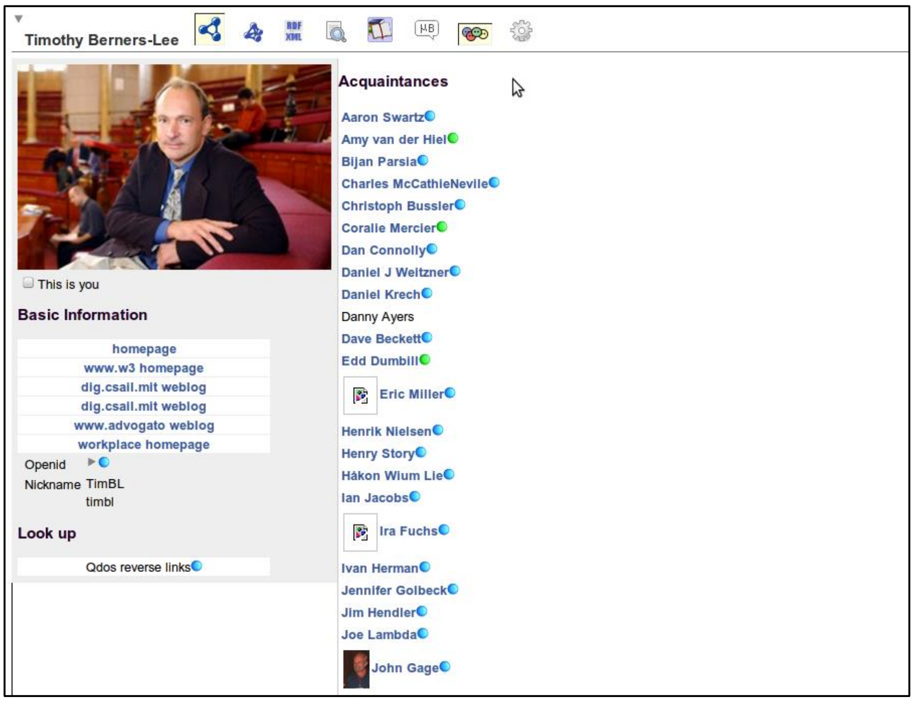
Comparison of existing Linked Data browsers Legend: W ~ Web E ~ Endpoint B ~ Both S ~ Silverlight	Tabulator	Marbles	Explorator	Quick and Dirty Browsers	OpenLink Faceted Browser	Oobian Insight	LODLive	Graphity Client	Justinian
Target platform	W	W	W	W	W	S	W	W	W
Data source	W	W	B	B	E	E	E	E	E
Traversing capabilities	●	●	●	●	●	●	●	●	●
Functionality out of the box	●	●	●	●	●	●	●		
Multiple data source support	●	●	●				●		
Search capabilities				●	●	●			●
Inverse connections			●		●	●	●	●	●
Graph restrictions			●		●			●	●
Smart big data handling			●			●		●	●
Customization options	●							●	●
Responsive features								●	●

Table 2 Comparison of existing Linked Data browsers

3.3 Description of existing Linked Data browsers

3.3.1 Tabulator

The Tabulator project[12] is a generic data browser and editor. Using outline and table modes, it provides a way to browse RDF data on the web. It can be used as a browser extension – currently available for Firefox and Chrome (experimental) – or as a standalone JavaScript library. It is extensible through various panes which act as read write data apps, therefore offering various views on particular data. [11] The main idea behind the project is to present data in an outline mode, opposed to the traditional circle-and-arrow diagrams most RDF visualizers use.



Timothy Berners-Lee

This is you

Basic Information

homepage
www.w3 homepage
dig.csail.mit weblog
dig.csail.mit weblog
www.advogato weblog
workplace homepage

Openid
Nickname TimBL
timbl

Look up

Qdos reverse links

Acquaintances

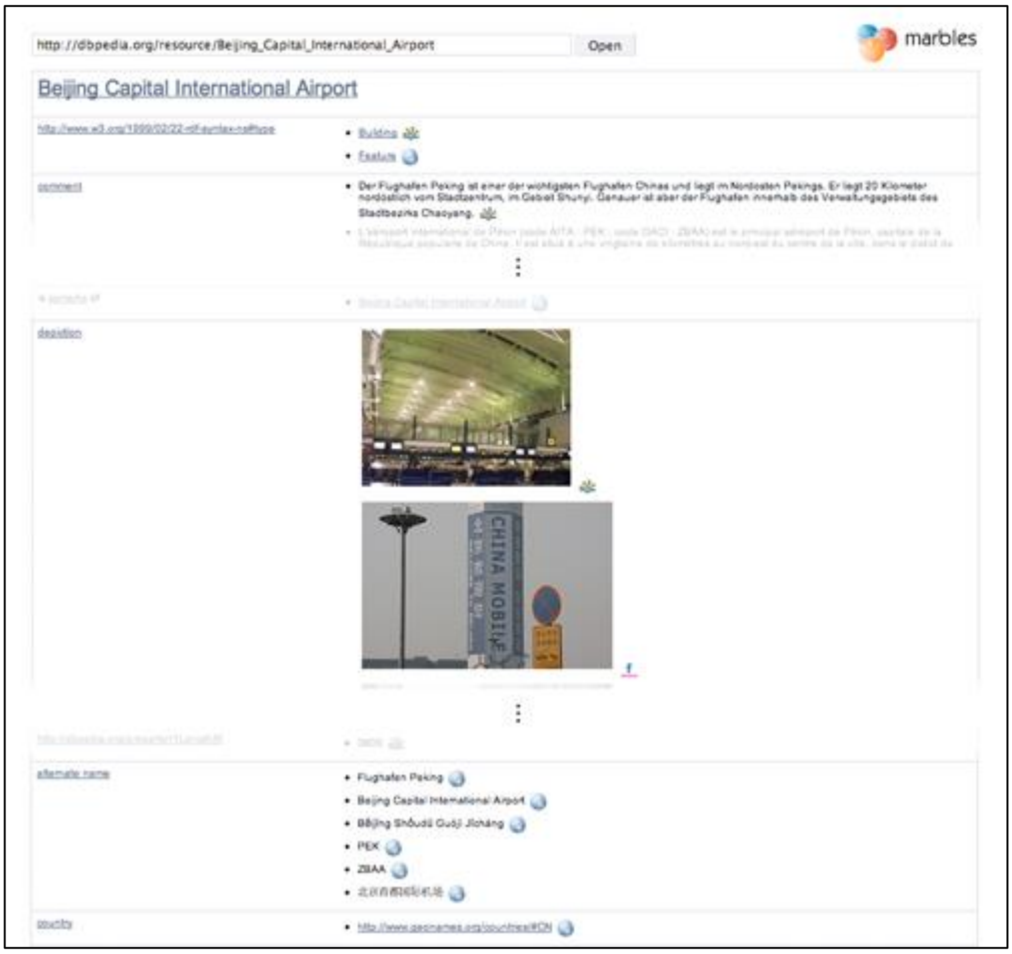
Aaron Swartz
Amy van der Hiel
Bijan Parsia
Charles McCathieNevile
Christoph Bussler
Coralie Mercier
Dan Connolly
Daniel J Weitzner
Daniel Krech
Danny Ayers
Dave Beckett
Edd Dumbill
Eric Miller
Henrik Nielsen
Henry Story
Håkon Wium Lie
Ian Jacobs
Ira Fuchs
Ivan Herman
Jennifer Golbeck
Jim Hendler
Joe Lambda
John Gage

Figure 10 Tabularor Linked Data browser

[12] The Tabulator. http://www.w3.org/2005/ajar/tab

3.3.2 Marbles

Marbles[13] is a server-side application that formats Semantic Web content for HTML clients using Fresnel[14] lenses and formats. By performing all formatting, data retrieval and storage activities on the server side rather than on a potentially thinly equipped client, the view generation can touch on large amounts of data and requests can be answered relatively quickly. [12]


http://dbpedia.org/resource/Beijing_Capital_International_Airport
Open
marbles

Beijing Capital International Airport

http://www.w3.org/1999/02/22-rdf-syntax-ns#type	• Building • Feature
comment	• Der Flughafen Peking ist einer der wichtigsten Flughafen Chinas und liegt im Nordosten Pekings. Er liegt 20 Kilometer nordöstlich vom Stadtzentrum, im Gebiet Shunyi. Genauer ist aber der Flughafen innerhalb des Verwaltungsgebiets des Stadtbezirks Chaoyang. • L'aéroport international de Pékin [illegible]
[illegible]	• Beijing Capital International Airport
depiction	
[illegible]	• [illegible]
alternate name	• Flughafen Peking • Beijing Capital International Airport • Běijīng Shǒudū Guójì Jīchǎng • PEK • ZBAA • 北京首都国际机场
country	• http://www.geonames.org/countries/#CN

Figure 11 Marbles Linked Data browser

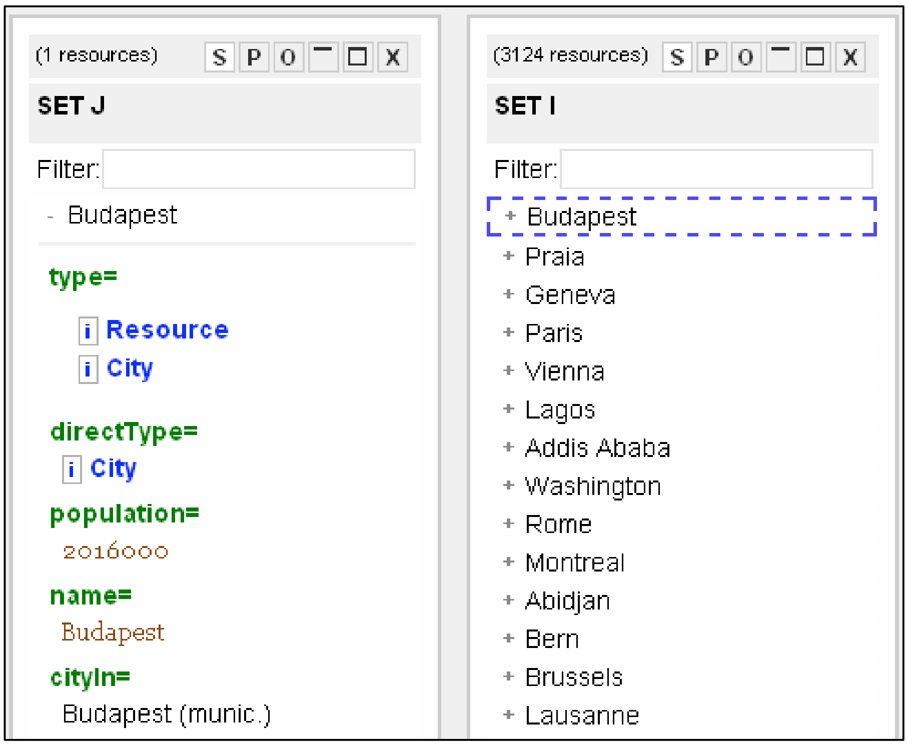
[13] Marbles. http://mes.github.io/marbles/

[14] Fresnel – Display Vocabulary for RDF. http://www.w3.org/2005/04/fresnel-info/

3.3.3 Explorator

Explorator is an open-source exploratory search tool for RDF graphs, implemented in a direct manipulation interface metaphor. It implements a custom model of operations, and also provides a Query-by-example interface. Additionally, it provides faceted navigation over any set obtained during the operations in the model that are exposed in the interface. It can be used to explore both a SPARQL endpoint as well as an RDF graph in the same way as traditional RDF browsers. [13]

In comparison to Tabulator, Explorator's target group of users are mostly IT specialists, as basic knowledge to RDF principles is required to effectively use the application. Explorator also does not provide any customization options for the data presentation.


(1 resources)

SET J

Filter:

- Budapest

type=

Resource

City

directType=

City

population=

2016000

name=

Budapest

cityIn=

Budapest (munic.)

(3124 resources)

SET I

Filter:

+ Budapest
+ Praia
+ Geneva
+ Paris
+ Vienna
+ Lagos
+ Addis Ababa
+ Washington
+ Rome
+ Montreal
+ Abidjan
+ Bern
+ Brussels
+ Lausanne

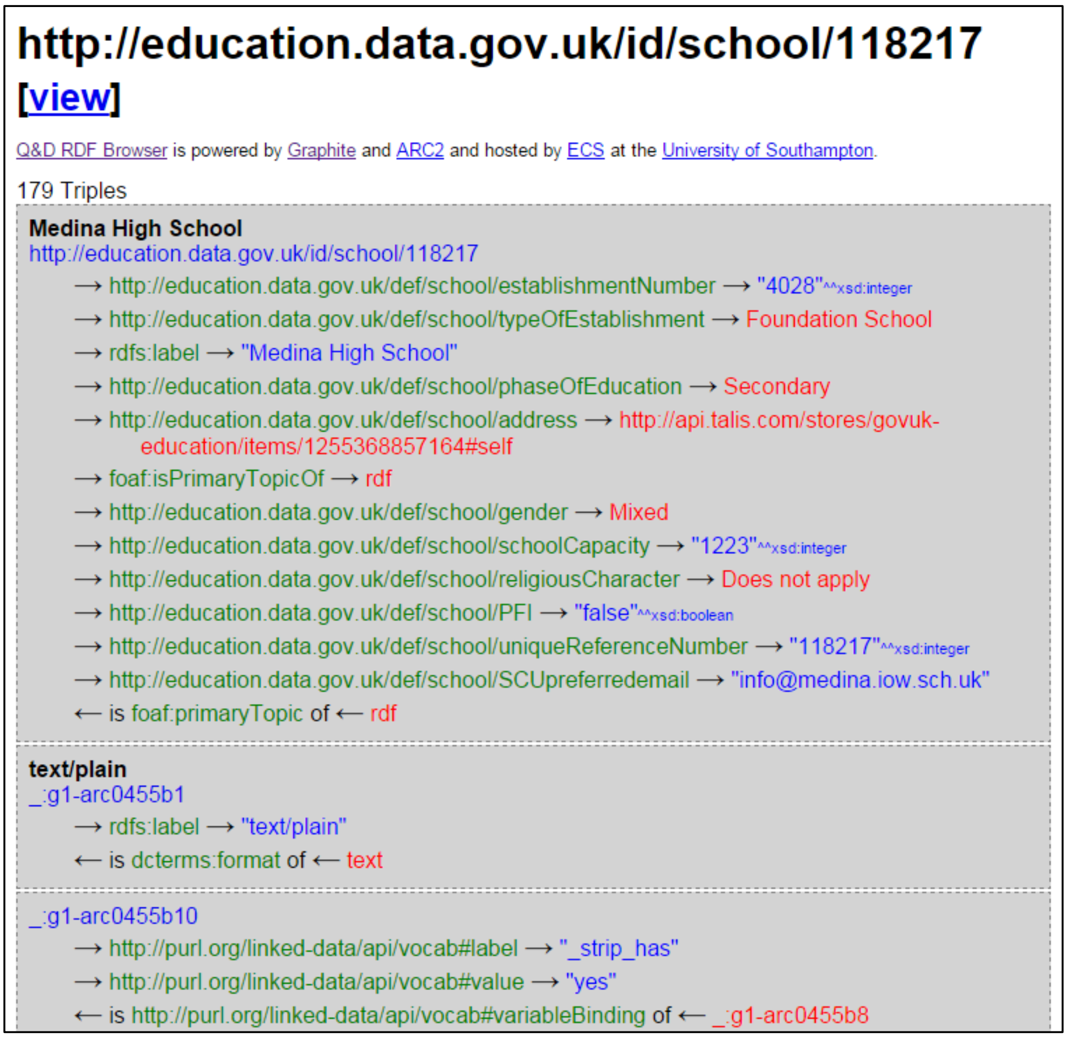
Figure 12 Explorator Linked Data browser

3.3.4 Quick and Dirty Browsers

Quick and Dirty RDF Browser[15] and Quick and Dirty SPARQL Browser[16] are simple wrappers around the Graphite PHP Linked Data Library.[17] The browsers can jointly explore both RDF data in a dereferenced URI and SPARQL endpoints.

The SPARQL browser can be used explore a selected endpoint by defining a particular resource URI or specifying a simple text search query.

Both browsers present the requested data by grouping the requested resource triples together and therefore creating an outline of the resource, but no further custom data views are provided, the data are displayed exactly in the form they are received.



http://education.data.gov.uk/id/school/118217 [view]

Q&D RDF Browser is powered by Graphite and ARC2 and hosted by ECS at the University of Southampton.

