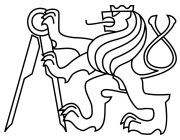
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF MASTER'S THESIS

Title:	Collection, Transformation, and Integration of Data from the Web Services Domain
Student:	Bc. Radmir Usmanov
Supervisor:	Ing. Milan Dojčinovski
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of winter semester 2018/19

Instructions

Currently, there are several repositories and data models that provide descriptions for Web APIs. The main goal of the thesis is to establish an automated collection, transformation, and integration of several data sources with Web API descriptions.
Guidelines:
- Analyze and get familiar with existing datasets and data models for Web APIs.
- Establish mappings between different data models.
- For each individual data source, design and implement an automated collection, transformation, and integration into the unified data model. The Linked Web APIs model will provide the basis and if needed further extended.
- Validate and evaluate the knowledge extraction process for each identified Web API dataset.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague February 21, 2017

Czech Technical University in Prague
Faculty of Information Technology
Department of Software Engineering

Master's thesis

Collection, Transformation, and Integration of Data from the Web Services Domain

Bc. Radmir Usmanov

Supervisor: Milan Dojčinovski

29th June 2017

Acknowledgements

I would like to thank my supervisor Milan Dojčinovski for his guidance, constructive notes and insights.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 29th June 2017 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Usmanov, Radmir. *Collection, Transformation, and Integration of Data from the Web Services Domain.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

V současné době existuje několik repozitářů a datových modelů, které poskytují popisy webových služeb. Diplomová práce řeší problém transformace popisů webových služeb z několika datových modelů do jednoho sjednoceného datového modelu. Práce analýzuje existující datasety a datové modely pro webové služby, vytváří mapování mezi různými datovými modely, automatizuje sběr, transformace a integrace datových modelů webových služeb do jednotného datového modelu, ověřuje a vyhodnocuje výsledky extrakce.

Klíčová slova Sémantický web, Linked Data, Sémantické webové služby, RDF, Linked Web API, Web Scrapping, Web Crawling, WSMO, WSMO-Lite

Abstract

Currently, there are several repositories and data models that provide descriptions for Web APIs. The diploma thesis tackles the problem of transforming descriptions of Web APIs from several data models into one unified data model. It analyzes existing datasets and data models for Web APIs, establishes mappings between different data models, collects, transforms and integrates Web APIs data models into the unified data model, validates and evaluates extraction results.

Keywords Semantic Web, Linked Data, Semantic Web Services, RDF, Linked Web API, Web Scrapping, Web Crawling, WSMO, WSMO-Lite

Contents

List of Figures

List of Tables

Introduction

Motivation

The World Wide Web is an information space where documents and other web resources are identified by Uniform Resource Locators (URLs), interlinked by hypertext links, and can be accessed via the Internet.

Nowadays the World Wide Web is used in many aspects, for studying, for searching information, for entertainment, for communication needs and etc. The WWW[1] initially was created as a hub of information for humans, for human needs and wasn't adapted for machine processing.

Meanwhile, the internet has been developed as a vast network of applications communicating and interlinking with each other. The applications are exchanging the data between each other, produce and consume it among the network. This was made possible by Web Services and APIs[2].

In the last decades, Web APIs have significantly increased in popularity. It becomes the core technology of web functionality and affect the whole WWW. Nevertheless, due to the lack of Web APIs semantic descriptions, Web Services have drawbacks in discovery, sharing, composition and monitoring.

Despite the fact of existing several directories describing Web APIs, there is still no generic Web APIs dataset. The Web APIs dataset will unify Web Services and RESTful APIs, describe them using semantic annotations and make them more discoverable and shareable.

The goal

The main goal of the thesis is to transform descriptions of Web APIs from several data models into one unified data model.

To achieve this, the following tasks need to be solved:

[1] World Wide Web

[2] Application Programmable Interface

- Analyze existing datasets and data models.
- Establishing mappings between data models.
- Establish an automated collection of Web APIs descriptions from data sources.
- Transforming and integrating of extracted data into the unified data model. The Linked Web APIs model can be taken as the basis for the unified data model and if needed further extended.

As a result, data must become available as a Semantic Web APIs dataset and can be consumed for further usage and development.

Thesis structure

The thesis is divided into four chapters.

The first chapter *Background and related work* describes basic concepts associated with this work and research of existing solutions, such as existing Web Services data models and Web API directories.

The second chapter which is called *Analysis and Design* defines functional and non-functional requirements of the application, use cases and ontology vocabularies that are used to create a unified data model. The concept of unified data model is also described in this chapter. Moreover, the chapter describes the architecture of application, approaches which provide a solution for transforming data from multiple heterogeneous sources into semantic data representation using RDF framework.

In the third chapter, which is called *Implementation*, we have represented technologies that are used in the thesis, and the implementation of the application. Besides, the chapter contains information about application deployment and existing implementation issues.

The last fourth chapter describes the coverage of the work - statistics of how many APIs were discovered and extracted. Also, the chapter classifies the quality of the unified data model from several sides.

Chapter 1

Background and related work

1.1 Background

This section introduces theoretical part of the work, describing ideas and principles that were used in creating of application. Most of the concepts and technologies talked here are Semantic Web, Linked Data, and Web APIs.

1.1.1 API

API is a set of subroutine definitions, protocols, and tools for building application software. In general terms, it is a set of clearly defined methods of communication between various software components.[6] An API is an architectural approach that revolves around providing programmable interfaces of a set of services to different applications serving different types of consumers.

1.1.1.1 Web API

A web API is an application programming interface (API) for either a web server or a web browser.[7]

Web API is an API over the internet which can be accessed using HTTP protocol. It is a web development concept that defines interfaces through which interactions happen between a service and applications that use its assets, see figure 1.1.

Web API can be separted into two types:

- Server-side API
- Client-side API

The most popular API is a server-side API, which provides a programmatic interface consisting of public endpoints, typically expressed in JSON or XML format. In this work we operate only with a client-side API.

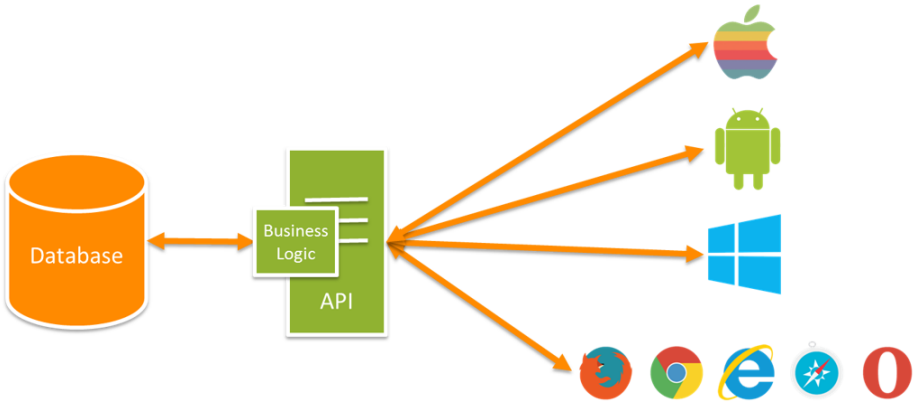


Figure 1.1: Web API [1]

Web APIs allow the combination of multiple APIs into new applications known as mashups.

1.1.2 Mashup

A mashup is a Web page or application that uses and combines data, presentation or functionality from two or more sources to create new services. The term implies easy, fast integration, frequently using open APIs and data sources to produce enriched results that were not necessarily the original reason for producing the raw source data.[8]

The main characteristics of the mashup are combination, visualization, and aggregation. Data provided by Web APIs become more useful offering possibility to create new valuable services.

Mashups are often defined by the type of content they aggregate. There are many types of mashups, such as business mashups, consumer mashups, and data mashups. The most common type of mashup is the consumer mashup, aimed at the general public.

1.1.3 Information extraction

Information extraction is the task of automatically extracting structured information from unstructured and semi-structured machine-readable documents.[9]

There are exist a lot of extraction techniques. The extraction techniques can be separated into the following categories:

- Methods based on manual rules
- Wrappers
- Algorithms of machine learning

- Bootstrapping
- Active (interactive) learning

We are not going deep in information extraction processes because it is a topic for another thesis. In our case, we use *Wrappers* extraction techniques, which helps to extract data from HTML documents traversing DOM[3] structure.

1.1.3.1 Web crawling

A web crawler (also known as a robot or a spider) is a system for the bulk downloading of web pages. Crawlers are one of the main components of web search engines, systems that assemble a corpus of web pages, index them, and allow users to issue queries against the index and find the web pages that match the queries.[10]

Data on the web can be crawled using different methods:

- **Data dumps**
  Sites may package their data and provide them as dumps.
- **URL downloads**
  Crawling process is based on URL templates, i.e a crawler browses only those pages which URLs are match to predefined URL template.
- **Web APIs**
  Sites provide RESTful API for getting their data.
- **Web Crawling**
  Crawler moves around interlinked HTML pages.

The basic web crawling algorithm is simple: Given a set of seed Uniform Resource Locators (URLs), a crawler downloads all the web pages addressed by the URLs, extracts the hyperlinks contained in the pages, and iteratively downloads the web pages addressed by these hyperlinks.

Despite the apparent simplicity of this basic algorithm, web crawling has many inherent issues:

1. **Politeness**
   Avoid requesting website too frequently.
2. **Performance**
   Crawlers must be implemented as multi-threaded applications or used distributed crawling.

[3]Document Object Model

3. **Crawler traps**
   Catching a crawler to infinite number of requests.

4. **Duplicate detection**
   Crawler needs to avoid visiting duplicate pages.

5. **Client-side scripting**
   Nowadays a lot of websites generates web pages using Javascript. A crawler must be smart enough to understand Javascript, interpret and run it. Otherwise, a crawler may miss a data, generated by Javascript.

1.1.3.2 Web scraping

Web scraping (web harvesting or web data extraction) is data scraping used for extracting data from websites. Web scraping software may access the World Wide Web directly using the HTTP[4] protocol, or through a web browser. While web scraping can be done manually by a software user, the term typically refers to automated processes implemented using a bot or web crawler. It is a form of copying, in which data is gathered and copied from the web, typically into a central data storage, for later retrieval or analysis [11].

The process of scraping information can be shown on the figure 1.2.

Current web scraping solutions range from the ad-hoc, requiring human effort, to fully automated systems that are able to convert entire web sites into structured information, with limitations [11]. Thus, various scraping techniques were created:

- **Human copy-and-paste**
  Human copy-and-paste is a technique of manually copying and pasting the web page content.

- **Text pattern matching**
  Text pattern matching technique is a technique based on regular expressions.

- **HTTP programming**
  HTTP programming is a technique where pages can be retrieved by posting HTTP requests to a server.

- **HTML parsing**
  HTML parsing is a technique of retrieving information from HTML documents using HTML structures.

- **DOM parsing**
  The technique retrieves the dynamic content generated by client-side Javascript.

[4]Hypertext Transfer Protocol

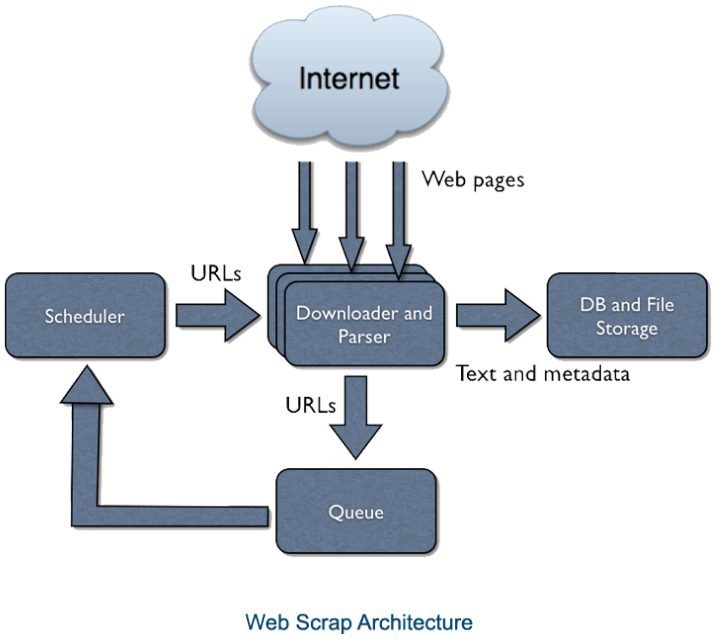


Figure 1.2: Web Scrapping Architecture[2]

- **Semantic annotation recognizing**
  Semantic annotation recognizing is a technique based on retrieving semantic annotation.
- **And couple more techniques**

The application that was created in this work uses only HTTP programming and HTML parsing techniques.

1.1.4 Semantic Web

According to the W3C (World Wide Web Consortium), "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries"[12]. Or in other words in article "The Semantic Web" Berners-Lee says "The Semantic Web is an extension of the current web in which information is given well-defined meaning, better-enabling computers and people to work in cooperation."

The collection of Semantic Web technologies (RDF, SPARQL, SAWSDL, MicroWSMO, WSMO-Lite, etc.) provides an environment where application can query data, draw inferences using vocabularies. These technologies are discussed in next sections.

1.1.4.1 Linked Data

Linked Data lies at the heart of what Semantic Web, is one of the core concepts of the Semantic Web. It is principles, best practices for publishing and linking entities on the internet. In other words, it is a method of publishing structured data so that it can be interlinked and become more useful through semantic queries.

Linked Data helps to create an eco-system of web applications which publish, enrich and consume data about things in one shared global data space.[13]

It builds upon standard web technologies such as HTTP, RDF and URIs[5]. Tim Berners-Lee outlined four principles of linked data:

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names
3. When someone looks up a URI, provide useful information, using standards (RDF, SPARQL)
4. Include links to other URIs so that they can discover more things

1.1.4.2 Linked Open Data

Linked Open Data is a powerful blend of Linked Data and Open Data: it is both linked and uses open data sources. In 2010 Tim Berners-Lee suggested a 5-star deployment scheme for Open Data[14].

1. Make your stuff available on the Web (whatever format) under an open license
2. Make it available as structured data (e.g., Excel instead of image scan of a table)
3. Make it available in a non-proprietary open format (e.g., CSV[6] instead of Excel)
4. Use URIs to denote things, so that people can point at your stuff
5. Link your data to other data to provide context

1.1.4.3 RDF

RDF stands for Resource Description Framework, is a standard model for data interchange on the Web.[15] It is a framework for describing resources on the web; it is designed to be read and understood by computers.

[5]Uniform Resource Identifier
[6]Comma-separated values

The information in RDF is represented by subject-predicate-object, known as triples. Triples are written in one of RDF notations: RDF/XML, RDFa, N-Triples, Turtle, JSON-LD and stored in a triplestore.

RDF is often used to represent, among other things, personal information, social networks, metadata about digital artifacts, as well as to provide a means of integration over disparate sources of information.[16]

1.1.4.4 SPARQL

SPARQL is an RDF query language, that is, a semantic query language for databases, able to retrieve and manipulate data stored in Resource Description Framework (RDF) format. SPARQL works for any data source that can be mapped to RDF.

SPARQL allows users to write queries against key-value data or, more specifically, data that can be mapped to RDF. The entire database is thus a set of subject-predicate-object triples.

1.1.4.5 Triplestores

Triplestores are Database Management Systems (DBMS[7]) for data modeled using RDF. Unlike Relational Database Management Systems (RDBMS), which store data in relations (or tables) and are queried using SQL[8], triplestores store RDF triples and are queried using SPARQL.

Being a graph database, triplestore stores data as a network of objects with materialized links between them. This makes RDF triplestore a preferred choice for managing highly interconnected data. Triplestores are more flexible and less costly than a relational database.

The RDF database often called a semantic graph database, is also capable of handling powerful semantic queries and of using inference for uncovering new information out of the existing relations.

1.1.4.6 Semantic Web Services

A semantic web service, like conventional web services, is the server end of a client-server system for machine-to-machine interaction via the World Wide Web.[17] It is a combination of web services and semantic web. Semantic web services use markup to make data more understandable to machines, thus making web services more understandable by machines.