179 Triples

Medina High School
http://education.data.gov.uk/id/school/118217
→ http://education.data.gov.uk/def/school/establishmentNumber → "4028"^^xsd:integer
→ http://education.data.gov.uk/def/school/typeOfEstablishment → Foundation School
→ rdfs:label → "Medina High School"
→ http://education.data.gov.uk/def/school/phaseOfEducation → Secondary
→ http://education.data.gov.uk/def/school/address → http://api.talis.com/stores/govuk-education/items/1255368857164#self
→ foaf:isPrimaryTopicOf → rdf
→ http://education.data.gov.uk/def/school/gender → Mixed
→ http://education.data.gov.uk/def/school/schoolCapacity → "1223"^^xsd:integer
→ http://education.data.gov.uk/def/school/religiousCharacter → Does not apply
→ http://education.data.gov.uk/def/school/PFI → "false"^^xsd:boolean
→ http://education.data.gov.uk/def/school/uniqueReferenceNumber → "118217"^^xsd:integer
→ http://education.data.gov.uk/def/school/SCUpreferredemail → "info@medina.iow.sch.uk"
← is foaf:primaryTopic of ← rdf

text/plain
_:g1-arc0455b1
→ rdfs:label → "text/plain"
← is dcterms:format of ← text

_:g1-arc0455b10
→ http://purl.org/linked-data/api/vocab#label → "_strip_has"
→ http://purl.org/linked-data/api/vocab#value → "yes"
← is http://purl.org/linked-data/api/vocab#variableBinding of ← _:g1-arc0455b8

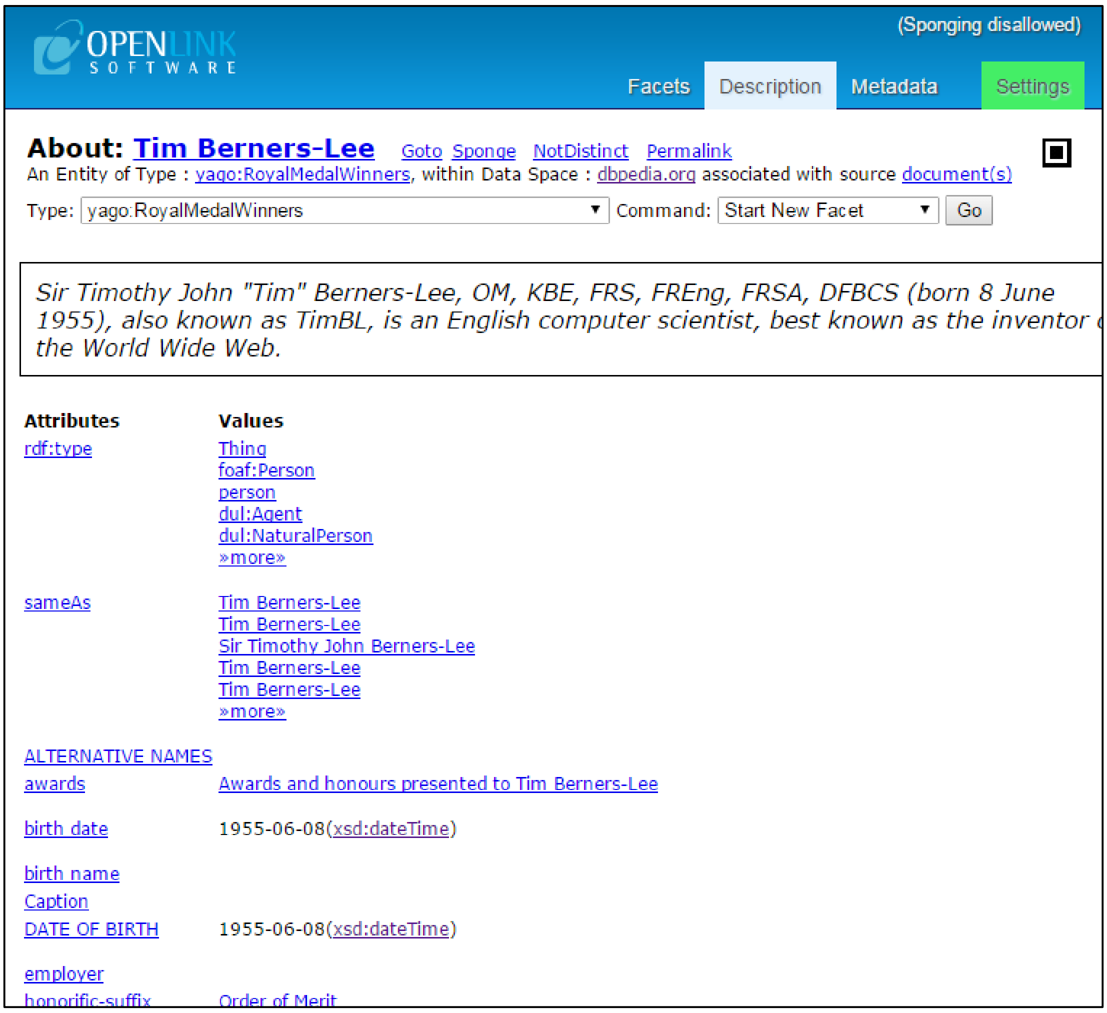
Figure 13 Quick and Dirty RDF browser

[15] Quick and Dirty RDF Browser. http://graphite.ecs.soton.ac.uk/browser/
[16] Quick and Dirty SPARQL Browser. http://graphite.ecs.soton.ac.uk/sparqlbrowser/
[17] Graphite PHP Linked Data Library. http://graphite.ecs.soton.ac.uk/

3.3.5 OpenLink Faceted Browser

The OpenLink Faceted Browser is an OpenLink Virtuoso Universal Server[18] built-in browser and search service. The browser is tied to the server instance, thus only capable to operate within the server databases.

A resource is described in a form of table, one row per one property. The browser does not handle resources with large amount of relations well, as it provides the option to paginate the result data within the scope of the whole resource, not within particular properties. Due to this limitation it is for example not capable to display all of the resource types of relations on a single page, which is a great disadvantage for the exploratory function of the product.



About: Tim Berners-Lee

An Entity of Type : yago:RoyalMedalWinners, within Data Space : dbpedia.org associated with source document(s)

Type: yago:RoyalMedalWinners Command: Start New Facet

Sir Timothy John "Tim" Berners-Lee, OM, KBE, FRS, FREng, FRSA, DFBCS (born 8 June 1955), also known as TimBL, is an English computer scientist, best known as the inventor the World Wide Web.

Attributes	Values
rdf:type	Thing foaf:Person person dul:Agent dul:NaturalPerson »more»
sameAs	Tim Berners-Lee Tim Berners-Lee Sir Timothy John Berners-Lee Tim Berners-Lee Tim Berners-Lee »more»
ALTERNATIVE NAMES	
awards	Awards and honours presented to Tim Berners-Lee
birth date	1955-06-08(xsd:dateTime)
birth name	
Caption	
DATE OF BIRTH	1955-06-08(xsd:dateTime)
employer	
honorific-suffix	Order of Merit

Figure 14 OpenLink Faceted Browser

[18] OpenLink Virtuoso Universal Server. http://virtuoso.openlinksw.com/

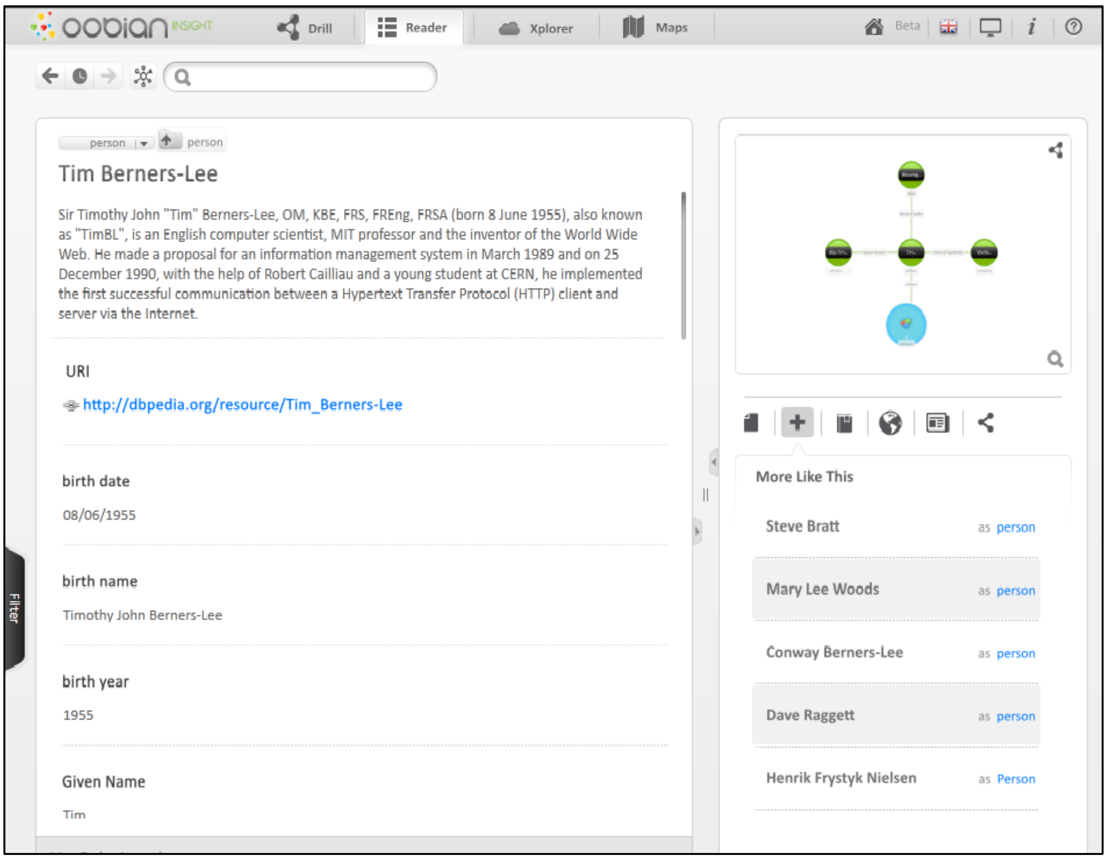
3.3.6 Oobian Insight

Oobian Insight[19] is a commercial Linked Data browser. As opposed to other browsers, it provides both graphical and textual user interface elements, therefore it is possible to explore the RDF graph visually through the collection of dots connected with lines and at the same time the data can be displayed sequentially in a reader mode.

In addition, the application provides a map mode that displays any available resources on a map, anywhere in the world.

As an exploratory tool, Oobian Insight stands out from all the other browsers, for it provides further connections outside of the world of Linked Data like integrated web search or news search and listing of similar resources.

Although the views of the data cannot be customized, the software product does a great job out of the box, as the information is presented in a comprehensive way.



Tim Berners-Lee

Sir Timothy John "Tim" Berners-Lee, OM, KBE, FRS, FREng, FRSA (born 8 June 1955), also known as "TimBL", is an English computer scientist, MIT professor and the inventor of the World Wide Web. He made a proposal for an information management system in March 1989 and on 25 December 1990, with the help of Robert Cailliau and a young student at CERN, he implemented the first successful communication between a Hypertext Transfer Protocol (HTTP) client and server via the Internet.

URI

http://dbpedia.org/resource/Tim_Berners-Lee

birth date

08/06/1955

birth name

Timothy John Berners-Lee

birth year

1955

Given Name

Tim

More Like This

Steve Bratt	as person
Mary Lee Woods	as person
Conway Berners-Lee	as person
Dave Raggett	as person
Henrik Frystyk Nielsen	as Person

Figure 15 Oobian Insight Linked Data Browser

[19] Oobian Insight. http://www.oobian.com/home/insight

3.3.7 LODLive

LODLive[20] is a graph visualization oriented Linked Data browser. It enables users to navigate the Linked Data graph by following oriented links (*relations*) between nodes (*data entities*).

Although the browsed does not provide separate text view interface, it is possible to display a list of all literal values associated by each resource by clicking an icon placed on the edge of each node.

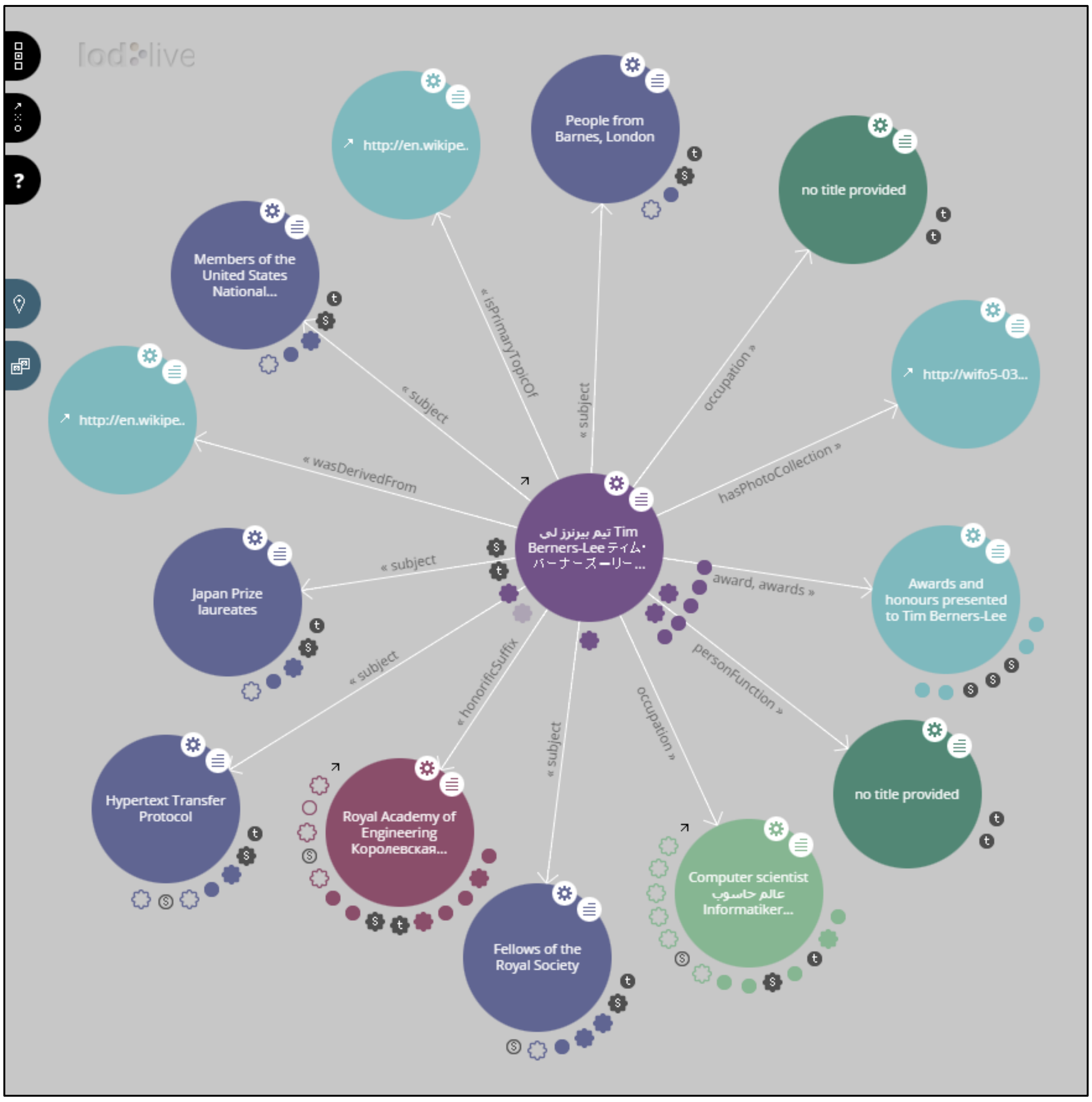

lod live
http://en.wikipe..
People from Barnes, London
no title provided
Members of the United States National...
http://wifo5-03...
http://en.wikipe..
تيم بيرنرز لي Tim Berners-Lee ティム・バーナーズ=リー...
Japan Prize laureates
Awards and honours presented to Tim Berners-Lee
no title provided
Hypertext Transfer Protocol
Royal Academy of Engineering Королевская...
Fellows of the Royal Society
Computer scientist عالم حاسوب Informatiker...
« isPrimaryTopicOf
« subject
occupation »
hasPhotoCollection »
« wasDerivedFrom
« subject
« subject
award, awards »
« subject
« honorificSuffix
« subject
occupation »
personFunction »

Figure 16 LODLive Linked Data browser

[20] LODLive. http://lodlive.it/

3.3.8 Graphity Client

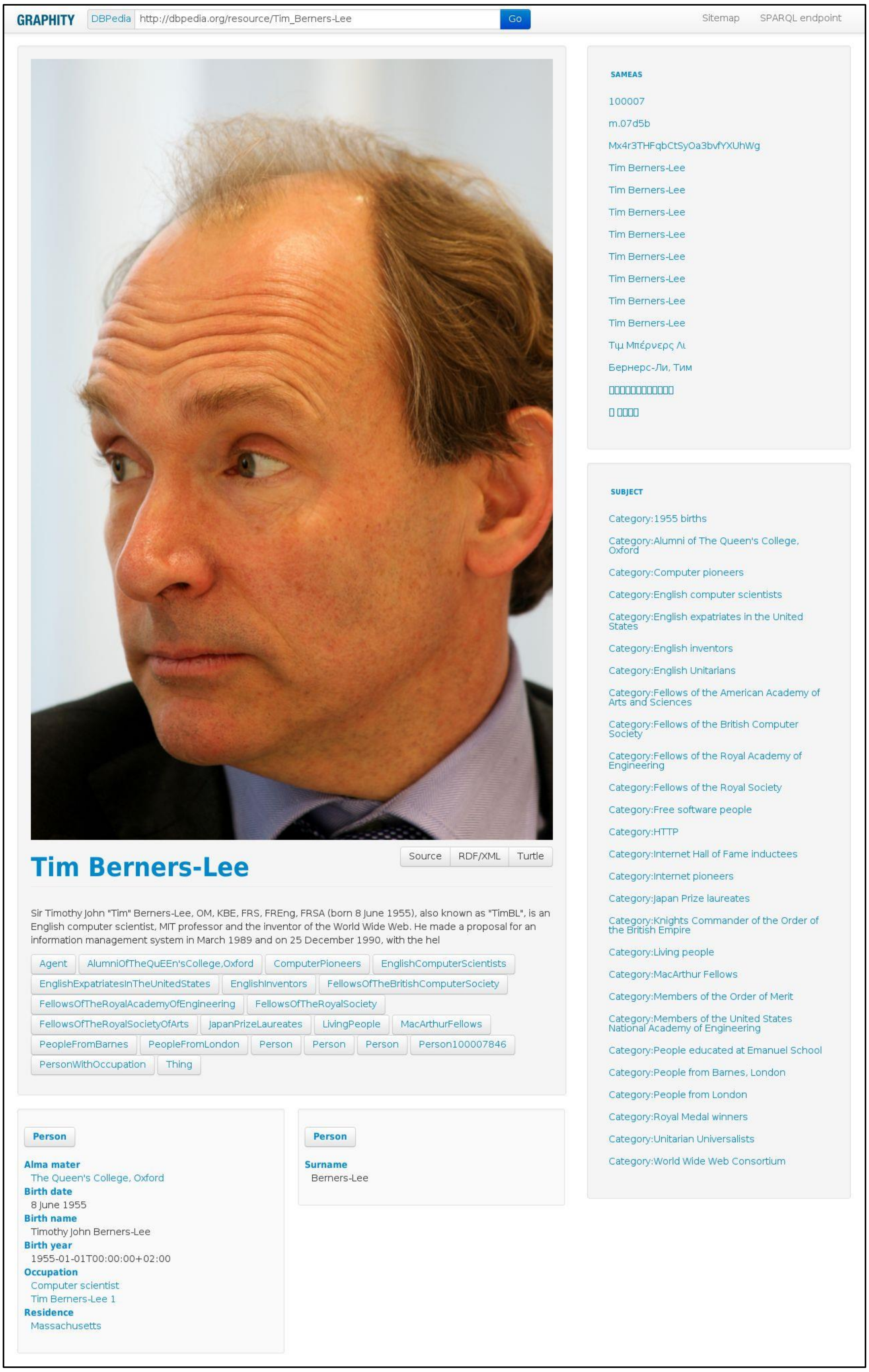
Graphity Client[21] is a part of Graphity Linked Data Platform.[22] The major goal of the project is to provide solution to build user-friendly web applications by managing data instead of writing code.