Research in the Semantic Web services area is addressing the problems of automated service discovery, invocation, composition, and monitoring by augmenting the existing Web services standards by additional semantic layers. For Semantic Web services to become a reality, a markup language must be

[7]Database Management System
[8]Structured Query Language

descriptive enough that a computer can automatically determine its meaning. The following is a list of tasks such a language would be required to perform[18]:

- **Discovery**
  An application must be able to automatically find, or discover required web service. A software must automatically determines its purpose based on its semantic description.
- **Invocation**
  An application must automatically invokes and executes a web service. The application must to know how to interact with a service, the web service provides detailed list of what should be done to execute the service.
- **Composition**
  To complete the certain objective, the application must be able to combine a number of web services. The services have to interoperate with each other in correct way to get a valid output solution.
- **Monitoring**
  Software must be able to monitor service properties to control the workflow of web services.

1.2 Web Services description models

In order to allow machines to manipulate services, we have to add semantics to these service descriptions. Typical examples of manipulations are service discovery, invocation, composition, and monitoring which were discovered before. This chapter will discuss Web Services Description Models such as WSMO, WSDL, SAWSDL, hRESTS, MicroWSMO, Swagger, APIS JSON, WSMO-lite and Linked Web APIs.

1.2.1 WSDL

WSDL[9] is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information [19]. It was developed by Microsoft and IBM.

The WSDL provides:

- A description of a Web Service.
- The communication mechanisms it understands.
- Description of operations it can perform.

[9]Web Services Description Language

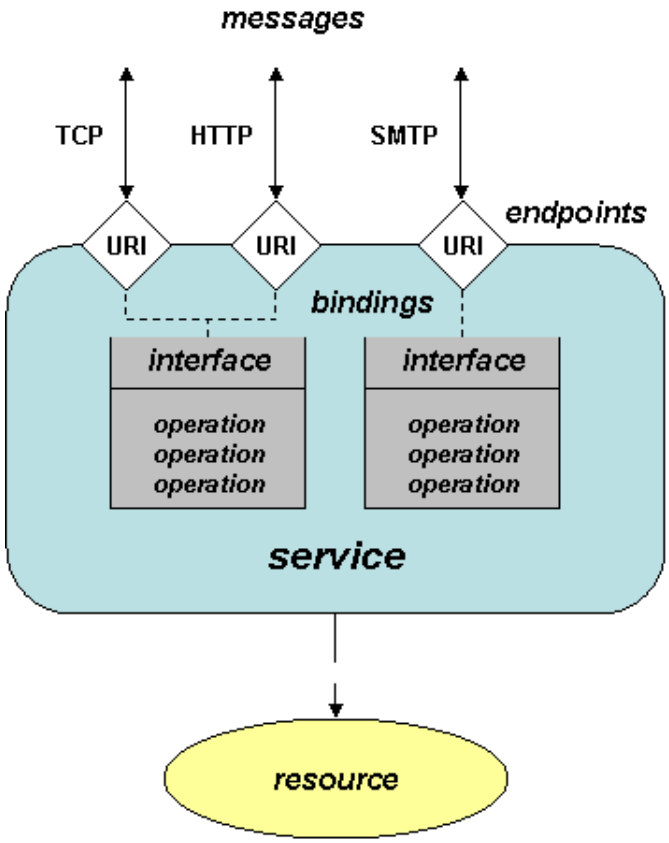


Figure 1.3: WSDL schema [3]

- Description of message's data structure.

The WSDL is often used to describe SOAP protocol using XML Schema languages. Moreover, from version 2.0 it can be used to describe Representational State Transfer (REST) Web Services.

WSDL 2.0 document uses the following elements in the definition of services [20]:

- Service - contains a set of functions that have been exposed to the Web-based protocols.
- Endpoint - is connection point or address to a Web Service.
- Binding - specifies the interface and defines the SOAP binding style and transport.
- Interface - defines a Web Service, the operations, and the messages.

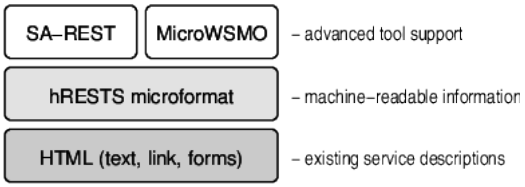


Figure 1.4: hRESTS microformat

- Operation - defines the SOAP actions and the approach of how the message is encoded.
- Types - describes the data type.

As were mentioned above, WSDL is an XML-based interface definition language that is used for describing the functionality offered by Web service. But the WSDL has no capability to describe Web Service in a semantic manner. Thus SAWSDL model was created.

1.2.2 SAWSDL

SAWSDL[10] is a Semantic Annotations for WSDL and XML[11] Schema.[21] The SAWSDL is a set of extension attributes for the WSDL and XML definition language that describes WSDL components semantically.

The primary purpose of SAWSDL is to describe Web Services using semantic annotation. The annotation helps to distinguish the meanings of Web Services description during automatic discovery and composition of Web Services.

1.2.3 hRESTS

hRESTS[12] is a microformat for machine-readable descriptions of Web APIs, backed by a simple service model. The hRESTS microformat describes RESTful APIs and most important aspects of it including operations, inputs, and outputs.[22]

hRESTS microformat is a service description layer, it helps to markup unstructured HTML documents, describing Web APIs, and turns it into a machine-readable format for further processing, see figure 1.4. The microformat identifies key pieces of information that are already present in the documentation, effectively creating an analog of WSDL.

The machine-readable hRESTS microformat can provide tool support or semi-automation in future development.

[10]Semantic Annotation for Web Services Description Language
[11]Extensible Markup Language
[12]HTML for RESTful Services

1.2.4 MicroWSMO

As were mentioned above, hRESTS microformat structures information from HTML pages of RESTful web services, so the information becomes amenable for machine processing.

Thus hRESTS forms the basis for further extensions, such as MicroWSMO.

MicroWSMO is a semantic annotation mechanism for RESTful Web services, based on a microformat called hRESTS (HTML for RESTful Services) for machine-readable descriptions of Web APIs, and backed by a simple service model [23].

It helps to structure and describe RESTful APIs semantically, making it more understandable by machines.

Because of similarity of hRESTS and WSDL, the MicroWSMO format adopts SAWSDL properties to add a semantic annotation.

1.2.5 WSMO

WSMO [13] is a conceptual framework and formal language for semantically describing all aspects of Web Services to gain an automation of discovery, services composition, and invocation. The goal of WSMO is to achieve dynamic, scalable and cost-effective infrastructure in business and public administration [24].

The WSMO has four main concepts:

- **Goals**
  Represent user desires.
- **Ontologies**
  Provide the terminology used by other WSMO elements to describe the relevant aspects of the domains.
- **Mediators**
  Mediators provide interoperabilities between different WSMO elements. It is core concept to resolve compatibility problems with data, processes, and protocols. It helps to communicate between Web Services and to combine them.
- **Web Services**
  The semantic description of Web Services domain, describe functional and behavioral aspects. Description of Web Services contains information about services capabilities and internal working.

But WSMO considered as heavyweight solutions with adoption the top-down approach to modeling of services, which is not suitable approach with the current industrial development of SOA technology, such as WSDL and

[13] Web Services Modeling Ontology

REST. Thus it is better to use bottom-up approaches to service modeling such as SAWSDL, MicroWSMO, and WSMO-Lite.

1.2.6 WSMO-Lite

WSMO-Lite is a lightweight semantic description for services on the web. It is the next evolutionary step after SAWSDL, filling the SAWSDL annotations with concrete semantic service descriptions.[22]

The primary goal of WSMO-Lite is to provide intelligent service integration.

WSMO-Lite advantages:

- It is very lightweight and defines a small vocabulary.
- The basic vocabulary is defined in RDFS with very limited reasoning requirements, but it can easily accommodate more expressive languages.
- WSMO-Lite builds on WSDL. The WSDL is well-known technology for many developers.
- Can be tightly scoped to RESTful web services.

WSMO-Lite ontology provides a possibility to describe Web Services. Furthermore, the ontology broadens the applicability of SAWSDL to RESTful services through hRESTS and MicroWSMO.

1.2.7 Swagger

Swagger is the world's largest framework of API developer tools for the OpenAPI Specification, enabling development across the entire API lifecycle, from design and documentation, to test and deployment [25].

The goal of OpenAPI Specification is to define a standard, language-agnostic interface to REST APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection.

1.2.8 APIS JSON

APIS JSON is a machine readable approach that API providers can use to describe their API operations [26].

The format is designed for public deployment and helps to consume Web APIs by automated software tools (robots).

The APIS JSON specification is a pre-existing set of rules and conventions which helps API developers to design and build APIs. The specification describes APIs in JSON data-interchange format.

1.2.9 Linked Web APIs

Previous sections describe several Web Services Description models which can be used by Web API providers to make it more self-descriptive and provide capabilities for easy use by third parties.

Linked Web APIs is a dataset providing information about Web APIs and mashups which utilize Web APIs in compositions [27].

The dataset provides descriptions of Web Services and RESTful APIs using a semantic mechanism (WSMO-Lite, SAWSDL, MicroWSMO) described in previous sections. The dataset represented as Open Linked Data and has 5-star classification:

- Make your stuff available on the Web (whatever format) under an open license
- Make it available as structured data (e.g., Excel instead of image scan of a table)
- Make it available in a non-proprietary open format (e.g., CSV instead of Excel)
- Use URIs to denote things, so that people can point at your stuff
- Link your data to other data to provide context

Thus based on the classification above, Linked Web APIs is a dataset of Web API and mashup descriptions that can be referenced, combined and reused for further easy service use by humans and machines.

1.2.10 Summary

Figure 1.5 shows WSMO-Lite ontology position relatively SAWSDL, MicroWSMO, WSDL, and hRESTS models.

WSDL is a standard format for Web Services descriptions. SAWSDL extends WSDL by a semantic annotation that can be pointed to WSMO-Lites ontology.

hRESTS is a microformat for describing RESTful APIs applying on HTML documents; it is analog of WSDL format. hRESTS format is extended by MicroWSMO format, which adds semantic annotations to the origin. In other words, MicroWSMO is a realization of SAWSDL annotations over hRESTS. MicroWSMO annotations can be pointed to WSMO-Lite ontology as well as SAWSDL.

Thus WSMO-Lite specifies a simple vocabulary that fits in SAWSDL and MicroWSMO annotations and can be used to describe Web Services as well as RESTful APIs.

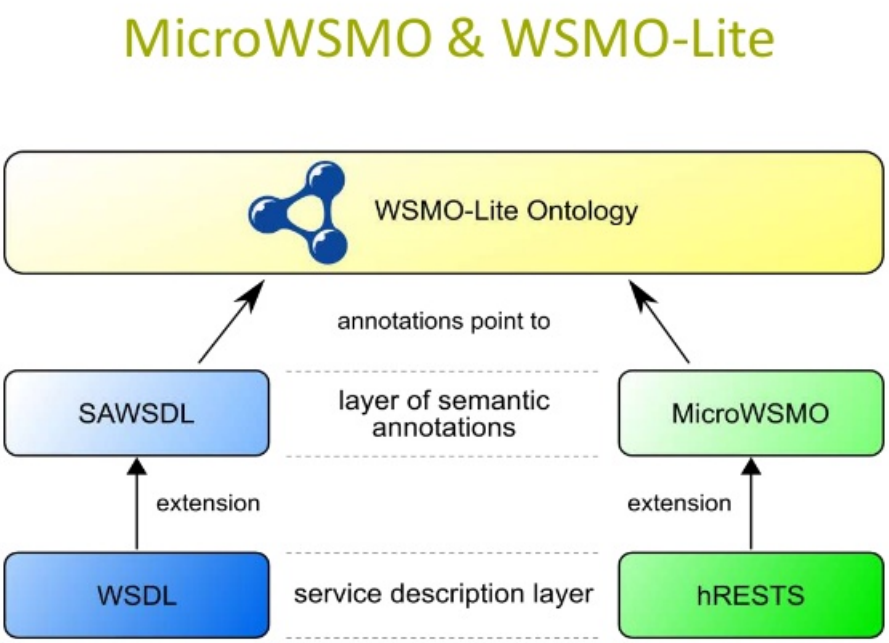


Figure 1.5: Web Services Description Models [4]

It is also worth noting that Swagger is a tool for developing RESTful APIs using OpenAPI Specification. It is a tool for describing RESTful APIs operations, inputs, outputs, etc. Whereas APIS JSON specification was created to impose a common structure on APIs.

To efficient Web API development flow, developers should use the combination of OpenAPI Specification, and APIS JSON approaches.

1.3 Web API data sources

This section describes Web APIs data sources from which information will be collected, extracted and processed. Based on APIs research, were collected the most valuable Web APIs directories:

- ProgrammableWeb[14]
- Apis.io[15]

[14] http://www.programmableweb.com
[15] http://apis.io

- Apis.guru[16]
- Api-for-that[17]
- Exicon[18]

1.3.1 Programmable Web

ProgrammableWeb is a leading source of news and information about Internet-based application programming interfaces (APIs) and the largest searchable Web API, mashup directory.[28]

It is the primary source of Web APIs and mashups in this work. ProgrammableWeb is the central repository of Web APIs descriptions. The ProgrammableWeb platform offers the ability to API's providers to publish and share their information about API with the public. The platform also provides the capabilities to the developers to explore APIs for their specific needs.

The website contains over 17,000 Web APIs and over 7,000 mashups. It provides API and mashup information in a human-readable interface, i.e. in HTML format.

1.3.2 Apis.io

APIs.io is an API searchable directory. The directory provides API information in a human-readable interface (HTML) and JSON[19], more precisely in APIs.json format. They have more than 1,000 APIs in their database.

APIs.io directory grants native API, which provides structured Web APIs information in JSON format.

1.3.3 Apis.guru

Apis.guru is an open-source API directory. Their goal is to create a machine-readable Wikipedia for REST APIs with the following principles:

- Open source, community driven project.
- Only publicly available APIs
- Anyone can add or change an API, not only API owners.
- All data can be accessed through a REST API.

[16] `https://apis.guru/openapi-directory/`
[17] `http://www.apiforthat.com`
[18] `https://app.exiconglobal.com/api-dir/`
[19] JavaScript Object Notation

It has more than 500 APIs, and it grants a native API, which provides data in structured JSON format.

Also, Apis.guru directory contains links to additional Web Services descriptions in Swagger format. The Swagger format is described next in section 1.2.7.

1.3.4 Api-for-that

API-For-That is a hand-curated API directory. Organized into about 20 industry categories, API-For-That catalogs an estimated 500+ API profiles with links to documentation, a provider home page, and a short description for each API.

The directory provides its APIs only in human-readable format.

1.3.5 Exicon

Exicon is an application management service, provides a directory of 1,900 APIs sortable by industry type and category.

The directory provides API descriptions in the human-readable interface, in HTML format.

Chapter 2

Analysis and Design

2.1 Requirements

The primary goal of the thesis is to extract required information from Web Services domain, normalize and transform it into RDF format. The program will be designed as a console application and must be able to execute automatically in the background as daemon application. Thus the following requirements are based on those facts.

2.1.1 Functional requirements

- The system is controlled by using a command line and configuration file.
- The system gives a choice to a user which action will be executed:
  1. Extraction
  2. Transformation
- The system gives a choice to a user which data source will be extracted.
- The system gives a choice to a user which data source will be transformed.
- The user can modify transformation rules independently for each data source.

2.1.2 Non-functional requirements

- The application must work on Linux operation system.
- For comfortably use, the application must be controlled using a command line.

- The application must be able to work uninterrupted during the whole process.
- The system must be divided into two parts:
  1. Extracting information from data sources.
  2. Transforming information into RDF format.
- The application must be able to bypass blocking by data sources.

2.2 Vocabularies

On the Semantic Web, vocabularies define the concepts and relationships (also referred to as "terms") used to describe and represent an area of concern. Vocabularies are used to classify the terms that can be used in a particular application, characterize possible relationships, and define possible constraints on using those terms.