Graphity Client is not only a standalone Linked Data browser, but according to its creators it is an extensible Linked Data framework. It can run as a standalone end-user Web application or serve as either a frontend or backend component in data-driven applications. The Client is generic software, it can operate on Linked Data and RDF independently of the domain of the data. [14]

The Graphity browser is designed with customization in mind. The presentation of data is rendered using XSLT[23] templates, which can be overridden if required, therefore it is possible to alter both layout and design of particular properties displayed to a user. The application can be further enhanced by providing custom SPARQL queries alongside the XSLT templates.

This platform is the only one of all the aforementioned browsers that provide means to extend the application in a declarative, not an imperative approach. The developer only has to write declarative SPARQL and/or XSLT code to implement a specialized Client-based Web application.

[21] Graphity Client. http://graphityhq.com/technology/graphity-client
[22] Graphity Linked Data Platform. http://graphityhq.com/
[23] Extensible Stylesheet Language Transformations.


GRAPHITY DBPedia http://dbpedia.org/resource/Tim_Berners-Lee Go Sitemap SPARQL endpoint

Tim Berners-Lee

Source RDF/XML Turtle

Sir Timothy John "Tim" Berners-Lee, OM, KBE, FRS, FREng, FRSA (born 8 June 1955), also known as "TimBL", is an English computer scientist, MIT professor and the inventor of the World Wide Web. He made a proposal for an information management system in March 1989 and on 25 December 1990, with the hel

Agent | AlumniOfTheQuEEn'sCollege,Oxford | ComputerPioneers | EnglishComputerScientists | EnglishExpatriatesInTheUnitedStates | EnglishInventors | FellowsOfTheBritishComputerSociety | FellowsOfTheRoyalAcademyOfEngineering | FellowsOfTheRoyalSociety | FellowsOfTheRoyalSocietyOfArts | JapanPrizeLaureates | LivingPeople | MacArthurFellows | PeopleFromBarnes | PeopleFromLondon | Person | Person | Person | Person100007846 | PersonWithOccupation | Thing

Person

Alma mater
The Queen's College, Oxford
Birth date
8 June 1955
Birth name
Timothy John Berners-Lee
Birth year
1955-01-01T00:00:00+02:00
Occupation
Computer scientist
Tim Berners-Lee 1
Residence
Massachusetts

Person

Surname
Berners-Lee

SAMEAS

100007
m.07d5b
Mx4r3THFqbCtSyOa3bvfYXUhWg
Tim Berners-Lee
Tim Berners-Lee
Tim Berners-Lee
Tim Berners-Lee
Tim Berners-Lee
Tim Berners-Lee
Tim Berners-Lee
Tim Berners-Lee
Τιμ Μπέρνερς Λι
Бернерс-Ли, Тим
□□□□□□□□□□□□
□ □□□□

SUBJECT

Category:1955 births
Category:Alumni of The Queen's College, Oxford
Category:Computer pioneers
Category:English computer scientists
Category:English expatriates in the United States
Category:English inventors
Category:English Unitarians
Category:Fellows of the American Academy of Arts and Sciences
Category:Fellows of the British Computer Society
Category:Fellows of the Royal Academy of Engineering
Category:Fellows of the Royal Society
Category:Free software people
Category:HTTP
Category:Internet Hall of Fame inductees
Category:Internet pioneers
Category:Japan Prize laureates
Category:Knights Commander of the Order of the British Empire
Category:Living people
Category:MacArthur Fellows
Category:Members of the Order of Merit
Category:Members of the United States National Academy of Engineering
Category:People educated at Emanuel School
Category:People from Barnes, London
Category:People from London
Category:Royal Medal winners
Category:Unitarian Universalists
Category:World Wide Web Consortium

Figure 17 Graphity Client LD browser

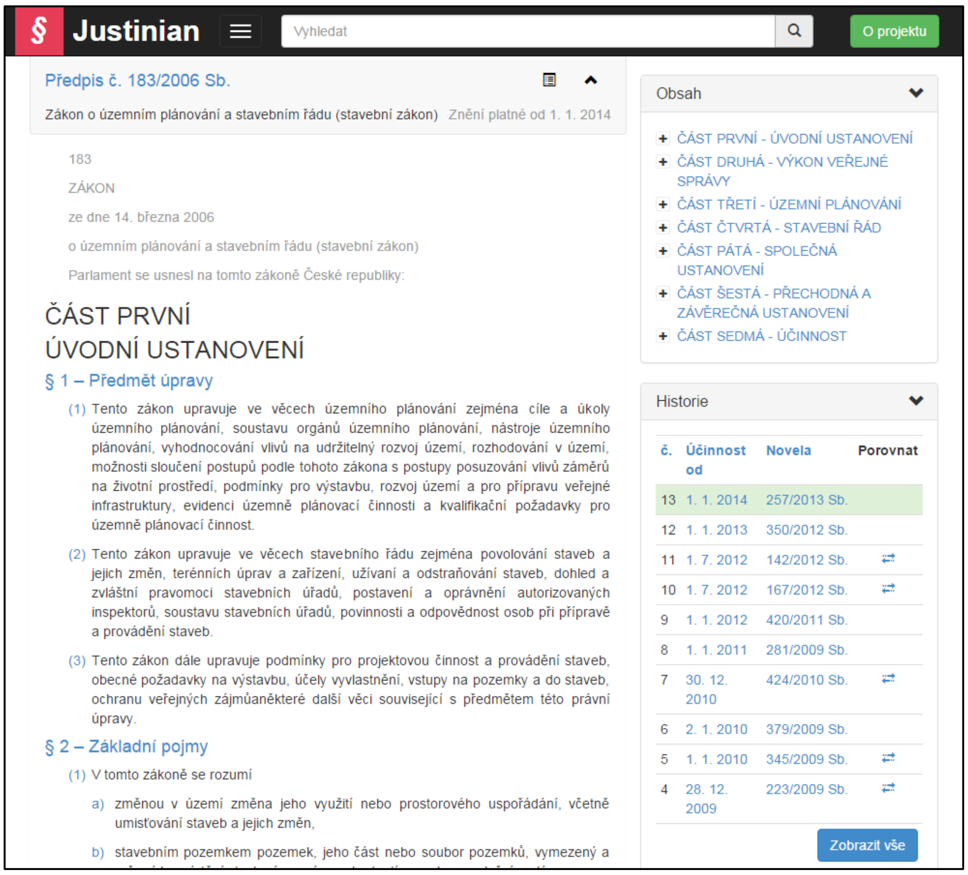
3.3.9 Justinian

Justinian[24] is a Linked Data browser designed to search and explore Czech legal acts, judicial decisions and related items. Although it is domain-specific, the framework it is built upon is domain-independent and may therefore be used to display generic data.

The architecture of the project is modular and consists of a series of widgets the authors call “miniapplications” used to display information about particular resources. Each miniaplication provides a custom template and may define a custom SPARQL query needed to fetch required data. In addition, miniapplications may set a display priority determining their position on the rendered web page.

The methods of extending Justinian are similar to those of extending the Graphity Linked Data Platform, although in case of Justinian some form of imperative programming is generally needed to achieve desired results.

The author of this thesis is one of the creators of Justinian.



Předpis č. 183/2006 Sb.

Zákon o územním plánování a stavebním řádu (stavební zákon) Znění platné od 1. 1. 2014

183

ZÁKON

ze dne 14. března 2006

o územním plánování a stavebním řádu (stavební zákon)

Parlament se usnesl na tomto zákoně České republiky:

ČÁST PRVNÍ
ÚVODNÍ USTANOVENÍ

§ 1 – Předmět úpravy

(1) Tento zákon upravuje ve věcech územního plánování zejména cíle a úkoly územního plánování, soustavu orgánů územního plánování, nástroje územního plánování, vyhodnocování vlivů na udržitelný rozvoj území, rozhodování v území, možnosti sloučení postupů podle tohoto zákona s postupy posuzování vlivů záměrů na životní prostředí, podmínky pro výstavbu, rozvoj území a pro přípravu veřejné infrastruktury, evidenci územně plánovací činnosti a kvalifikační požadavky pro územně plánovací činnost.

(2) Tento zákon upravuje ve věcech stavebního řádu zejména povolování staveb a jejich změn, terénních úprav a zařízení, užívaní a odstraňování staveb, dohled a zvláštní pravomoci stavebních úřadů, postavení a oprávnění autorizovaných inspektorů, soustavu stavebních úřadů, povinnosti a odpovědnost osob při přípravě a provádění staveb.

(3) Tento zákon dále upravuje podmínky pro projektovou činnost a provádění staveb, obecné požadavky na výstavbu, účely vyvlastnění, vstupy na pozemky a do staveb, ochranu veřejných zájmůaněkteré další věci související s předmětem této právní úpravy.

§ 2 – Základní pojmy

(1) V tomto zákoně se rozumí

a) změnou v území změna jeho využití nebo prostorového uspořádání, včetně umisťování staveb a jejich změn,

b) stavebním pozemkem pozemek, jeho část nebo soubor pozemků, vymezený a

Obsah

- ČÁST PRVNÍ - ÚVODNÍ USTANOVENÍ
- ČÁST DRUHÁ - VÝKON VEŘEJNÉ SPRÁVY
- ČÁST TŘETÍ - ÚZEMNÍ PLÁNOVÁNÍ
- ČÁST ČTVRTÁ - STAVEBNÍ ŘÁD
- ČÁST PÁTÁ - SPOLEČNÁ USTANOVENÍ
- ČÁST ŠESTÁ - PŘECHODNÁ A ZÁVĚREČNÁ USTANOVENÍ
- ČÁST SEDMÁ - ÚČINNOST

Historie

č.	Účinnost od	Novela	Porovnat
13	1. 1. 2014	257/2013 Sb.	
12	1. 1. 2013	350/2012 Sb.	
11	1. 7. 2012	142/2012 Sb.	⇄
10	1. 7. 2012	167/2012 Sb.	⇄
9	1. 1. 2012	420/2011 Sb.	
8	1. 1. 2011	281/2009 Sb.	
7	30. 12. 2010	424/2010 Sb.	⇄
6	2. 1. 2010	379/2009 Sb.	
5	1. 1. 2010	345/2009 Sb.	⇄
4	28. 12. 2009	223/2009 Sb.	⇄

Figure 18 Justinian Linked Data browser

[24] Justinian. http://www.justinian.cz

3.4 Summary

Based on the research and examination of existing Linked Data browsers, we can assume that none of them satisfies all the desired features presented using the ideal model solution. However, some projects come close to fulfilling majority of the requirements, most notably the Graphity Client, although it may seem that its primary objective is to create tailored domain specific applications, i.e. to provide a presentation layer for the data.

All of the studied solutions have one thing in common – their target user group are expert users, that is, users who possess extensive knowledge of Linked Data and RDF. One of the challenges that remain unanswered is therefore development of a browser comprehensible enough for common users while providing advanced functionality for the expert ones, especially regarding possible customization of data presentation.

The browser proposed in this thesis must embrace this challenge and special means of user interaction and data display customization must be developed.

4 Analysis

The introductory chapter of this thesis presented the goal of developing a customizable Linked Data browser. The suggested customization options should be implemented in a way that users provide templates to alter the default display of specific Linked Data entities identified by their RDF type.

The previous chapter examining existing Linked Data browsers shed light on possible customization solutions, including analogous template based approaches. The chapter also demonstrated that although some of the existing solutions provide sophisticated methods of altering the data presentation, none of the projects enable their users to do so without extensive knowledge of RDF principles, programming languages and other technologies. While accomplishing the original goal of this thesis, the findings regarding possible customization methods must be taken into account and a technique satisfactory for both expert and non-expert users must be developed.

The model of an ideal Linked Data browser that has been presented and used as a reference browser for the comparison of other existing solutions will be re-used again for the purpose of the analysis, as it is suitable for the most of the features of the reference browser to be included as requirements for the solution being developed as an integral part of this thesis.

4.1 Customization requirements

Given the fact that the customization properties of the browser being developed are to be the distinctive feature in comparison with other existing solutions, it is suitable to discuss this topic separately from other functional and non-functional requirements.

The previous examination of existing solutions and the proposed model of an ideal Linked Data browser clearly show that to create a truly universally customizable generic browser, it is necessary to consider potential users, that is, both experts and non-experts, and provide each target group with set of means to alter the style of presentation of data. The desired goal is to develop a solution that enables straightforward adaptation of data presentation by non-expert users, while avoiding restricting modification options that could be possibly leveraged by expert users.

The objective can be facilitated by creating a two-level customization system. The lower level would enable the expert user to specify how certain data types should be presented to the end user by providing a custom template or styles. The higher level would then enable the non-expert user to rearrange the data and specify the final appearance and layout of the data displayed to the end user. The suggested flow of transformations is depicted on Figure 19. It is necessary to emphasize that the customization process is optional and default behavior must be defined for every step of the process.

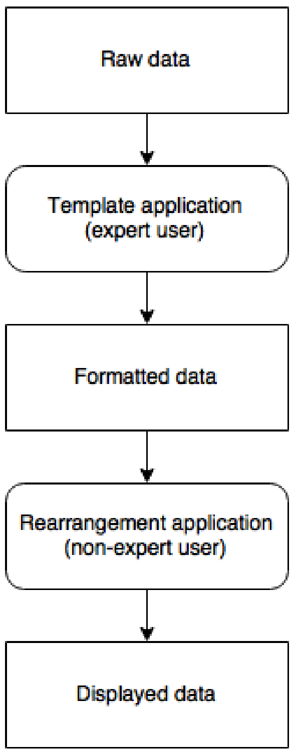

Raw data
Template application (expert user)
Formatted data
Rearrangement application (non-expert user)
Displayed data

Figure 19 Flow of data presentation customization

4.1.1 Expert user customization

On the lower level, expert users are enabled to perform following tasks:

1. Specify of a custom template to display data.
2. Define conditions and priorities for each template, i.e. when to apply the template and how to resolve conflicts, if any.
3. Specify a SPARQL query to fetch additional required data for the custom template, if necessary.

4.1.2 Non-expert user customization

On the higher level, non-expert users are enabled to perform following tasks:

1. Specify a custom *view* to present data, that is, override the default display of data by rearranging information on the page or hiding unwanted data.
2. Define conditions and priorities for each view, i.e. define when to apply the view and how to resolve conflicts, if any.
3. Edit previously created views.

The specification of a view is to be performed using a What You See Is What You Get interface by rearranging data on the page in a drag and drop manner.

4.2 Functional requirements

The fact that the application provides customization options implies that there has to be some form of authentication of authorization present in the application, as it is suitable that the changes in the configuration of the application are performed "on the fly" directly through the application interface, without the need to stop and restart the application.

4.2.1 Authentication and authorization

The application serves three kinds of users – a guest, a member and an administrator. A guest is a user that is not authenticated, a member is an authenticated user and an administrator is a member with administrative rights.