Some of the vocabularies do not cover the full scope of the domain. Thus it requires the combination of vocabularies to describe the whole particular domain such as Web Services. Besides, creating of own vocabulary usually required.

2.2.1 Dublin Core

Dublin core is a small set of terms to describe web resources or other physical resources. Dublin Core vocabulary is used to describe basic information about Web Services such as Web API's title, created date, creator, publisher, and description.

2.2.2 XSD

XSD[20], a recommendation of the World Wide Web Consortium, specifies how to formally describe the elements in an Extensible Markup Language (XML) document.[29] XSD can be used to express a set of rules to which an XML document must be valid.

XSD vocabulary is used to describe data type of objects.

2.2.3 FOAF

FOAF[21] is an RDF based schema to describe persons, their activities and their relations to other people and object. It is a descriptive vocabulary expressed using the Resource Description Framework and OWL[22].

[20] XML Schema Definition
[21] friend of a friend
[22] Web Ontology Language

Also, it can be used to describe different areas such as Web Services. The FOAF vocabulary can describe categorization of Web Services and basic description, such as name and homepage.

2.2.4 RDFS

RDFS (Resource Description Framework Schema) is a set of classes with certain properties using the RDF extensible knowledge representation data model, providing basic elements for the description of ontologies, otherwise called RDF vocabularies, intended to structure RDF resources. The rdfs:label is used to describe labels of resources.

2.2.5 VCARD

VCARD vocabulary is used for the description of people and organizations. For example vcard:email term is used to describe APIs contact email.

2.2.6 PROV

Provenance is information about entities, activities, and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability or trustworthiness. The PROV Family of Documents defines a model, corresponding serializations, and other supporting definitions to enable the inter-operable interchange of provenance information in heterogeneous environments such as the Web.[29]

The PROV ontology is used in the application to define a relationship between providers, APIs and mashups entities. The ontology is used to define mashups and APIs association with providers and to determine what APIs are used in mashups.

2.2.7 Linked Web APIs

Linked Web APIs is a custom vocabulary used for describing Web Services domain. It was created specifically to help to describe Web Services which were not completely described by others vocabularies.

2.3 Use cases

The system was designed as a console application. All interaction with users is made through the command line with predefined console commands. The list of commands can be found in appendix section D.3

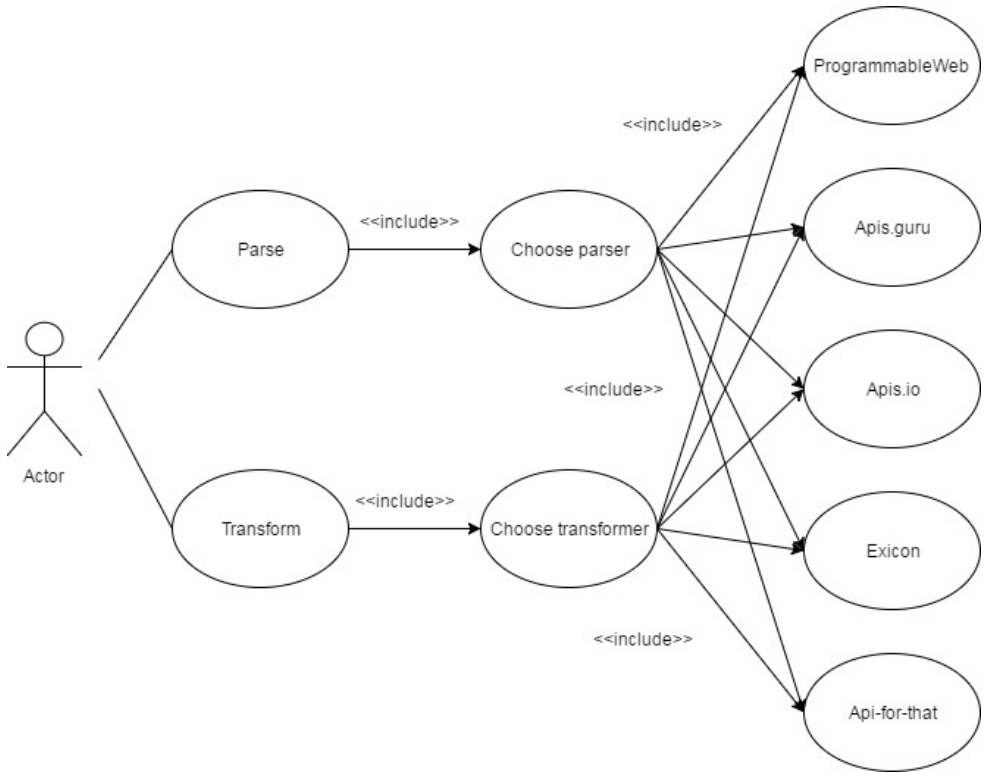


Figure 2.1: User interaction use cases

As you can see on above image, a user can launch the application for two purposes: extracting API directories and transforming extracted APIs to RDF format. No matter which action is chosen, the application must be specified with what data source it should work.

2.4 The architecture

Before implementing the application, it needs to be designed in well-structured architecture. As were mentioned before, the primary purpose of the thesis is to provide a solution for transforming data from multiple heterogeneous sources into semantic representation using RDF framework. The object of the application is data. Thus were decided to separate the application based on the data concept.

The data concept is divided the application into three parts: data collection, data normalization and data transformation. The controller orchestrates the whole application process from data collection to data integration. Thus thanks to controller component these processes become more independent.

2.4.1 Data collection

Data collection is one of the most important parts of the application. Data collection part is based on using an extraction component. An extraction component consists of a crawler part and a parser part.

When data collection process starts, the application begins to crawl a data source and collects all possible links referenced to Web APIs. The data collection process runs on the following data sources:

- **ProgrammableWeb APIs**
  Crawler entry point: `http://www.programmableweb.com/category/all/apis?page=1&deadpool=1`
- **Apis.io**
  Crawler entry point: `http://apis.io/api/apis?limit=99999`
- **Apis.guru**
  Crawler entry point: `https://api.apis.guru/v2/list.json`
- **Api-for-that**
  Crawler entry point: `http://www.apiforthat.com/apis?page=1`
- **Exicon**
  Crawler entry point: `https://app.exiconglobal.com/apis?terms=&companyId=&show=50&page=1&status=enable`
- **PrgrammableWeb Mashups**
  Crawler entry point: `http://www.programmableweb.com/category/all/mashups?page=1&deadpool=1`

To start data collection process, every crawler must have an entry point, a point from where the crawler starts a crawling process.

After successful link extraction, the application executes parsing process. The parsing process goes through each link and extracts Web APIs information using extraction rules specifically for this data source.

Crawling and parsing processes can extract information from HTML and JSON formats using different extraction techniques. Parsing of HTML data is based on DOM traversal technique implemented in open-source JSOUP library. Parsing of JSON data is based on converting JSON string into a tree structure of Java objects implemented in open-source GSON library.

As a result of data collection process, the application gets data in JSON format ready for further processing.

To be more clear we introduce an example of how extracting Web API from ProgrammableWeb directory works. First, the application identify a data source from which an extraction process will extract a data, in our case it is ProgrammableWeb. Then, the application runs crawler process to get all possible links referenced to Web APIs. The crawler process starts crawling from the entry point that was define for each API directory, in our case it is `http:`

//www.programmableweb.com/category/all/apis?page=1&deadpool=1. As a result the application gets a list of Web API links. After that, the application executes parsing process for each link, in our case the parsing process parses pages using JSOUP library and saving extracted information into JSON objects. Thus we get Web API descriptions in a JSON format, below is an example of extracted Google Maps API description:


```
[{
    "created": "2005-12-05",
    "url": "https://www.programmableweb.com/api/google-maps",
    "name": "Google Maps API",
    "description": "The Google Maps API allow for the embedding of
     Google Maps onto web pages of outside developers, using a
    simple JavaScript interface or a Flash interface. It is
    designed to work on both mobile devices as well as traditional
     desktop browser applications. The API includes language
    localization for over 50 languages, region localization and
    geocoding, and has mechanisms for enterprise developers who
    want to utilize the Google Maps API within an intranet. The
    API HTTP services can be accessed over a secure (HTTPS)
    connection by Google Maps API Premier customers.",
    "isDeprecated": false,
    "logo_url": "https://www.programmableweb.com/sites/default/
    files/styles/article_profile_150x150/public/apis/at22.png?itok
    \u003dAM2D9JYC",
    "endpoint": "https://www.google.com/maps/embed/v1/",
    "homepage": "https://developers.google.com/maps/",
    "first_category": [
      "Mapping"
    ],
    "second_category": [
      "Viewer"
    ],
    "api_provider_name": "Google",
    "api_provider": "http://www.google.com",
    "is_ssl_support": false,
    "api_forum": "http://groups-beta.google.com/group/Google-Maps-
    API?pli\u003d1",
    "twitter_url": "http://twitter.com/googlemapsapi",
    "console_url": "http://code.google.com/apis/ajax/playground/",
    "auth_model": [
      "API Key"
    ],
    "terms_of_service_url": "http://code.google.com/apis/maps/
    terms.html",
    "scope": "Single purpose API",
    "is_device_specific": false,
    "docs_home_page": [
      "https://developers.google.com/maps/"
    ],
    "architectural_style": [
      "REST"
    ],
```

```
    "request_formats": [
      "KML",
      "URI Query String/CRUD",
      "XML",
      "VML",
      "JavaScript"
    ],
    "response_formats": [
      "XML",
      "JSON",
      "KML"
    ],
    "is_unofficial_api": false,
    "is_hypermedia_api": true,
    "is_restricted_access": false
}]
```

Listing 2.1: Extracted Google Maps API description

2.4.2 Normalization

Data normalization is a connecting element between data collection and data transformation. Sometimes it is not possible to transform collected data into specific format without data normalization. Data normalization is a way to convert/normalize input data to the desired data structure. It helps to convert data based on predefined data format requirements.

Thus the data, mapped to subject or object part of RDF triple "subject-predicate-object", needs to be normalized.

To accomplish this, the normalization process can be separated into two parts: data normalization and data structure normalization.

2.4.2.1 Data normalization

The first part helps to normalize data format, which can be used then in the transformation process. It recursively goes through the whole JSON object and applies normalization for those fields which were defined.

The data normalization process applies the following rules for each field:

1. Trims spaces at the beginning and end of a string.
2. Lowercase string.
3. Identify digits at the beginning of the string and adds "the-" preposition to the beginning of the string.
4. Replace all symbols which are not [a-b] or [0-9] symbols to "-" symbol.
5. Trims "-" symbol at the beginning and end of a string.

6. Replaces sequences of ”-” symbol to ”-” symbol.

As a result of data collection part, the normalization process receives a Web API description in JSON format, and normalizes it in appropriate data format to be ready for transformation process. More precisely, it creates a normalized copy of certain fields and add it to JSON object.

For example, after data normalization, the following normalized data will be added to JSON object which contains Google Maps API description:

```
{
    "name_normalized": "google-maps-api",
    "first_category_normalized": [
      "mapping"
    ],
    "second_category_normalized": [
      "viewer"
    ],
    "api_provider_normalized": "google.com",
    "auth_model_normalized": [
      "api-key"
    ],
    "architectural_style_normalized": [
      "rest"
    ],
    "request_formats_normalized": [
      "kml",
      "uri-query-string-crud",
      "xml",
      "vml",
      "javascript"
    ],
    "response_formats_normalized": [
      "xml",
      "json",
      "kml"
    ],
}
```

Listing 2.2: Google Maps API normalized fields

2.4.2.2 Data structure normalization

The second part helps to transform inappropriate data structure to appropriate.

The normalization process receives as input a list of fields that should be added to a data structure. The list contains strings which are consist of paths and fields. The method recursively goes to a particular path in JSON object and adds a field with the empty value.

As a result, the method will recursively bypass JSON object and add all required fields in appropriate places defined by paths.

As a result of data structure normalization, the application gets structured data in JSON format ready for further transformation.

2.4.3 Data transformation

Data transformation is the second most important part of the application. The application executes the transformation process as soon as it gets an extracted, normalized information from a parser. The application then transforms processed data into RDF format using predefined RML rules. Each data source has its own, specific RML rules, which help to map data from JSON format to RDF.

Below is an fragment of RML rules for ProgrammableWeb API directory for transforming data from JSON to RDF format:

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix semapi: <http://linked-web-apis.fit.cvut.cz/ns/core#> .
@prefix pw: <http://linked-web-apis.fit.cvut.cz/resource/pw/> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix lso: <http://linked-web-apis.fit.cvut.cz/ns/core#> .

<#PWebMapping>

  rml:logicalSource [
    rml:source "data/programmableweb/pweb.json";
    rml:referenceFormulation ql:JSONPath;
    rml:iterator "$";
  ];

  rr:subjectMap [
    rr:template "http://linked-web-apis.fit.cvut.cz/resource/pw/{
    name_normalized}_api";
        rr:class semapi:WebAPI;
        rr:class prov:Entity;
  ];

    rr:predicateObjectMap [
    rr:predicate dcterms:created;
    rr:objectMap
    [
      rml:reference "created";
            rr:datatype xsd:date
    ]
  ];
```

Listing 2.3: Fragment of ProgrammableWeb RML rules

When transformation process is finished, the application receives transformed data in RDF format:

```
pw:google-maps-api_api a prov:Entity , lso:WebAPI ;
```

```
dcterms:created "2005-12-05"^^xsd:date ;
```

Listing 2.4: ProgrammableWeb RDF

The full text of ProgrammableWeb RML rules can be found in the appendix section .

As a result of transformation, data becomes available on the web through an endpoint using SPARQL query language. The data then can be used by developers, end-users or consumed by third-party applications.

2.5 Domain model

The domain model is a model describing Web API area for particular data source. Each data source provider has its domain model, representing different data structures.

As shown in the image 2.2, the application gets Web API descriptions as HTML[23] and JSON documents, normalizes and transforms it into JSON representation.

After that, the JSON representation is transformed into a unified data model.

2.5.1 Unified data model

Unified data model is a data model which unifies data properties from different data models into one data model.

The unified data model is produced based on transformation process of datasets into RDF. As a result, the unified data model is represented in RDF format.

The data model is shown on figure 2.3 is a minimal model that covers the most valuable and frequency used Web APIs information. It does not cover all aspects of Web APIs description.

The unified data model is an ontology called Linked Web APIs provided by the supervisor [27]. Linked Web APIs data model was created based on information extracted from ProgrammableWeb data source. Considering that current application extracts data from several data sources, the Linked Web APIs data model has been modified and extended.

The Linked Web APIs ontology has three main classes: WebAPI class, Provider class, and Mashup class.

WebAPI class represents Web APIs, Mashup class describes mashup services which are constructed using Web APIs, Provider class describes WebAPI's and Mashup's providers.

The data model is shown on figure 2.3 is based on Dublin Core, XSD, FOAF, RDFS, VCARD, PROV and WSMO-Lite ontologies. Prefixes of ontologies are shown on figure 2.3 as well.

[23] Hypertext Markup Language

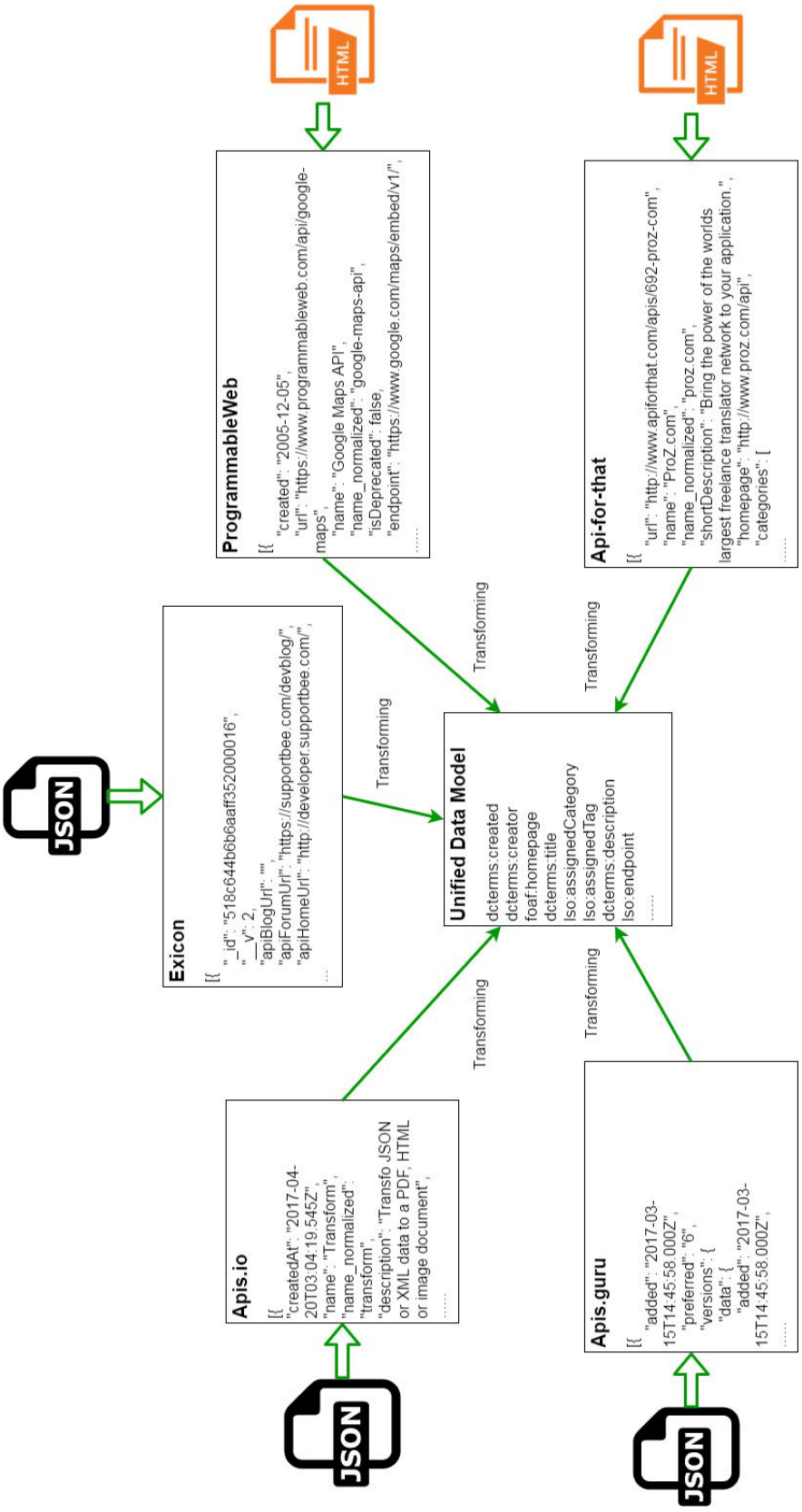


Figure 2.2: Domain model process

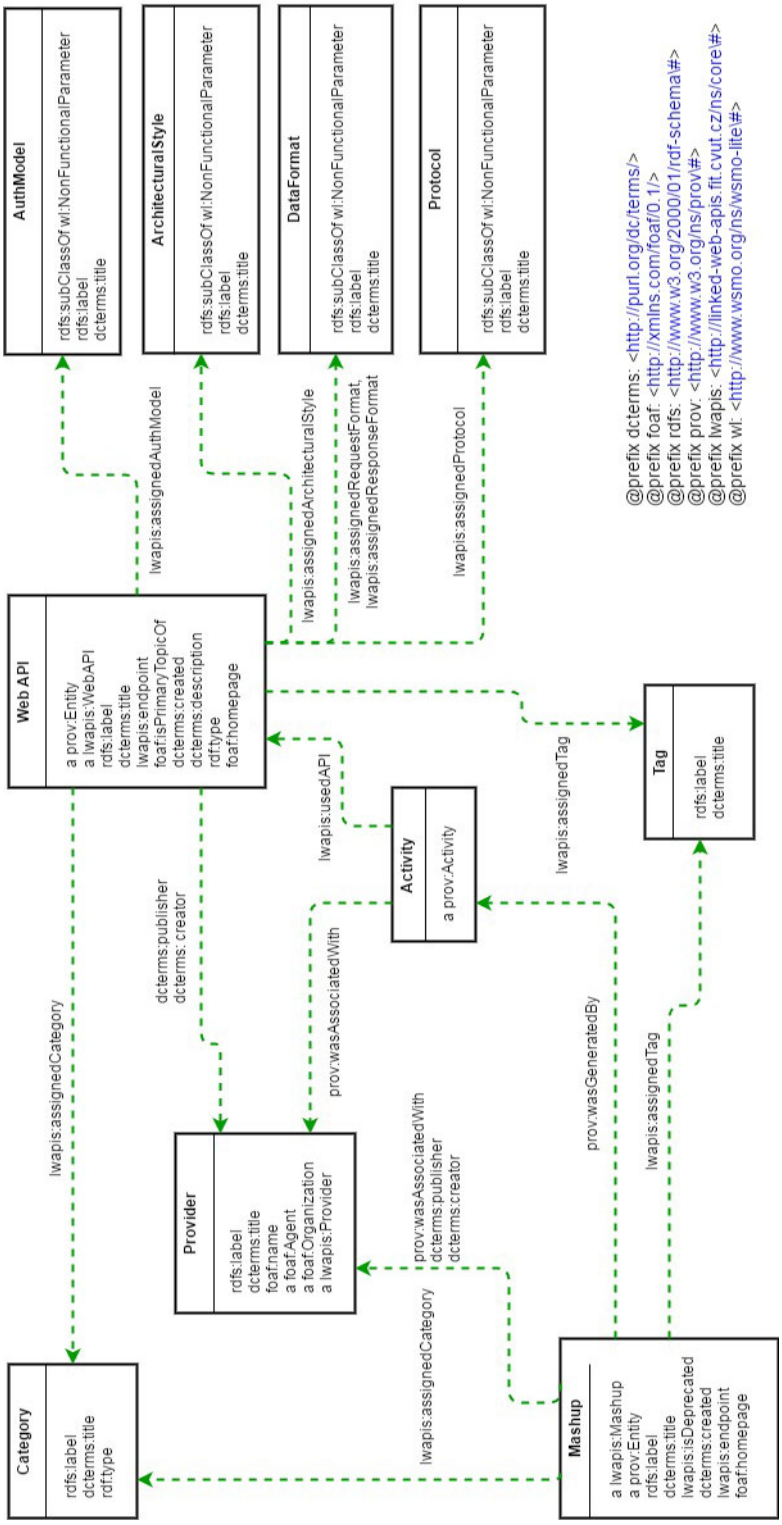


Figure 2.3: Extended Linked Web APIs data model

Chapter 3

Implementation

This chapter describes the implementation approaches and concepts and used technologies as well. Used technologies section answers the questions why particular technology is used and makes a short overview of it.

The concept section describes application's architecture, application's components, and implementation approaches.

3.1 Used technologies

The application is written in Java programming language. The reasons it is chosen are cross-platform, availability of required libraries and the fact that it is one of the most mature languages.

As were mentioned before, data sources are divided into two types: HTML and JSON. ProgrammableWeb, Api-for-that, and ProgrammableWeb mashups are data sources representing data in HTML documents; Apis.guru, Apis.io and Exicon are data sources representing data in JSON.

Thus JSOUP libraries are chosen as a parsing library for HTML documents, and Google's GSON library is chosen for storing extracted information in JSON format.

The whole project is controlled by Apache Maven, which is a build manager for Java projects.

Based on the supervisor recommendation, RML mapping library is chosen as a transformation library, which provides a mapping from JSON format to RDF format.

As a storage for RDF triples, OpenLink Virtuoso database engine is selected with the ability to provide an SPARQL endpoint.

More precisely, each technology is described below.

3.1.1 Java

Java is a programming language, which is an object-oriented and has developed much of its syntax from C. It was first developed by James Gosling at Sun Microsystems.

Java applications are usually compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM). Thus it can run on many different operating systems, and this makes Java platform-independent. It is possible because Java compiler turns code into Java bytecode instead of machine code. When Java application is executed, the JVM interprets the bytecode into machine code.

One of the major pros of Java is simplicity, object-oriented, distributed, mature and easy to learn.

Java language has been chosen as a programming language for this project based on its cross-platform, availability of required libraries, object-oriented approach, and maturity.

3.1.2 Apache Maven

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting, and documentation from a central piece of information.[30]

It addresses two aspects of building software: first, it describes how software is built, and second, it describes its dependencies.

The information about the project is stored in POM file, which is executed by Maven program with all operations specified. Apache Maven advantages list:

1. **Plugins**
   A huge selection of plugins is available in Maven repository.
2. **Automation**
   The software development process becomes less time consuming and more convenient.
3. **Testing**
   The ability to run tests as a part of your project lifecycle.
4. **Dependency management**
   Maven will resolve and manage project dependencies for you.
5. **Version managing**
   Maven helps you to manage dependency version.
6. **Standardization**
   Maven is a good way to standardize a project.

3.1.3 GSON

GSON is an open source Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. The advantages of using Gson library:

1. **Widely used**
   GSON is widely used by Google itself, and it is now used by a number of public companies. It is the most famous library for its purposes.

2. **Performance**
   GSON library has great performance. It can serialize a collection of 1.4 million objects and has deserialization limit for byte arrays and collection to over 11MB.