The application provides means to register an account and to log in to an existing account, and optionally log out. The primary identifier of a user is his or hers e-mail address that has to be unique within all user accounts, i.e. it is not allowed to register a user account when another account with the same e-mail address already exists in the system.

Customization of data views as well as global configuration of the application are tasks that affect all end users of the application and are therefore strictly reserved for administrators only.

Searching and browsing is not restricted, thus the end user does not have to be authenticated in order to exploit the functionality.

4.2.2 Administration

The following functionality is reserved for users with administrative rights only.

- User management – assigning and revoking administrative rights to registered users, deleting of users.
- Endpoint management – specification of available SPARQL endpoints.
- Language management – specification of available language parameters.
- View management – creating, editing and deleting customized data views.

4.2.3 Browsing data

When browsing data, the application displays information about a single resource at once, the resource is identified by its URI. The following information is presented to the user:

- The URI of the currently displayed resource.
- RDF types of the resource.
- All the properties that exist as predicates in relations of the resource, for the resource being both the subject and object of those relations.[25]
- For each property a list of its values is presented.

The application must manage large amounts of data correctly and by default display only a limited number of values per one property. Upon request, the application loads more results and provides a convenient way to paginate through the results, displaying only the selected subset of results.

Should there exist a user defined customized view applicable to the currently displayed resource, the data must be presented according to the definition contained in the view.

In addition, the application provides a *raw* mode, inhibiting all the configured customizations, enabling the user to see the data exactly in the way they are stored in the database. Furthermore, the *raw* mode displays all the graphs associated with the current resource[26] and enables the user to restrict the displayed result by selecting one or more graphs, i.e. display only properties contained in those graphs.

[25] I.e. (*resource property otherResource*), (*otherResource property resource*)

[26] All the graphs containing one or more triples where the currently displayed resource figures as a subject or an object.

To summarize – the data displayed in the standard browse mode can be altered by creating a customized view, whereas in the *raw* more it is possible to modify the result by restricting graphs the resource is contained in.

4.2.4 Searching data

A search function serves as a primary means of exploring data. An input field is provided to the user to insert a search query, which is then executed upon request. The application performs a fulltext search in any strings present in the database. Every string found in such a way is by definition an *object* in some (*subject predicate object*) relation, and in that case the *subject* is in fact a URI of a resource the user looks for and should be therefore included in the results of the query. On the top of that, the application provides following search options to refine the search results:

- Option to specify which RDF data types to search in.
- Option to specify which data properties to search in.
- Option to specify which graph or dataset to search in.

The options can be used separately or all at once depending on the user's preference. In addition, the user may specify multiple options of each type.

The search results are presented as a table, including links to found data entities. Upon clicking the link, the *browse* mode is initiated and relevant information about the target resource is displayed.

4.2.5 Data sources

The key component of the browser are the data sources it operates with. When defining the model of an ideal Linked Data browser in the previous chapter, it was determined that the browser should support retrieving data from SPARQL query services, as any other forms of data sources can be conveniently mirrored to a database and supported with a SPARQL endpoint.

It is therefore both necessary and satisfactory for the suggested browser to provide access to data through SPARQL endpoints.

An array of SPARQL endpoints can be configured by users with administrative rights, as well as the default endpoint which is used if no end user preference is set.

The application supports any generic SPARQL endpoint that is compliant with the SPARQL 1.1 Protocol.[27]

A user can select one or more pre-configured SPARQL query services as the data source. This settings applies for both searching and browsing and is persistent throughout the user session. However, the settings must be dependent on the application instance, i.e. it should be possible to have two instances of the application open side by side, each one with different data source settings.

Introduction of multiple SPARQL endpoints brings new problems that will have to be solved in the design phase. The desired behavior is such that the application operates on all the selected endpoints at once, however this goal might be hard to achieve, especially regarding possible performance problems. Therefore it will be satisfactory if the application provided access to only one endpoint at once. However, should such circumstances appear, it is necessary to ensure that the user can handle selection and changes of endpoints with ease, without the loss of current application context, i.e. when browsing a particular resource, the change of an endpoint should not change the state of the application, the same context and same resource should be displayed, except for the data, which will be provided by the newly selected endpoint.

4.3 Non-functional requirements

Akin to the model Linked Data browser presented in the previous chapter, the proposed browser is a web application embracing responsive design principles. On top of that, it implements a Single-page application (SPA) design approach. The goal is to provide fluid user experience comparable to a desktop application. All the necessary resources should be therefore loaded at once when the application is initially loaded in the browser, that is, in the moment when the user first accesses the application. After that, only necessary data should be transmitted to and from the server, as the following communication should preferably consist only of API requests, eliminating the need of ever reloading the page.

Another major requirement – and probably the most important one – is the necessity of good performance. When browsing the selected data sources, the average processing time of API data requests should be no longer than five seconds, even when

[27] SPARQL 1.1 Protocol. http://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/

displaying resources that are parts of hundreds of thousands of relations. All SPARQL queries should be optimized and only necessary information requested. However, the performance of the queries is strictly dependent on the performance of the SPARQL endpoint and cannot therefore be directly controlled. Still, the application must be optimized especially regarding the customization features, that is, if there are custom templates defined for a particular resource or a custom view, or a combination of both, the resulting performance should not be negatively affected in an intolerable way.

Nonetheless, sometimes it is impossible to avoid complex SPARQL queries that require a lot of computation, especially when looking for entities using fulltext search. In some cases, it is possible for the request to take significantly long time, therefore the execution time of the queries must be limited. If the maximum execution time is reached, the query is aborted and the user must be notified. Hopefully, the user will be able to issue a different, more restricting query that would still satisfy his concerns.

The web application should work in all contemporary major web browsers, most importantly the ones using WebKit[28] as a core rendering component of web pages. This engine (or its variants) is used by Google Chrome, Opera or Safari browsers.

[28] The WebKit Open Source Project. https://www.webkit.org/

5 Design and Implementation

Based on the requirements that arise from the Analysis, this chapter suggests a design of a customizable Linked Data browser and describes key design and implementation features and decisions.

Given the fact that the browser is a web based application, it is essential to examine possibilities of deployment and employed technologies as the first order of business, for those will likely determine or at least influence architecture of the system as well as many further major and minor design decisions.

5.1 Deployment and employed technologies

When building web applications, there are two major deployment variants that are broadly used across the Internet. The traditional and probably the most common one is the LAMP stack, which stands for Linux OS, Apache Server, MySQL database and PHP (or sometimes Perl or Python). In the recent years, a modern challenger of this approach emerged the MEAN stack – MongoDB, Express, AngularJS, Node.js. The MEAN stack represents a thoroughly modern approach to web development: one in which a single language (JavaScript) runs on every tier of the application, from client to server to persistence. [17]

There is no need to describe the individual parts of the LAMP stack further, as they are commonly well known. However, it might be useful to briefly describe the components of the MEAN stack.

MongoDB[29] is a cross-platform document-oriented database. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas.

Express[30] is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. The framework is easily extensible through various middleware.

[29] MongoDB. http://www.mongodb.org
[30] Express. http://expressjs.com

*Angular*JS[31] is a frontend web application framework maintained by Google designed for creating dynamic single page applications. Using the framework, it is possible to implement client side model–view–controller (MVC) architecture.

Finally, *Node.js*[32] is an open source, cross-platform JavaScript runtime environment for server-side and networking applications. The platform uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications. [18]

Because of the fact that one of the requirements is for the browser to implement Single-page application design, the MEAN stack is a logical choice, as it is a priori designed to provide such a functionality, especially thanks to the AngularJS component. In addition, these technologies can prove useful in many areas of the project. Here is a list of major reasons in favor of utilization the MEAN stack:

- Parts of the stack are designed to work together.
- Single programming language – JavaScript – on both client and server.
- Platform independent solution thanks to Node.js.

Furthermore, the author has already significant experience with the stack as it has been used to develop the aforementioned Justinian project and it has proven effective and suitable to be used in the Linked Data environment.

For the purpose of the Linked Data browser, we are not going to employ the MongoDB, as the expected size of the application database is likely to be small and the data can be therefore held in-memory and eventually persisted in the file system. The in-memory database is discussed further in following sections of the thesis.

5.2 Architecture and data flow overview

The architecture is largely determined by the analysis and by the employed MEAN stack. Figure 20 depicts an overview of the system architecture, together with key system components and data flow between those. Majority of requests is realized using the REST approach. JSON was selected as a transmission format of data between components, its JSON-LD variant is used when sending RDF data.

[31] AngularJS. https://angularjs.org
[32] Node.js. http://nodejs.org

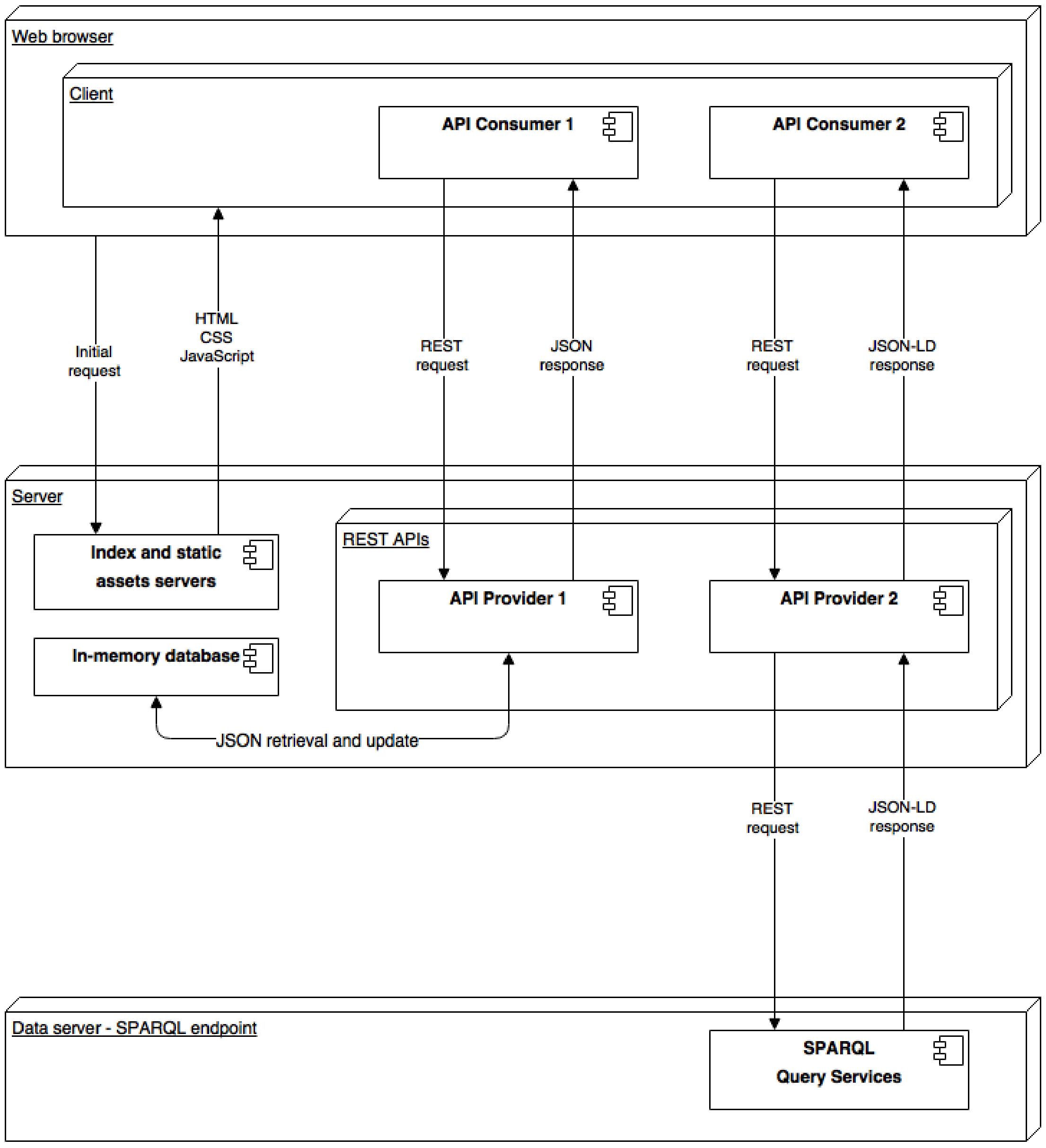


Figure 20 Architecture and data flow overview

The server side provides two kinds of API providers. The first kind is aimed to facilitate application configuration and communicates with a local in-memory database only. These APIs are designed to provide CRUD interface for application dynamic data, i.e. users, endpoints or customized views, so that these resources can be stored on the server once created on the client side.

The second kind of APIs provide access to remote SPARQL endpoint, the aim of those is to facilitate data retrieval by issuing a SPARQL query that is defined and constructed server-side. Data is transmitted in JSON-LD format.

5.3 Resource description

An integral part of the developed Linked Data browser is resource description, which is a process of obtaining information about the resource that describes the resource thoroughly. Specifically, it is a SPARQL query designed in such a way that it requests a portion of a graph that contains the resource and its *surroundings*. The problem is that the content of the surroundings is unknown at the time of issuing the query.

This section discusses possible solutions of constructing such a subgraph. The most important aspect of designing such a query is that the fetched subgraph should contain enough information to be displayed to the end user, that is, the ideal solution includes sufficient amount of information so that no additional queries regarding the described resource are needed. At the same time, the query must be able to be processed fast enough and return only the minimum amount of data truly needed. This is the key idea of this section of the thesis and will be referred to repeatedly.

According to the analysis (section 4.2.3 Browsing data), the requested subgraph should contain at least the following information:

- Resource URI
- RDF types of the resource
- All the properties that exist as predicates in relations of the resource, for the resource being both the subject and object of those relations, and their values.

SPARQL language contains a special kind of query designed especially for such cases – the DESCRIBE query, introduced in the Preliminaries chapter. Even though it is a great candidate to use for resource description, it is necessary to find an alternate solution, as the DESCRIBE query has two drawbacks:

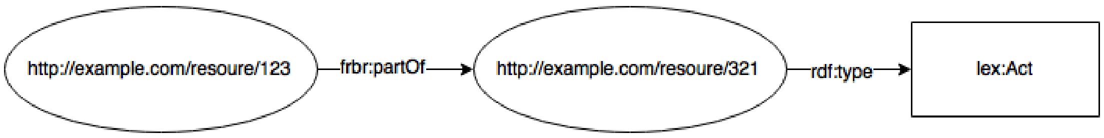
1. The result of the query is dependent on the SPARQL query service and its configuration, that is, different SPARQL endpoints may provide different results, even though when operating on the same database.
2. The query cannot be limited, i.e. if a resource has thousands of relations of a particular RDF type, it always returns all of them, which can cause performance issues.

3. The query does not return enough information to really describe the resource, as it only examines the adjacent nodes, but not other levels of the graph.

```
DESCRIBE <http://example.com/resource/12345>
```

Figure 21 SPARQL DESCRIBE query

What we need is to invent a query that provides similar results, but in addition it enriches the resulting graph with some essential information about objects on the second level – an example is demonstrated by Figure 22. To clarify, it is suitable to fetch additional information for each resource that is adjacent to the described resource, that is, every resource that is contained in a triple with the described resource either as an object or a subject.



http://example.com/resoure/123
frbr:partOf
http://example.com/resoure/321
rdf:type
lex:Act

Figure 22 Example of a second level relation of a resource

In accordance with the key idea of the minimal sufficient subgraph presented previously, the amount of information to be fetched about adjacent resources has to be limited, namely, it seems satisfactory that only a RDF type and a title of the resource are retrieved. As for the title, by default the application considers following RDF properties as titles: *foaf:name*, *dcterms:title* and *skos:prefLabel*. This default configuration can be overridden (more on that topic later).

In addition to the titles of adjacent nodes, it is desirable to fetch labels of the particular properties involved.

Furthermore, the result of the query must be a valid RDF representation so that it can be serialized to JSON-LD format, therefore it is necessary to use the CONSTRUCT SPARQL query.

Figure 23 shows a possible SPARQL query that improves on the DESCRIBE query and that satisfies presented conditions.

```
PREFIX my: <http://my/>
PREFIX resource: <http://example.com/resource/123>

CONSTRUCT {
    resource: a ?resourceType .
    resource: my:out my:outcontainer .
    my:outcontainer ?outPredicate ?outObject .
    resource: my:labels ?outPredicate .
    ?outPredicate my:label ?outPredicateLabel .
    ?outObject a ?outObjectType ;
            ?outSecondaryPredicate ?outSecondaryObject .

    resource: my:in my:incontainer .
    my:incontainer ?inPredicate ?inObject .
    resource: my:labels ?inPredicate .
    ?inPredicate my:label ?inPredicateLabel .
    ?inObject a ?inObjectType ;
            ?inSecondaryPredicate ?inSecondaryObject .

}

WHERE {
    OPTIONAL {
        resource: ?outPredicate ?outObject .
        MINUS { resource: a ?outObject } .
        OPTIONAL { resource: a ?resourceType }
        OPTIONAL { ?outObject a ?outObjectType } .
        OPTIONAL { ?outPredicate rdfs:label ?outPredicateLabel }
.
        VALUES ?outSecondaryPredicate { foaf:name dcterms:title
skos:prefLabel }
        OPTIONAL { ?outObject ?outSecondaryPredicate
?outSecondaryObject } .
    } .
    OPTIONAL {
        ?inObject ?inPredicate resource: .
        OPTIONAL { ?inObject a ?inObjectType } .
        OPTIONAL { ?inPredicate rdfs:label ?inPredicateLabel } .
        VALUES ?inSecondaryPredicate { foaf:name dcterms:title
skos:prefLabel }
        OPTIONAL { ?inObject ?inSecondaryPredicate
?inSecondaryObject } .
    }
}
```

Figure 23 Advanced resource describe query

Although the query presented in Figure 23 is quite complex, it surprisingly works alright. Figure 24 displays a partial result of this query. The original subgraph is modified in a way that the described resource contains only three properties – *rdf:type*, *my:labels*, *my:out* and *my:in*. The three *my:* prefixed properties are in fact virtual containers for labels and subject and object relations.

```
{
    "@id":" http://example.com/resource/123",
    "@type": [ ... ],
    "http://my/labels": [ ... ],
    "http://my/out": [ ... ],
    "http://my/in": [ ... ]
}
```

Figure 24 Partial result of advanced resource describe query

Even though the query suggested in Figure 23 describes well resources engaged in a small number of relations, it is unfitting for describing *larger* data entities. Following problems still have to be solved:

1. Query returns all available relations instead of a sample of relations for each property.
2. Query is too complex and therefore slow to execute.

Since it is impossible to simplify the query without any loss of information, it is necessary to employ a different strategy – divide the query to small subqueries and execute them in parallel whenever possible. A schema of this strategy is depicted in Figure 25.

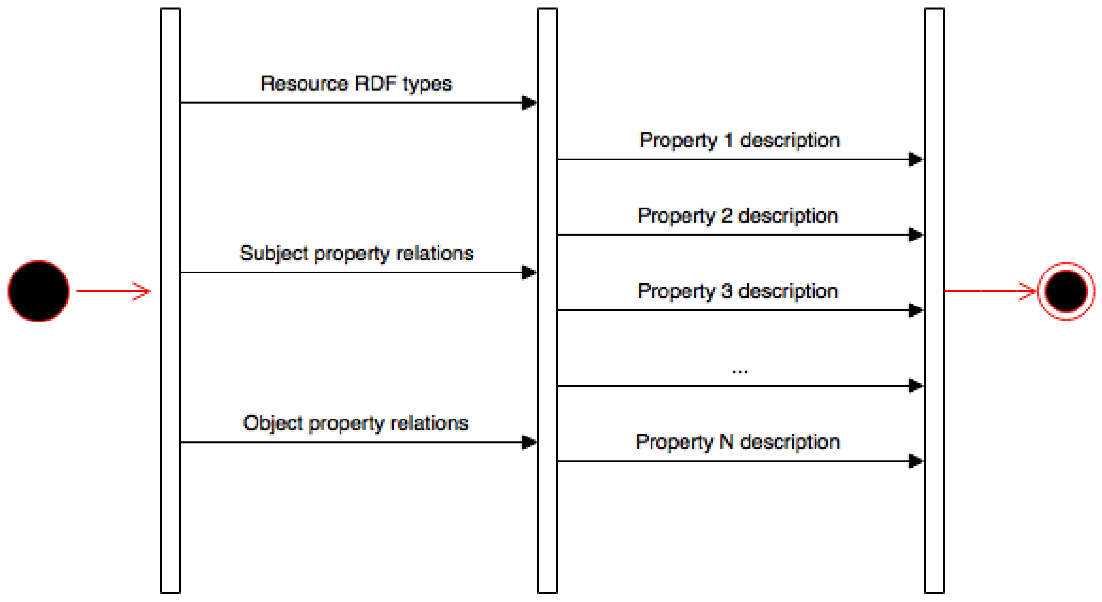


Figure 25 Parallel resource description

The strategy is realized in two steps. Firstly, only basic information about the object is fetched – its RDF type and properties (types of its relations), that is, for every triple (*resource predicate other*) and (*other predicate resource*) a distinct list of predicates is fetched. We are going to refer to these relations as *subject relations* and *object relations* depending on whether the described resource figures in the relation as a subject or an object. The first step can be executed by three SPARQL queries performed all at once.[33]

[33] The process could be actually realized using only two SPARQL requests by including the RDF types as the subject property relations list. However, excluding the type to a separate query may speed further processing. Since the queries are performed in parallel, it should not affect performance in a negative way.

```
PREFIX my: <http://my/>
PREFIX resource: <http://example.com/resource/123>

CONSTRUCT {
    resource: my:subject ?property
}

WHERE {
    resource: ?property [] .
    FILTER(?property NOT IN (rdf:type))
}
```

Figure 26 A query to fetch subject relations of a resource

Once the three preliminary queries are resolved, the algorithm shifts to the second phase. For each distinct predicate found in the first step, a separate query is issued to gain information about the particular property. This tactics has the following advantages:

1. Only a limited number of values of each property may be requested.
2. A total count of the values of each property can be calculated.
3. A possibility of a parallel execution and therefore improved performance.

An example of a query that asks for information about a particular property of a resource is presented in Figure 27.

```
PREFIX my: <http://my/>
PREFIX resource: <http://example.com/resource/123>
PREFIX property: <http://example.com/properties/someProperty>
CONSTRUCT {
    my:property my:id property: .
    my:property my:relation my:subject .
    my:property my:count ?count .
    my:property my:label ?label .
    my:property my:data ?object .
    ?object ?property ?value .
}

WHERE {
    {
        SELECT ?object
        WHERE { resource: property: ?object . }
        LIMIT 5
    }
    {
        SELECT (COUNT(DISTINCT ?dist) as ?count)
        WHERE { resource: property: ?dist . }
    }
    OPTIONAL {
        VALUES ?property { rdf:type foaf:name dcterms:title
                                                skos:prefLabel } .
        ?object ?property ?value .
    }
    OPTIONAL {
        property: rdfs:label ?label .
    }
}
```

Figure 27 A query to request a subject property relation information

It is necessary to emphasize that employment of this strategy can lead to a significant number of SPARQL queries when describing a single resource. The total number of requests is:

$$Total = 3 + \#subjectProperties + \#objectProperties$$

That is, three preliminary queries and one query for each type of subject or object relations.

After all the data are fetched, they need to be merged programatically into one aggregate graph. This merging should not cause any difficulties performance-wise, as the time needed for code execution and in-memory manipulation is generally negligable compared to the SPARQL queries.

On the other hand, it is a well-known fact that performing a large number of HTTP queries may cause performance issues, it is therefore an absolute necessity to test the proposed solution on both *small* and *big* data entities, denominated by the number of relations they are part of.

Preliminary empiric testing has shown that this strategy performs exceptionally well when fetching *big* entities, and on top of that it is comparable to the naïve approach presented earlier in this chapter that requested all the data at once. However, the performance of the design must be further thoroughly examined in the evaluation phase of the project.

To summarize, the resulting graph includes following properties of a described resource:

1. Resource URI.
2. Resource RDF type
3. List of resource's properties, consisting of both subject and object relations.

In addition, every property item consists of following information:

1. Property URI.
2. Property RDF label, if present.
3. Type of the relation - *subject* or *object*.
4. A collection of five sample *values.*
5. Total count of *values.*

Should a value be in fact a resource, additional information is provided:

1. Value resource URI.
2. Value resource RDF type.

3. Value resource title, if present, by default consisting of aforementioned properties *foaf:name, dcterms:title* and *skos:prefLabel*.

The resulting graph describing a particular resource will be from now onwards referred to as a *resource graph*.

5.4 Data presentation and customization

This section describes a process and methods involved in delivering Linked Data to the end user, that is, transforming a *resource graph* presented in previous section in such a way that is can be displayed to the user.

The analysis defined that the application must provide means to customize the data presentation on two levels – firstly, a custom templates may be provided to alter the default style of particular data entities identified by their RDF type, secondly, a user must be enabled to modify the final layout of the data, i.e. to rearrange the data to his or hers liking.

In accordance with the analysis, the following paragraphs present a solution that satisfies the requirements and further extends customization capabilities of the data browser.

5.4.1 Miniapplications

Let us consider the customization of data presentation on the higher level, that is, modifying a layout of the web page. Should such a rearrangement be possible, the data must be divided into autonomous blocks that could then be somehow organized. To comply with the analysis, each property of the *resource graph* should be embodied in such a separate block, so that the user can decide which properties to include in the view (and in what order) and which to hide. Since the blocks ought to be independent and in some cases might need to perform non trivial tasks like operate with the data model etc., they cannot be considered as just ordinary templates, but instead should be considered as *miniapplications* – templates enriched with a controller and other non-static capabilities.

Furthermore, it is not necessary to restrict such miniapplications to displaying just one property. Besides utilizing multiple properties, a miniapplication might not work with any of the resource's properties at all, and, for example, display some general or even unrelated information. The key idea is to permit as much customization as possible. From this point of view, the resulting data presentation will be actually a collection of miniapplications that the user would be able to rearrange to his taste.

In addition, it is not necessary to restrict the number of the same miniapplications on the page, as both of them should be enabled to serve for example as display providers for different resource's properties. Each displayed miniapplication should be therefore identified by its *name* as well as its *instance*. An obvious use case is the existence of a *default* miniapplication capable of presenting any data, every instance displaying distinct property of the resource.

To satisfy the requirements set by the analysis, each miniapplication must provide a server-side configuration script with following functions or properties included:

1. function *matchInstances*(*resourceGraph*) : *Array*.
2. function *inhibitInstances*(*resourceGraph*) : *Array*.
3. function *setupApplication*(*app, authorization*) : *Void*.
4. *displayTemplate : File*.
5. *displayPriority : Integer*.
6. *setupPriority : Integer*.

The reason for the miniapplication configuration being resolved on the server side is especially the third function – *setupApplication*, that permits the miniapplication to define custom API providers.

Function *matchInstances* returns an array of miniapplication's *instances* that can be deployed using the particular resource graph passed as an argument. An *instance* is an *object* that serves a special purpose – in addition to identifying the miniapplication, it is used to bootstrap the miniapplication on the client side. Therefore, miniapplications must include sufficient information in the configuration script so that they can be properly invoked on the client side. There is no explicit restriction on the contents of the *instance* object, but the following rule is proposed: should the miniapplication cover a single property, the instance object ought to include the property's URI and relation (*object* or *subject*).

A custom miniapplication that displays information about a particular property might want to prevent other miniapplications covering the same property from being shown to the user. In such case the miniapplication should define *inihibitInstances* function that returns *names* and *instances* of other miniapplications that are to be inhibited. This feature is designed so that it provides the required functionality of a custom template definition for particular types of data.

The *displayTemplate* property specifies the miniapplication's default template file to be loaded on the client side when the miniapplication is initialized.

The *displayPriority* property defines a default sort order priority if no customized view is defined by user. The higher the number, the higher the miniapplication is displayed on the web page.

The *setupPriority* property defines a setup order priority which is related to the inhibition function. A miniapplication can inhibit other miniapplications and their instances only if it has a higher setup priority defined. The setup priority is introduced in order to prevent circular inhibition – the case when two miniapplications try to inhibit each other.

One of the requirements of the analysis is to provide two distinct modes for browsing the data – a standard mode and the *raw* mode that would display the data in the way they are stored in the database. To accomplish this functionality, a special class of *raw* miniapplications must be set apart. These miniapplications should provide a *raw* Boolean indicator in their configuration file so that they can be distinguished from the standard miniapplications. In addition, even the *raw* browse mode should enable the user to define special *raw* miniapplications and possibly override the default ones.

5.4.2 Layouts

In order to enable the user to rearrange miniapplications on the web page, it is necessary to specify boundaries or containers that would permit such an arrangement. This functionality can be realized by introducing *layouts* that are in fact special types of miniapplications. The core of the layout is a HTML template with annotated panels – identified by their *panel name* – that are to be populated with miniapplications. The population and user interaction has to be handled by the application, not by the layout itself, so that the layout template can be as simple as possible.

As opposed to standard miniapplications, each layout is identified only by its *name*. In addition, it must provide a server-side configuration script with following information:

1. *panels* : *Array of Strings.*
2. *defaultPanel* : *String.*
3. *displayTemplate* : String.

The *panels* array contains a list of all the *panel names* contained in the layout template.

The *defaultPanel* denotes a panel that is populated by default, that is, if no customized view exists.

The *displayTemplate* property specifies the layout's default template file to be loaded on the client side when the layout is initialized.

As well as default miniapplications to present the data, the browser provides a *default* single-column layout.

5.4.3 Views

Layouts and miniapplications are organized into *views*. There are three types of views defined, each serving a different purpose – a *default view*, a *raw view* and a *custom view*.

A *default view* of a *resource graph* is a configuration of a default layout and a collection of resolved miniapplications that can be displayed for the resource graph, that is, miniapplications that provide a list of instances minus miniapplications that are inhibited. The display order of the miniapplications corresponds to the *displayPriority* parameter.

A *raw view* is constructed in the same manner as a *default view*, except for the fact that only miniapplications with *raw* parameter are included.

A *custom view* is a view defined by a user that can be applied to one or more resources. Each custom view consists of the following properties:

1. A reference resource

2. A selected layout.
3. An ordered list of miniapplications for each panel that is defined by the selected layout.
4. Matching rules – a set of rules that must be all satisfied in order to apply the layout
5. Priority – in case that two views possibly match for one particular resource.
6. Strict mode – defines how to manage miniapplications that are not part of the view definition.

A *custom view* is created using a *reference resource* – which is a particular resource graph. The structure of the resource graph determines the set of available matching rules. To clarify, let us consider the following example: a user is browsing the data and stumbles upon a resource he would like to customize. The application enables him to create a custom view for the resource – to select a particular layout and reorder the displayed miniapplications. The user can then define a class of resources the created view can be applied to using matching rules, for example all resources with the same RDF type as the resource used to create the view – hence the reference resource.

The matching rules are in fact a *subgraph* of the resource graph. The custom view can be applied to any other resource graph that includes the subgraph. These matching rules are generally available:

1. Matching based on the reference resource URI (if this rule is selected, only the reference resource will ever satisfy the subgraph inclusion)
2. Matching based on the resource RDF type
3. Matching based on resource properties – whether or not such properties exists
4. Matching based on resource properties' types – whether a property exists that contains an object of a particular RDF type

The property matching rules are available for both subject and object property relations. It is necessary to emphasize that all the selected rules must be satisfied at the same time in order to apply the custom view on the desired resource.

If there is a conflict of views, i.e. if two custom view definitions are both satisfied for a particular resource, the view with higher *priority* is selected.

Finally, *Strict mode* property of the custom view specifies the strategy to employ when dealing with miniapplications not covered by the custom view definition. It is possible and probable that the structure of some resources from the class of target resources specified by matching rules will differ from the structure of the reference resource, and it is therefore possible that there may be matching miniapplications that can be displayed in a *default view* of the particular resource, but are not included in the *custom view* definition, because they do not provide any matching instances for the reference resource. If the strict mode is selected, those additional applications are omitted, otherwise they are included in the *default panel* of the layout.

5.5 Authentication and authorization

Authentication and authorization is implemented using JSON web tokens.[34]

The process of the token negotiation is depicted on Figure 28. When the user logs in, the authentication is performed on the server. If the authentication succeeds, the server sends a signed token to the client and the client saves the token. With following requests, the client includes the token as a part of the HTTP request and the token is verified on the server.

It is possible to include the user's identity in the token, thus making it a tool suitable for user authorization, as one can provide information about user's roles or rights directly into the token as a part of the user's identity.

This authentication and authorization approach is suitable for single page applications, because the server does not have to maintain a user session, as all the information necessary for the authentication and verification is encoded in the token.

[34] "JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted." [19]

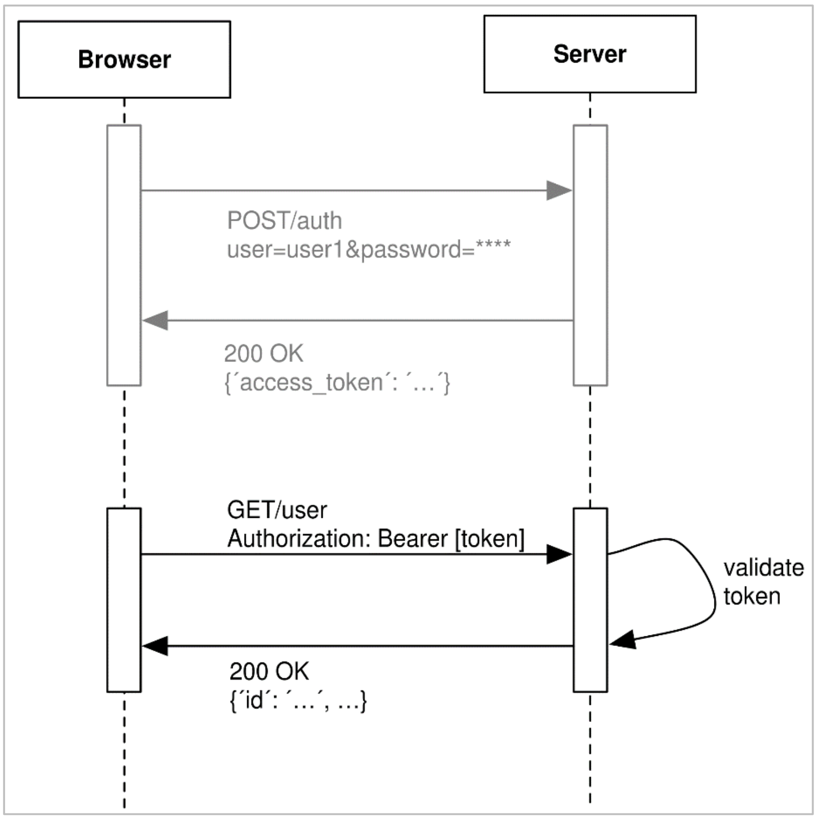


Figure 28 JSON web token negotiation

5.6 Key server-side components

The server-side part of the application is built using the previously introduced Express web framework. The framework works as a standalone server managing HTTP request and responses as well as routing.

The server-side is organized in a Model-View-Controller manner. A set of possible URL routes is defined, each of them corresponding to a different controller or a method of a controller. Controller can utilize models, key system components and provide the content for HTTP responses. Regarding the View part in the MVC approach – according to the fact that the application follows a Single-page application design and thus the majority of its URL routes are REST API resources, the controllers generally provide a JSON or JSON-LD output. The only exception is *index controller* which provides the initial HTML code needed to bootstrap the application on the client side.

The server-side includes following key components.

5.6.1 Index router and controller

Index router and controller component is responsible for providing initial HTML code to bootstrap the application on the client side. The resulting HTML includes links to all JavaScript and CSS files present in the application, including files contained in custom layouts and miniapplications. Those files are provided by the Assets manager component.

5.6.2 AssetsManager

AssetsManager's primary objective is to maintain a list of application's assets that need to be loaded in the client bootstrap phase. Upon initiation of the server-side of the application, the manager scans all relevant directories for JavaScript and CSS assets. The list of assets can be then accessed from other components, for example by the index controller.

5.6.3 SparqlQueryBroker

SparqlQueryBroker manages communication with SPARQL endpoints, facilitates sending query requests and manages endpoint responses.

The component is designed to work closely with two other key components – *SparqlQueryRenderer* and *SparqlQueryAdapter*.

The workflow of the broker is as follows: firstly, the *SparqlQueryRenderer* component is invoked to provide a generated SPARQL query. Secondly, the SPARQL endpoint is contacted and the component waits for response. Finally, upon receiving the response, the broker invokes the *SparqlQueryAdapter* tool chain to further modify the response and serves the result to the client-side.

5.6.4 SparqlQueryRenderer

SparqlQueryRenderer is a component that facilitates parametrized SPARQL queries. The components receives a SPARQL query template and a list of parameters; then, it processes the template and looks for parameter annotations. Finally, it replaces the annotations found in the template with the actual values of the input parameters. An example of a SPARQL query template is presented in Figure 29. The example query contains a placeholder for a *resource* parameter.

```
CONSTRUCT {
    {{ resource }} a ?type .
}
WHERE {
    {{ resource }} a ?type .
}
```

Figure 29 SPARQL query template example

5.6.5 SparqlQueryAdapter

SparqlQueryAdapter is a tool chain designed to modify SPARQL endpoint responses before re-sending them to the client. The component is in fact a set of consecutive task, where each task can alter the response in a different manner. Following tools are included:

- *JSON-LD context application* – the response can be simplified and compacted by providing a JSON-LD data context.[35]
- *Complex object reconstruction* – the RDF data contained in the JSON-LD response are a graph, and as such may and generally are scattered across an array of separate objects, each object corresponding to a particular Linked Data resource; thus, to facilitate the use of the data on the client side, this tool merges all the response objects into one resulting object, creating a tree-like structure. The merging is based on the resolution of the resources URIs.
- *Properties prefixes replacement* – this tool explores the result graph and considers URIs of any properties it finds – if a prefix is defined for each such URI, the URI is replaced with the corresponding contracted variants. Prefixes are provided by the *PrefixManager* component.
- *Model definition* – in addition to previous modification, a *model* of the response may be defined. A model is a description or a schema of the response. This tool ensures that the resulting response corresponds to the defined model.

[35] The following document provide additional details: http://www.w3.org/TR/json-ld/#the-context; furthermore, examples of application of context to real RDF data can be found at: http://json-ld.org/.

5.6.6 PrefixesManager

PrefixesManager is a component that facilitates usage of *namespace prefixes* of URIs across the application.

Upon creation of a SPARQL endpoint definition, the component contacts the endpoint and requests a list of predefined namespace prefixes. The requested prefixes definition is then stored in-memory and can be even propagated to the client side.

To summarize, the component provides implicit application-wide automatic services regarding prefixes definition and management. Therefore, there is no need for the namespace prefixes to be explicitly defined in the application configuration.

5.6.7 ResourceGraphBuilder

ResourceGraphBuilder implements thoroughly the process of creating a *resource graph* described previously in this chapter (section 5.3 Resource description).

The component takes a resource URI as an input parameter and assembles the *resource graph* according to the design of the process.

5.6.8 ViewBuilder

ViewBuilder implements the process of generating *default*, *raw* and *custom views* described previously (section 5.4.3 Views).

The component takes a *resource graph* as an input parameter and constructs a desired view corresponding to the *resource graph* according to the design of the process already described.

5.7 Key client-side components

The client-side part of the application is built using the previously introduced AngularJS Single-page application web framework. Part of the framework works as a frontend routed.

Similarly to the server-side, the client-side is also organized in a Model-View-Controller manner. A set of possible URL routes is defined, each of them corresponding to a particular application *state*, which then determines the target HTML template and controller. The models are provided by services that request the desired data from corresponding REST API providers located on the server-side.

The client-side includes three kinds of components – services, directives and filters. Each of the following components can be utilized by custom miniapplications or layouts.

5.7.1 Services

A service is a singleton instance of an object that can be requested in a controller. The application includes the following key services.

- *Config* – provides a model of application configuration, including existing endpoints, languages, layouts and miniapplications.
- *Describe* – provides access to server-side API for describing resources, that is, given a particular URI, the service returns the *resource's graph* along with the requested *view* containing the information of the data presentation layout.
- *Identity* – provides the current user's identity, most importantly his e-mail address and role if the user is authenticated.
- *Miniapp* – provides functions to facilitate miniapplication bootstrapping and management of custom server-side API requests.
- *PrefixesReplacer* – provides functions to contract or expand predefined namespace prefixes in URIs.

5.7.2 Custom directives

A directive is a method of extending AngularJS HTML templates to provide additional functionality. To facilitate custom miniapplication and layout development, following directives are implemented:

- *loading-bar* – displays a block with a loading indicator.
- *datapager* – a component to facilitate pagination of large arrays of data. The displayed data can be fetched asynchronously upon request.
- *describe* – a directive to facilitate generation of hyperlinks for Linked Data browsing – an input resource URI is converted to a functioning link.
- *describe-list* – the same as describe, except it expects the input to be a list of resource URIs, which are then transformed to a horizontal list of hyperlinks. A filter can be defined to be applied to each item in the input array (see 5.7.3 Custom filters for details).
- *list* – converts an input array of values to a vertical list.

- *list-inline* – the same as list, except the resulting list is horizontal.
- *print-values* – the same as list, except a value filter is applied to each item (see 5.7.3 Custom filters for details).
- *layout-panel* – a directive to be used strictly by layouts *display* templates to annotate its defined panels, that is, containers to be populated with miniapplications.
- *layout-panel-setup* – the same as *layout-panel*, except designed for usage in layouts *setup* templates; for example, the developer may choose to provide a simplified version of layout to be used when creating a *custom view*.

5.7.3 Custom filters

A filter is a function that extends AngularJS data output formatting. Generally, a filter takes an input value and transforms it according to its definition. In addition, filters can be chain in sequences. To facilitate custom miniapplication and layout development, following filters are implemented:

- *truncate-uri* – takes an input URI and shortens it, keeping the domain and the ending of the URI intact.
- *truncate-uri-large* – similar to *truncate-uri*, but produces longer strings.
- *id* – takes a JSON-LD object as an input and extracts its URI from the *@id*[36] field; should an array of objects be provided as the input, an array of URIs will be returned.
- *value* – similar to *id*, but extracts information from the *@value*[37] field; furthermore, if the input object contains more than one value, the filter returns either the value corresponding to a preferred defined language or the first one in the array.
- *label* – takes a JSON-LD object and extracts a possible label of the object to be displayed to the user. The following RDF properties are considered labels: *rdfs:label, foaf:name, dcterms:title, skos:prefLabel*. In none of these properties are present in the object, the resource URI is returned instead.
- *contract* – shortens an input URI by replacing its predefined namespace prefix.
- *expand* – the opposite of *contract*, produces full URIs.

[36] See JSON-LD reference for details. http://www.w3.org/TR/json-ld/
[37] Ibid.

5.8 Supported SPARQL endpoints

The analysis determined that any SPARQL query service should be supported by the application. Further research showed that this desired goal is currently not possible to achieve without the sacrifice of significant functionality of the application.

Namely, there are two features that are – for the time being – not standardized by SPARQL 1.1 definition and therefore not broadly supported.

1. SPARQL query including fulltext search.
2. Predefined namespace prefixes definition resolution.

Currently, the only known solution the author identified that provides those features is OpenLink Virtuoso Universal Server.[38] This solution satisfies the requirements in the following ways:

1. The server implements a built-in `bif:contains` SPARQL function for effective fulltext search.
2. Predefined namespace prefixes can be obtained directly from the SPARQL endpoint by a GET request including a `?nsdecl` HTTP query parameter.[39] Unfortunately, the endpoint service provides the list of prefixes in a form of HTML table, thus non-trivial data processing is needed to extract the required information.

The design of the application is therefore restricted by the findings and for the time being the application only supports the aforementioned OpenLink Virtuoso solution. Extending the support for other SPARQL query services will therefore have to be an objective for future work.

5.9 Project directory structure

In this section, the file and directory structure of the application is presented.

5.9.1 Structure overview

- `%ROOT_PATH%` - the root of the project
    - `/config` – application configuration

[38] OpenLink Virtuoso Universal Server. http://virtuoso.openlinksw.com/
[39] List of Opendata.cz predefined namespaces: http://linked.opendata.cz/sparql?nsdecl

    - `/datastore` – persistent storage of in-memory database
    - `/layouts` – directory of defined layouts
    - `/miniapps` – directory of defined miniapplications
    - `/public` – the root of the client-side part of the application
    - `/server` – the root of the server-side part of the application
    - `/bootstrap.js` – the initialization script to run the application

5.9.2 Suggested miniapplication structure

- `%MINIAPP_ID%` – the containing directory serves as a unique identifier
    - `/public`
        - `/assets` – optional assets to be included on the client-side
        - `/controllers` – miniapplication controllers directory
        - `/views` – client templates
            - `/display.html` – default display template location
            - `/setup.html` – default setup template location
    - `/queries` – defined SPARQL queries and query adapters
    - `/miniapp.js` – the miniapplication configuration file

5.9.3 Suggested layout structure

- `%LAYOUT_ID%` – the containing directory serves as a unique identifier
    - `/public`
        - `/assets` – optional assets to be included on the client-side
        - `/controllers` – optional layout controllers directory
        - `/views` – client templates
            - `/display.html` – default display template location
            - `/setup.html` – default setup template location
    - `/layout.js` – the layout configuration file

6 Evaluation

In this chapter we are going to evaluate the developer customizable Linked Data browser to find out if the project solution satisfies all the requirements. Two of the crucial properties of the browser will be evaluated – the ability to customize data presentation and performance.

6.1 Demonstration of browser customization

6.1.1 Custom address field

The first customization demonstrates how to build a custom display for the address field, namely for entities of type `http://schema.org/PostalAddress`. Values of this type are by default displayed as a list of URIs, we are going to alter this default behavior to display a string containing the street address, postal code and address locality.

STEP 1: Firstly, let us create miniapplication root folder that will contain all the associated files:

```
%PROJECT_ROOT%/miniapps/s-postal-address
```

STEP 2: Let us create a miniapplication configuration file:

```
%PROJECT_ROOT%/miniapps/s-postal-address/miniapp.js
```

```
var _ = require('lodash');

var matchInstancesFunction = function (resourceGraph) {
    var instances = [];
    _.forEach(resourceGraph.property, function (property) {
        if (_.includes(property.sampleTypes,
"http://schema.org/PostalAddress")) {
            instances.push({
                property: property['@id'],
                relation: property['relation']
            });
        }
    });
    return instances;
};
```

```
module.exports = {
    description: 'Postal address display',
    matchInstances: matchInstancesFunction,
    inhibitInstances: function (resourceGraph) {
        return [{
            miniapp: '*',
            instances: matchInstancesFunction(resourceGraph)
        }];
    },
    displayPriority: 100,
    setupPriority: 0
};
```

Figure 30 Custom address miniapplication configuration

The configuration file is designed in such a way that it provides a miniapplication instance for every property of the resource graph which is of RDF type `http://schema.org/PostalAddress`. At the same time, it inhibits all similar instances of all other miniapplications with setup priority defined lower than zero.

STEP 3: Now let us create a miniapplication controller.

`%MINIAPP_ROOT%/public/controllers/s-postal-address-controller.js`

```
angular.module('app.miniapps')
.controller('SPostalAddressController',
    ['$scope', 'Miniapp', 'lodash', 'Describe',
    function($scope, Miniapp, _, Describe) {
        Miniapp.decorateScope($scope);

        $scope.addressesLoaded = false;

        var properties = [
                 "http://schema.org/streetAddress",
                 "http://schema.org/addressLocality",
                 "http://schema.org/postalCode"];

        Describe.describeProperty($scope.$resource, $scope.$instance, 5, 0,
                                                             properties)
                    .then(function(data) {
                        var miss = false;
                        var sample = data.data[0];
                        _.forEach(properties, function(property) {
                            if (!_.has(sample, property)) {
                                miss = true;
                                return false; // exit loop
                            }
                        });
                        if (!miss) {
                            $scope.$property = data;
                            $scope.addressesLoaded = true;
                        }
                    });

                $scope.showMore = function() {
                    $scope.more = true;
                };

                $scope.results = [];

                $scope.datasource = {
                    get: function(offset, limit) {
                        return Describe.describeProperty($scope.$resource,
$scope.$instance, limit, offset, properties)
                            .then(function(data) {
                                return data.data;
                            });
                    }
                };

            }
        ]);
```

Figure 31 Custom address miniapplication controller

When initialized, the miniapplication issues a request to load custom data determined by the list of selected properties by calling `Describe.describeProperty` function. When the data is received, the result is inserted into the `$scope.$property` variable. Furthermore, a datasource is defined (`$scope.datasource`) to provide pagination function. Other defined variables are helpers for the template we are going to create in the next step.

STEP 4: The display template. Let us create a file:

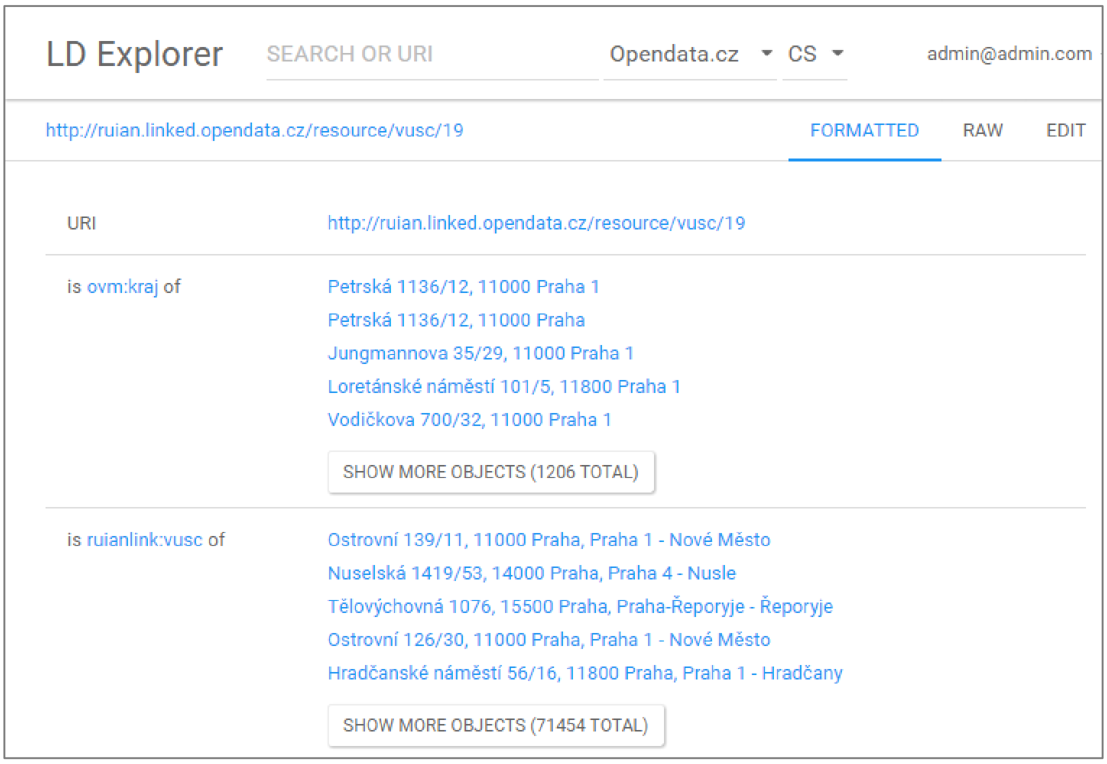
`%MINIAPP_ROOT%/public/views/display.html`

```
<div ng-controller="SPostalAddressController" class="miniapp-underlined">
    <div class="col-sm-3">
        <span ng-if="$property.relation == 'object'">is</span>
        <a describe resource="{{ $property['@id'] }}">
           {{ $property | label | contract | truncateUri }}
        </a>
        <span ng-if="$property.relation == 'object'">of</span>
    </div>
    <div class="col-sm-9">
        <div ng-if="!addressesLoaded">
            <div ng-repeat="item in $property.data track by item['@id']">
                <a describe="item['@id']">
                    {{ item | label | contract | truncateUriLarge }}
                </a>
            </div>
        </div>
        <div ng-if="!more && addressesLoaded">
            <div ng-repeat="item in $property.data track by item['@id']">
                <a describe="item['@id']">
                    {{ item['http://schema.org/streetAddress'] | value }},
                    {{ item['http://schema.org/postalCode'] | value }}
                    {{ item['http://schema.org/addressLocality'] | value }}
                </a>
            </div>
            <div ng-if="$property.count > $property.data.length">
               <hr class="spacer-10"/>
               <button class="btn btn-default btn-sm" ng-click="showMore()">
                   Show more objects ({{ $property.count }} total)
               </button>
            </div>
        </div>
        <div ng-if="more">
            <div ng-if="!datasource.isLoading"
                        ng-repeat="item in results track by item['@id']">
                <div ng-class-even="'row-even'" ng-class-odd="'row-odd'">
                    {{ (datasource.page - 1) * 10 + $index + 1 }}.
                    <a describe="item['@id']">
                     {{ item['http://schema.org/streetAddress'] | value }},
                     {{ item['http://schema.org/postalCode'] | value }}
                     {{ item['http://schema.org/addressLocality'] | value }}
                    </a>
                </div>
            </div>

            <div class="well well-lg text-center"
                 ng-if="datasource.isEmpty">
                Server responded with an empty result.
            </div>

            <loading-bar ng-show="datasource.isLoading"></loading-bar>

            <div datapager source="datasource" target="results"
                 items-per-page="10" total-count="{{ $property.count }}">
            </div>
        </div>

    </div>
</div>
```

Figure 32 Custom address miniapplication display template

Although both the controller and the display template might seem too complex, the solution provides the best user experience possible. When the web page and the resource currently displayed are loaded, default information is presented, that is, a list of URIs of the addresses. When the first Describe request in the miniapplication controller is performed and the data is received, the list of URIs is replaced with a list of formatted addresses, that is, a list of strings containing a street address, postal code and address locality. Furthermore, if there are more than five entries present, a “Show more objects” is displayed to the user. Upon clicking the button, the original address list is replaced with a numbered list, and in addition a pagination is included, so that the user can seamlessly browse the list even if it contains thousands of addresses.

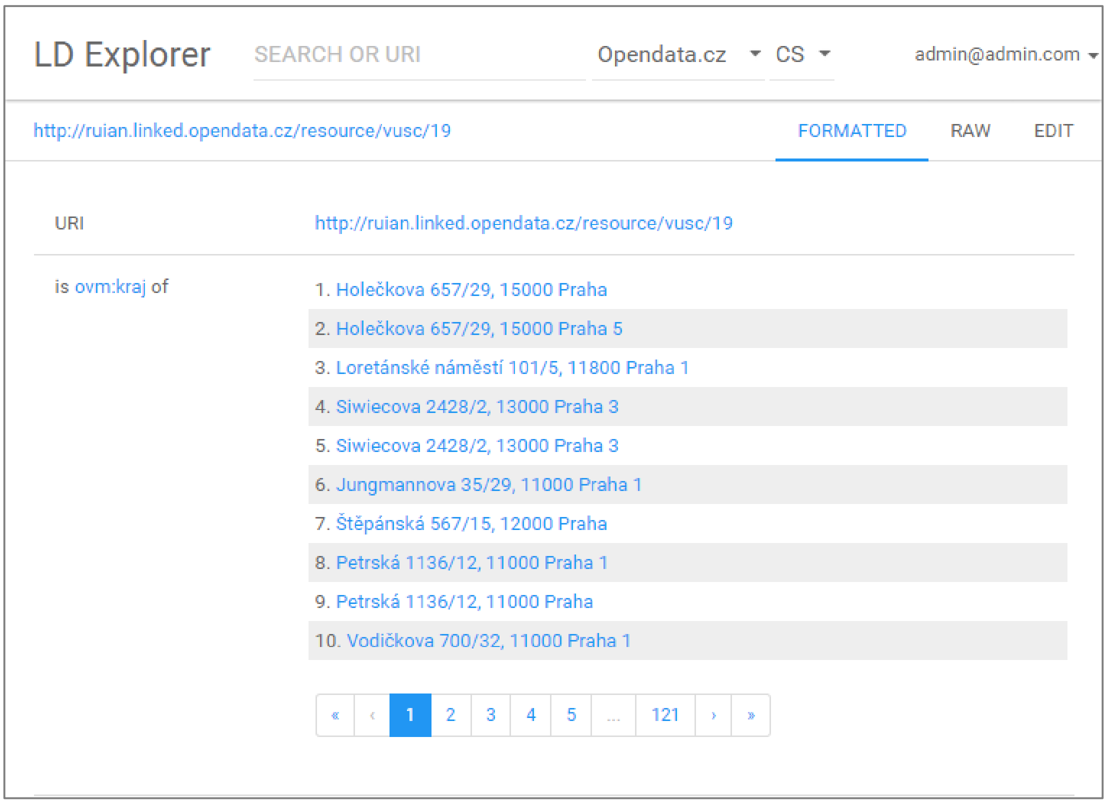
STEP 5: Verification of the result.



http://ruian.linked.opendata.cz/resource/vusc/19

URI	http://ruian.linked.opendata.cz/resource/vusc/19
is ovm:kraj of	Petrská 1136/12, 11000 Praha 1 Petrská 1136/12, 11000 Praha Jungmannova 35/29, 11000 Praha 1 Loretánské náměstí 101/5, 11800 Praha 1 Vodičkova 700/32, 11000 Praha 1
is ruianlink:vusc of	Ostrovní 139/11, 11000 Praha, Praha 1 - Nové Město Nuselská 1419/53, 14000 Praha, Praha 4 - Nusle Tělovýchovná 1076, 15500 Praha, Praha-Řeporyje - Řeporyje Ostrovní 126/30, 11000 Praha, Praha 1 - Nové Město Hradčanské náměstí 56/16, 11800 Praha, Praha 1 - Hradčany

Figure 33 Custom address field - default view



http://ruian.linked.opendata.cz/resource/vusc/19

URI	http://ruian.linked.opendata.cz/resource/vusc/19
is ovm:kraj of	1. Holečkova 657/29, 15000 Praha 2. Holečkova 657/29, 15000 Praha 5 3. Loretánské náměstí 101/5, 11800 Praha 1 4. Siwiecova 2428/2, 13000 Praha 3 5. Siwiecova 2428/2, 13000 Praha 3 6. Jungmannova 35/29, 11000 Praha 1 7. Štěpánská 567/15, 12000 Praha 8. Petrská 1136/12, 11000 Praha 1 9. Petrská 1136/12, 11000 Praha 10. Vodičkova 700/32, 11000 Praha 1

Figure 34 Custom address field - expanded view

6.1.2 Custom company field

The second customization demonstrates how to build a custom display for a company field in such a way that the application displays the company's name and identification number. In comparison with the first example, this demonstration will have to employ a custom SPARQL query.

The miniapplication will be applied to properties of type `http://purl.org/goodrelations/v1#BusinessEntity` or `http://schema.org/Organization`.

The identification number of a company is in fact a resource of type `adms:Identifier` that has a property `skos:prefLabel` that contains the desired identification number value.

In addition, the company might not contain the property with its identification number, but instead may contain the `owl:sameAs` property that leads to another resource that may or may not include the desired identification number. The SPARQL query will therefore need to examine the chain of `owl:sameAs` links and find the value, if it exists somewhere in the chain.

STEP 1: Firstly, let us create a miniapplication root folder that will contain all the associated files:

```
%PROJECT_ROOT%/miniapps/organization-property
```

STEP 2: Now we are going to prepare the custom SPARQL query. Let us create the SPARQL file:

```
%MINIAPP_ROOT%/queries/get-info.sparql
```

```
PREFIX my: <http://my/>

CONSTRUCT {
    {{ resource }} my:title ?title .
    {{ resource }} my:identifier ?id .
}

WHERE {
    {{ resource }} dcterms:title ?title .
    {{ resource }} owl:sameAs* ?sameObj .
    ?sameObj adms:identifier ?idObject .
    ?idObject skos:prefLabel ?id
}
```

Figure 35 Custom company miniapplication SPARQL query

The `{{ resource }}` expression will be later replaced by the actual resource URI.

STEP 3: Next, we are going to create a query adapter:

`%MINIAPP_ROOT%/queries/get-info-adapter.js`

```
module.exports = function(query) {

    query.getContext = function() {
        return {
            "title" : "http://my/title",
            "identifier" : "http://my/identifier"
        }
    };

};
```

Figure 36 Custom company miniapplication SPARQL query adapter

What this adapter does is pretty simple. The SPARQL query returns a JSON object that has two properties – `http://my/title` and `http://my/identifier`. If we specify a context in the way presented, those properties will be translated to `title` and `identifier` respectively, thus when accessing those properties in a template, we do not have to enter the full URI with the `http://my/` prefix.

STEP 3: Let us create a miniapplication configuration file:

```
%MINIAPP_ROOT%/miniapp.js
```

```
var _ = require('lodash');

var Broker = require('../../server/lib/sparql-query-broker');

function partnerMatchInstances(resourceGraph) {
    var instances = [];
    _.forEach(resourceGraph.property, function (property) {
        if (property.relation != 'subject'
            || property.data.length != 1) {
            return; // continue
        }
        if (_.includes(property.sampleTypes,
            'http://purl.org/goodrelations/v1#BusinessEntity')
            || _.includes(property.sampleTypes,
            'http://schema.org/Organization')) {
            instances.push({
                property: property['@id'],
                relation: property['relation']
            });
            return false; // exit loop
        }
    });
    return instances;
}

module.exports = {
    description: 'Organization or BusinessEntity display',
    matchInstances: partnerMatchInstances,
    inhibitInstances: function(resourceGraph) {
        return [{
            miniapp: '*',
            instances: partnerMatchInstances(resourceGraph)
        }];
    },
    setupApplication: function(app, auth) {

        var query = require('./queries/get-info.sparql');
        var adapter = require('./queries/get-info-adapter');
        var brokerInstance = new Broker(query, adapter);

        app.route('/api/organization-property')
            .get(brokerInstance.serve);

    },

    displayPriority: 100,
    setupPriority: 0

};
```

Figure 37 Custom company miniapplication configuration

The `matchInstances` and `inhibitInstances` functions are similar to the custom address miniapplication demonstrated earlier. In addition, the configuration file defined a `setupApplication` function that registers a REST API provider for the `/api/organization-property` URL. When sending a GET request to such URL along with a *resource* parameter, the parameter replaces the {{ resource }} element present in the SPARQL query and processes the query. The Broker component sends a request including the prepared SPARQL query to a SPARQL endpoint and waits for the result. When the response arrives, the Broker delegates it to the query adapter which modifies the response (in this case it renames two of the properties of the JSON-LD object). The modified response is then sent back to the client and the original REST request is thus complete.

STEP 4: Now let us create a miniapplication controller.

`%MINIAPP_ROOT%/public/controllers/organization-property-controller.js`

```
angular.module('app.miniapps')
    .controller('OrganizationPropertyController',
       ['$scope','Miniapp', 'lodash', 'Describe',
       function($scope, Miniapp, _, Describe) {

                Miniapp.decorateScope($scope);

                $scope.organizationLoaded = false;

                Miniapp.request('/api/organization-property',
                      { resource: $scope.$property.data[0]['@id'] })
                    .then(function(data) {
                          $scope.$property.data = data['@graph'];
                          $scope.organizationLoaded = true;
                        }
                    });
            }
        ]);
```

Figure 38 Custom company miniapplication controller

What this controller does is clear – when it is initialized, it invokes an API request to the custom REST provider, providing the resource parameter. When the data is loaded, it is placed to the `$scope.$property.data` variable.

STEP 5: The display template. Let us create a file:

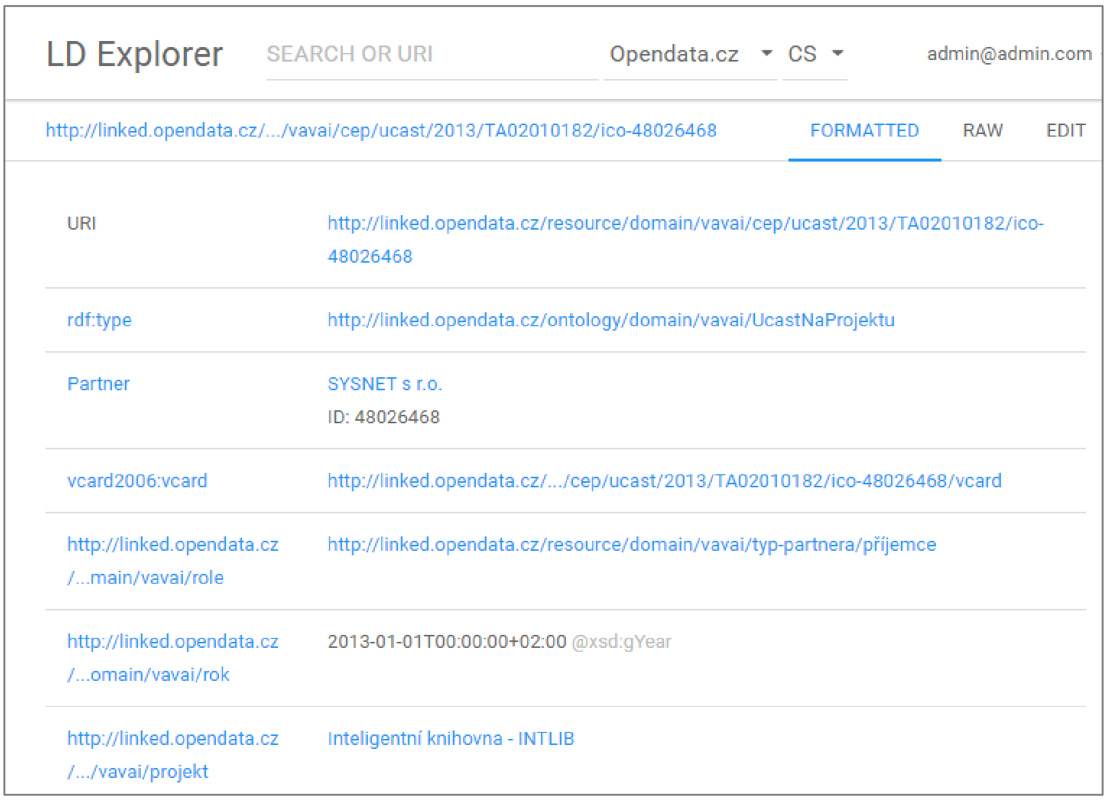
```
%MINIAPP_ROOT%/public/views/display.html
```

```
<div ng-controller="OrganizationPropertyController"
     class="miniapp-underlined">
    <div class="col-sm-3">
        <a describe resource="{{ $property['@id'] }}">
           {{ $property | label | contract | truncateUri }}
        </a>
    </div>
    <div class="col-sm-9">
        <span ng-if="!organizationLoaded">
            <span ng-init="firstObject = $property.data[0]">
                <a describe="firstObject['@id']">
              {{ firstObject | label | contract | truncateUriLarge }}
                </a>
            </span>
        </span>
        <span ng-if="organizationLoaded">
            <span ng-init="organization = $property.data[0]">
                <a describe="sampleAddress['@id']">
                    {{ organization['title'] | value }}
                </a><br/>
                ID: {{ organization['identifier'] | value }}
            </span>
        </span>
    </div>
</div>
```

Figure 39 Custom company miniapplication display template

The template in tandem with the controller provide great user experience similar to the custom address miniapplication. When the web page is loaded, the default information is displayed, either URI or a label of the organization, if defined. Whenever is the custom API request issued in the controller completed, the data is replaced with the custom view that displays the name of the organization as well as its identification number.

STEP 6: Verification of the result.


LD Explorer SEARCH OR URI Opendata.cz CS admin@admin.com

http://linked.opendata.cz/.../vavai/cep/ucast/2013/TA02010182/ico-48026468 FORMATTED RAW EDIT

URI	http://linked.opendata.cz/resource/domain/vavai/cep/ucast/2013/TA02010182/ico-48026468
rdf:type	http://linked.opendata.cz/ontology/domain/vavai/UcastNaProjektu
Partner	SYSNET s r.o. ID: 48026468
vcard2006:vcard	http://linked.opendata.cz/.../cep/ucast/2013/TA02010182/ico-48026468/vcard
http://linked.opendata.cz /...main/vavai/role	http://linked.opendata.cz/resource/domain/vavai/typ-partnera/příjemce
http://linked.opendata.cz /...omain/vavai/rok	2013-01-01T00:00:00+02:00 @xsd:gYear
http://linked.opendata.cz /.../vavai/projekt	Inteligentní knihovna - INTLIB

Figure 40 Custom company miniapplication default view

6.1.3 Two-column layout

This section demonstrates the possibility to create a custom two-column layout.

STEP 1: Firstly, let us create a layout root folder that will contain all the associated files: `%PROJECT_ROOT%/layouts/two-column`

STEP 2: Next, let us create a layout configuration file:

`%PROJECT_ROOT%/layouts/two-column/layout.js`

We are going to configure two panels, *main* and *side*.

```
module.exports = {
    name: 'Two Columns',
    defaultPanel: 'main',
    panels: ['main', 'side']
};
```

Figure 41 Two-column layout configuration file

STEP 3: Now it is time to create display and setup templates. The templates should be located the following files:

```
%LAYOUT_ROOT%/public/views/display.html
```

```
%LAYOUT_ROOT%/public/views/setup.html
```

```
<div layout class="row layout-two-columns">
    <div layout-panel="main" class="col-md-8 main-panel"></div>
    <div layout-panel="side" class="col-md-4 side-panel"></div>
</div>
```

Figure 42 Two-column layout display template

```
<div layout class="row layout-two-columns">
<div layout-panel-setup="main" class="col-md-8 main-panel"></div>
<div layout-panel-setup="side" class="col-md-4 side-panel"></div>
</div>
```

Figure 43 Two-column layout setup template

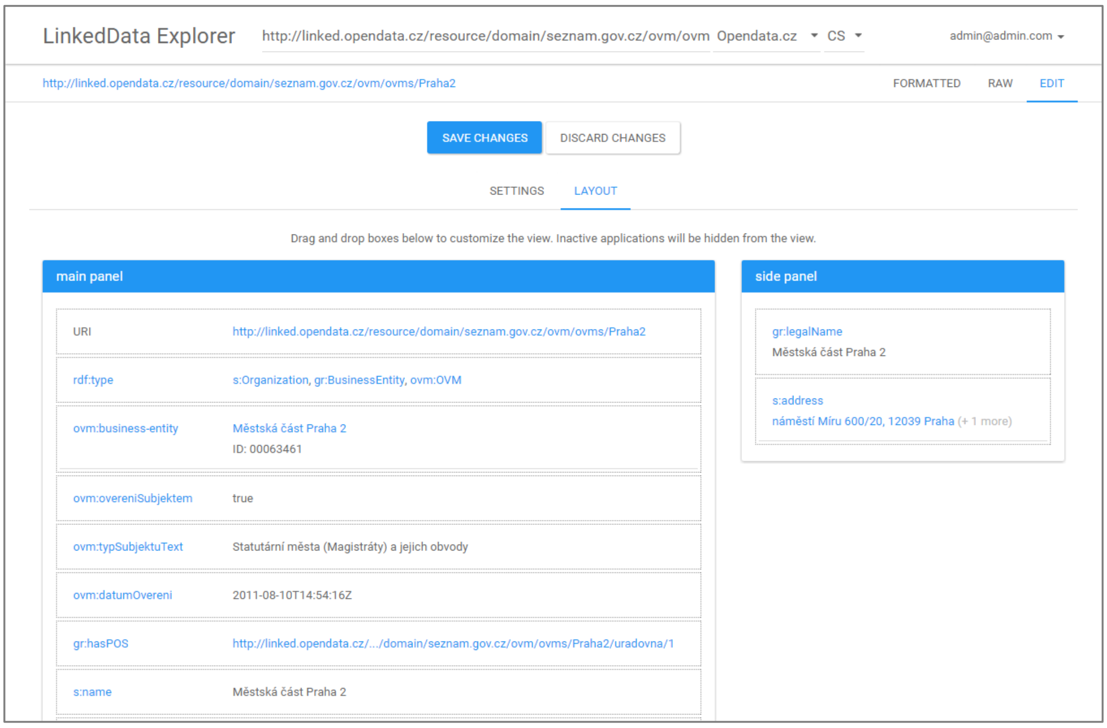
STEP 4: Fine-tuning the styling. The *default* miniapplication that displays resource properties presents the information in a tabular form – implicitly, it displays the property name on the left side of the row and the value on the right. This presentation is alright for the main panel, but the sidebar is too narrow, so that we need to alter the stylesheet in such a way that the property name is displayed above the data. Let us create a CSS file:

```
%LAYOUT_ROOT%/public/assets/layout.css
```

```
@media (min-width: 992px) {
    .layout-two-columns .side-panel .col-sm-3,
    .layout-two-columns .side-panel .col-sm-9
    {
        float: none;
        width: auto;
    }
}
```

Figure 44 Two-column layout custom CSS stylesheet

STEP 5: The layout is configured, now we need to verify the result, thus we are going to create a custom view employing the created layout.



LinkedData Explorer http://linked.opendata.cz/resource/domain/seznam.gov.cz/ovm/ovm Opendata.cz CS admin@admin.com

http://linked.opendata.cz/resource/domain/seznam.gov.cz/ovm/ovms/Praha2 FORMATTED RAW EDIT

SAVE CHANGES DISCARD CHANGES

SETTINGS LAYOUT

Drag and drop boxes below to customize the view. Inactive applications will be hidden from the view.

main panel

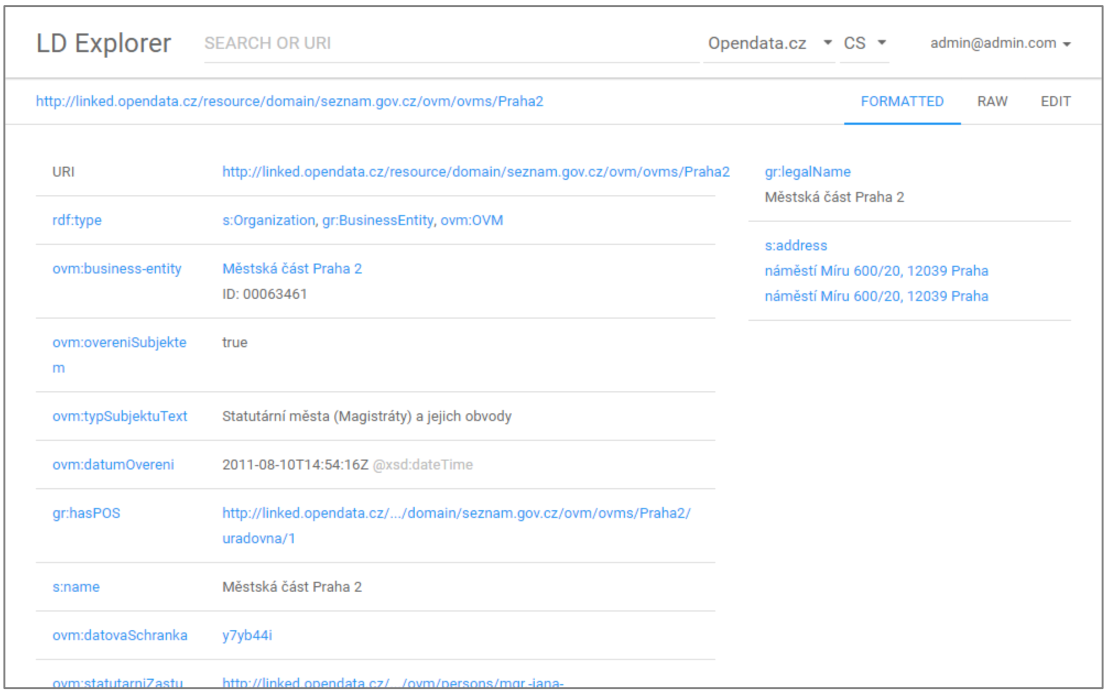
URI	http://linked.opendata.cz/resource/domain/seznam.gov.cz/ovm/ovms/Praha2
rdf:type	s:Organization, gr:BusinessEntity, ovm:OVM
ovm:business-entity	Městská část Praha 2 ID: 00063461
ovm:overeniSubjektem	true
ovm:typSubjektuText	Statutární města (Magistráty) a jejich obvody
ovm:datumOvereni	2011-08-10T14:54:16Z
gr:hasPOS	http://linked.opendata.cz/.../domain/seznam.gov.cz/ovm/ovms/Praha2/uradovna/1
s:name	Městská část Praha 2

side panel

gr:legalName
Městská část Praha 2

s:address
náměstí Míru 600/20, 12039 Praha (+ 1 more)

Figure 45 Two-column layout - setup of a custom view



LD Explorer SEARCH OR URI Opendata.cz CS admin@admin.com

http://linked.opendata.cz/resource/domain/seznam.gov.cz/ovm/ovms/Praha2 FORMATTED RAW EDIT

URI	http://linked.opendata.cz/resource/domain/seznam.gov.cz/ovm/ovms/Praha2
rdf:type	s:Organization, gr:BusinessEntity, ovm:OVM
ovm:business-entity	Městská část Praha 2 ID: 00063461
ovm:overeniSubjektem	true
ovm:typSubjektuText	Statutární města (Magistráty) a jejich obvody
ovm:datumOvereni	2011-08-10T14:54:16Z @xsd:dateTime
gr:hasPOS	http://linked.opendata.cz/.../domain/seznam.gov.cz/ovm/ovms/Praha2/uradovna/1
s:name	Městská část Praha 2
ovm:datovaSchranka	y7yb44i
ovm:statutarniZastu	http://linked.opendata.cz/.../ovm/persons/mgr-jana-

gr:legalName
Městská část Praha 2

s:address
náměstí Míru 600/20, 12039 Praha
náměstí Míru 600/20, 12039 Praha

Figure 46 Two-column layout - custom view

6.2 Performance

6.2.1 Testing methodology

To test the performance of the developed application, we are going to browse arbitrary selected resources and measure the time it takes to display the resource. Each resource will be accessed ten times and the resulting time will be calculated as an average of the ten attempts' times.

Because the developed browser embraces the Single-page application principle, it takes some time to initially load the application. Due to this fact two sets of measurements will be performed for each resource, one set consisting of the time it takes to display the particular resource along with full loading the application, whereas regarding the other set the application will be already preloaded.

The test will be performed on Opendata.cz SPARQL endpoint[40] using these resources:

1. http://linked.opendata.cz/resource/domain/seznam.gov.cz/ovm/ovms/Praha2
   Total number of relations: 35.
2. http://linked.opendata.cz/resource/legislation/cz/act/2012/89-2012
   Total number of relations: 3100.
3. http://ruian.linked.opendata.cz/resource/vusc/19
   Total number of relations: 73000.
4. http://purl.org/vocab/frbr/core#Work
   Total number of relations: 267325.

In addition, to include customization properties of the browser to the mix, each resource will be displayed using a custom view consisting of a two-column layout and arbitrary sorted properties.

[40] http://linked.opendata.cz/sparql

6.2.2 Testing environment

The application will be tested in the following environment:

- Laptop Lenovo ThinkPad X1
    - Intel Core i7-5500U CPU @ 2.40 GHz
    - 8 GB RAM
    - Windows 7 Professional 64bit
- Google Chrome browser, version 44.0.2403.107
- Internet connection: VDSL 16 Mbit download / 1 Mbit upload

6.2.3 Test results

Table 3 below shows the average times measured.

Resource number	Average time Full load	Average time Preloaded application
1	4.25 s	2.6 s
2	3.7 s	1.8 s
3	19.3 s	17.6 s
4	7.2 s	5.7 s

Table 3 Performance test results

6.3 Evaluation summary

Regarding the evaluation of customization options, the application satisfied all the requirements defined in the Analysis chapter. Furthermore, it is safe to assume that design and implementation of the application exceeded those requirements, especially regarding the *low level* customization. It is evident that the introduction of miniapplications instead of planned plain templates was indeed a great design decision.

As for the evaluation of performance, the application operates within required time boundaries. A nice surprise is that the average time to fully load the application is not much higher than the performance of preloaded application, which confirms that the Single-page design approach is indeed a viable option and preloading all required resources at once does not significantly affect the performance.

7 Conclusion

7.1 Achieved goals

The goal of the thesis was to identify key requirements for exploring Linked Data and design and implement a web application which would serve as a Linked Data browser, including certain customization features. In comparison to existing approaches the application should have enabled the user to provide templates that would define a visual style for presentation of particular types of Linked Data resources.

After a thorough research of existing Linked Data browsers, the analysis proposed a two layer customization solution. On the lower level, the customization of the application would follow principles stated in the original goal and enable expert users to modify the default data presentation style by providing templates for particular data types. In addition, the higher level of customization would allow non-expert users – that is, users with no or limited previous knowledge of RDF or Linked Data principles – to alter the final appearance of data presentation by creating a customized view and rearranging the layout of the data of each resource. Based on the analysis, a solution was designed and implemented.

Thanks to the two layer customization design, the developed solution exceeded the originally declared goal, as it provides truly versatile and thorough application customization options.

In comparison with existing identified solutions, regarding the expert users the developed browser provides arguably the best customization opportunities, especially thanks to the introduction of the miniapplication concept. However, regarding the non-expert users, the developed browser undoubtedly outshines all existing products, thanks to a unique approach of implementing the What You See Is What You Get visual method of customizing the presentation of data.

7.2 Future work

The solution presented in the thesis offers a number of future work opportunities. First and foremost, the developed browser could be extended to support plurality of data sources at once and thus be able to obtain data from various data sources in parallel and present the merged result to the user. However, such a solution will probably be hard to achieve computing-wise as it has the potential to easily result in performance drawbacks.

Another major possibility to extend the developed browser is to design a set of default templates for existing dominant ontologies in the Linked Data world such as Friend of a friend[41] or Dublin Core Metadata Terms.[42]

Finally, let us conclude the thesis by a rather utopian, but all the more exciting thought. Given the fact that the browser leverages Linked Data, and in the world of Linked Data every resource has its own URI, it would be suitable to provide each custom miniapplication, layout and view with such an identifier and create a global repository of these customization elements. In addition, the developed Linked Data browser could be extended to facilitate straightforward import of those elements to a particular instance of the application just by providing a resource URI of the desired custom element.

[41] FOAF ontology. http://www.foaf-project.org/
[42] Dublin Core Metadata Terms. http://dublincore.org/documents/dces/

Bibliography

[1] BERNERS-LEE, Tim, et al. The semantic web. Scientific american, 2001, 284.5: 28-37.

[2] BIZER, Christian; HEATH, Tom; BERNERS-LEE, Tim. Linked data-the story so far. Semantic Services, Interoperability and Web Applications: Emerging Concepts, 2009, 205-227.

[3] BERNERS-LEE, Tim. Linked data-design issues (2006). URL http://www.w3.org/DesignIssues/LinkedData.html, 2011.

[4] RDF. URL http://www.w3.org/RDF/.

[5] JSON for Linking Data. URL http://json-ld.org/

[6] RUSHER, Jack. Triple Store, Feb. 2005. World Wide Web Consortium. URL http://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/rusher.html.

[7] RAPOZA, Jim. SPARQL will make the web Shine. eWeek. 2006.

[8] SPARQL 1.1 Query Language. World Wide Web Consortium. URL http://www.w3.org/TR/2013/REC-sparql11-query-20130321/

[9] SPARQL 1.1 Update. World Wide Web Consortium. URL http://www.w3.org/TR/sparql11-update/

[10] RICHARDSON, Leonard; RUBY, Sam. RESTful web services. O'Reilly Media, Inc., 2008.

[11] The Tabulator. URL http://www.w3.org/2005/ajar/tab

[12] Marbles. URL http://mes.github.io/marbles/

[13] DE ARAÚJO, Samur FC; SCHWABE, Daniel. Explorator: a tool for exploring RDF data through direct manipulation. In: Linked data on the web WWW2009 workshop (LDOW2009). 2009.

[14] Graphity Client. URL http://graphityhq.com/technology/graphity-client

[15] BERNERS-LEE, Tim, et al. Tabulator: Exploring and analyzing linked data on the semantic web. In: Proceedings of the 3rd International Semantic Web User Interaction Workshop. 2006.

[16] SPARQL 1.1 Protocol. URL http://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/

[17] DAVIS, Scott. Mastering MEAN: Introducing the MEAN stack. IBM developerWorks. URL http://www.ibm.com/developerworks/library/wa-mean1/index.html

[18] Node.js. URL https://nodejs.org/

[19] JSON web token (JWT). IETF. URL http://self-issued.info/docs/draft-ietf-oauth-json-web-token.html

List of Tables

List of Figures

List of Abbreviations

LD	Linked Data
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
HTTP	HyperText Transfer Protocol
HTML	HyperText Markup Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations
JSON	JavaScript Object Notation
JSON-LD	JSON for Linking Data
MVC	Model–View–Controller Architecture
REST	Representational State Transfer
CRUD	Create, Read, Update, Delete

Attachments

A ZIP file containing the developed application is attached.

An instance of the application is also available via the Internet on the following URL:

http://linked-data-browser.herokuapp.com/

Installation requirements:

- Node.JS version 0.10 or newer.
- Python version 2.7.
- A web browser.

Installation instructions:

1. Extract the attached archive to a desired directory.
2. Open Node.JS command prompt and navigate to the directory containing extracted application files.
3. Run command “npm install” to install the application.
4. Run command “node bootstrap.js” to initialize the application.
5. Using a web browser, navigate to http://localhost:3000.