3. **Supported by Maven**
   GSON library can be found in Maven Central repository, which simplifies the process of installing the library.

4. **Completeness**
   GSON library is one of the most complete JSON library for Java.

3.1.4 JSOUP

JSOUP is an open source Java library for working with real-world HTML.[31] It consists of methods designed to extract and manipulate data, using Document Object Model (DOM), CSS, and jquery-like methods. Jsoup capabilities:

1. Parse HTML from URL, file, or string.

2. Find and extract data from HTML using CSS selectors and DOM traversal

3. Manipulate the HTML elements, attributes and text.

4. Implements the latest HTML5 specification.

5. XML support

JSOUP is the most popular and most widely used Java library for HTML parsing and extraction. It provides a convenient way to extract information from HTML documents.

Thus it is the most suitable library for information extraction from HTML documents.

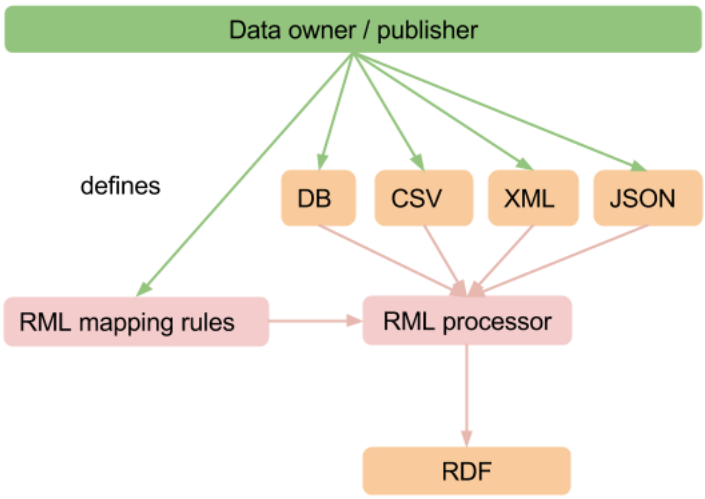


Figure 3.1: RML processing diagram [5]

3.1.5 RML

RML (RDF Mapping Language) is a mapping language defined to express customized mapping rules from heterogeneous data structures and serializations to the RDF data model.[5] It is based on and extending R2ML.

R2ML[24] is the W3C standard to express customized mappings from relational databases to RDF, so it helps to transform non-RDF data to raw RDF.

Thus RML is defined as a superset of the W3C-standardized mapping language, having the same syntax, aiming to extend its applicability and broaden its scope, adding support for data in other structured formats.

It is possible to transform data from several heterogeneous data structures (DB, CSV, TSV, XML, JSON) to RDF data model using RML mapping language, see figure 3.1.

The application uses RML library to transform data from JSON format into RDF data model using predefined rules written in R2ML language.

3.1.6 OpenLink Virtuoso

OpenLink Virtuoso is a middleware and database engine hybrid that combines the functionality of a traditional relational database management system, object-relational database (ORDBMS), RDF, XML, free-text, web application server and file server functionality in a single system.[22]

The database advantages:

[24]REWERSE Rule Markup Language

1. **Widely used**
   OpenLink Virtuoso is the most popular and widely used RDF triple-store database.

2. **Multi-model**
   Virtuoso is a multi-model hybrid-RDBMS that supports management of data represented as relational tables and/or property graphs.

3. **Supported languages**
   Virtuoso is supported by multiple programming languages, such as .NET, C, C#, C++, Java, Javascript, Perl, PHP, Python, Ruby, Visual Basic.

4. **Wide range of access methods**
   The database has a wide range of access methods, such as: HTTP API, OLE DB, WebDAV, ADO.NET, JDBC, ODBC.

5. **Format support**
   Support of different formats such as HTML, TEXT, TURTLE, RDF/XML, JSON, JSON-LD, XML.

Thus Openlink Virtuoso database has been chosen as a data storage based on above advantages.

3.2 Application architecture

The application architecture can be separated into four parts: collecting and extracting, normalizing, transforming and storing. The implementation of collecting part and extracting part is realized in parser component, the normalization part is implemented in normalization component, the transformation part is implemented in mapping (TriplesEngine) component, and data storing part is realized in StorageEngine component.

The components dependencies are shown on figure 3.2.

As we can see on figures 3.2 and 3.3, the application is controlled by central class, called controller. The controller component advocates as orchestration element. It controls the whole application process.

The execution of parsers is handled by controller's runParser method. The method receives parser object as a parameter and executes a crawling method and a parsing method consequentially. It was decided not to separate crawler and parser as independent components, because of simplicity of crawling part.

A crawl is a method in parser component which crawls data source documents using predefined link extraction rules. As a result (after crawling execution), the controller receives a queue of extracted links.

A parse is a method of parser component which extracts information from document referenced by specific URL. As were mentioned above, the controller

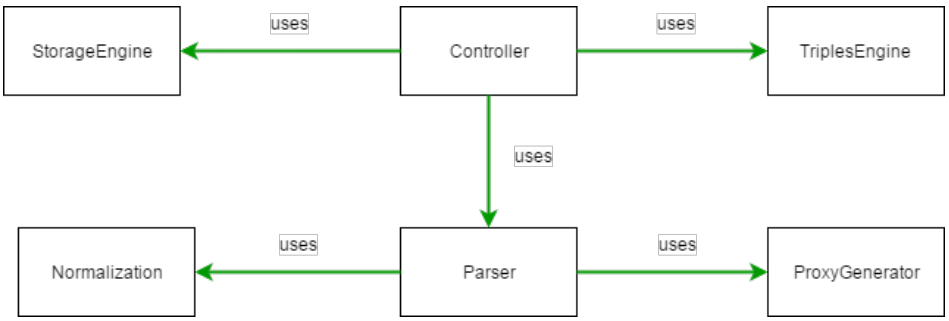


Figure 3.2: Components dependency diagram

component orchestrates the whole process. When it receives a list of extracted links from the crawler method, it runs the parse method for each link in a loop.

The parse method extracts information from documents pointed by URLs using predefined extraction rules and saves in JSON object. Then the parser, to get a normalized data, normalizes extracted data using normalization component.

The normalization component executes normalization process using rules that were defined in parser component. Thus every parser has its own normalization rules for particular data source. As a result of normalization, the controller receives JSON object with normalized data. After that, the controller saves it in an array of JSON objects.

The figure 3.3 shows an interaction process of application components.

After that, the application saves normalized JSON data in a file for further transformation. The transformation process is based on applying predefined RML rules.

Finally, the application runs RML rules on saved JSON file and generates RDF triples. As a result, the controller receives generated RDF triples as an RMLDataset object and saves it permanency using StorageEngine.

Each component will be discussed more precisely in next subsections:

- Controller 3.2.1
- Parser 3.2.2
- Normalization 3.2.3
- Mapping 3.2.4
- Storage 3.2.5

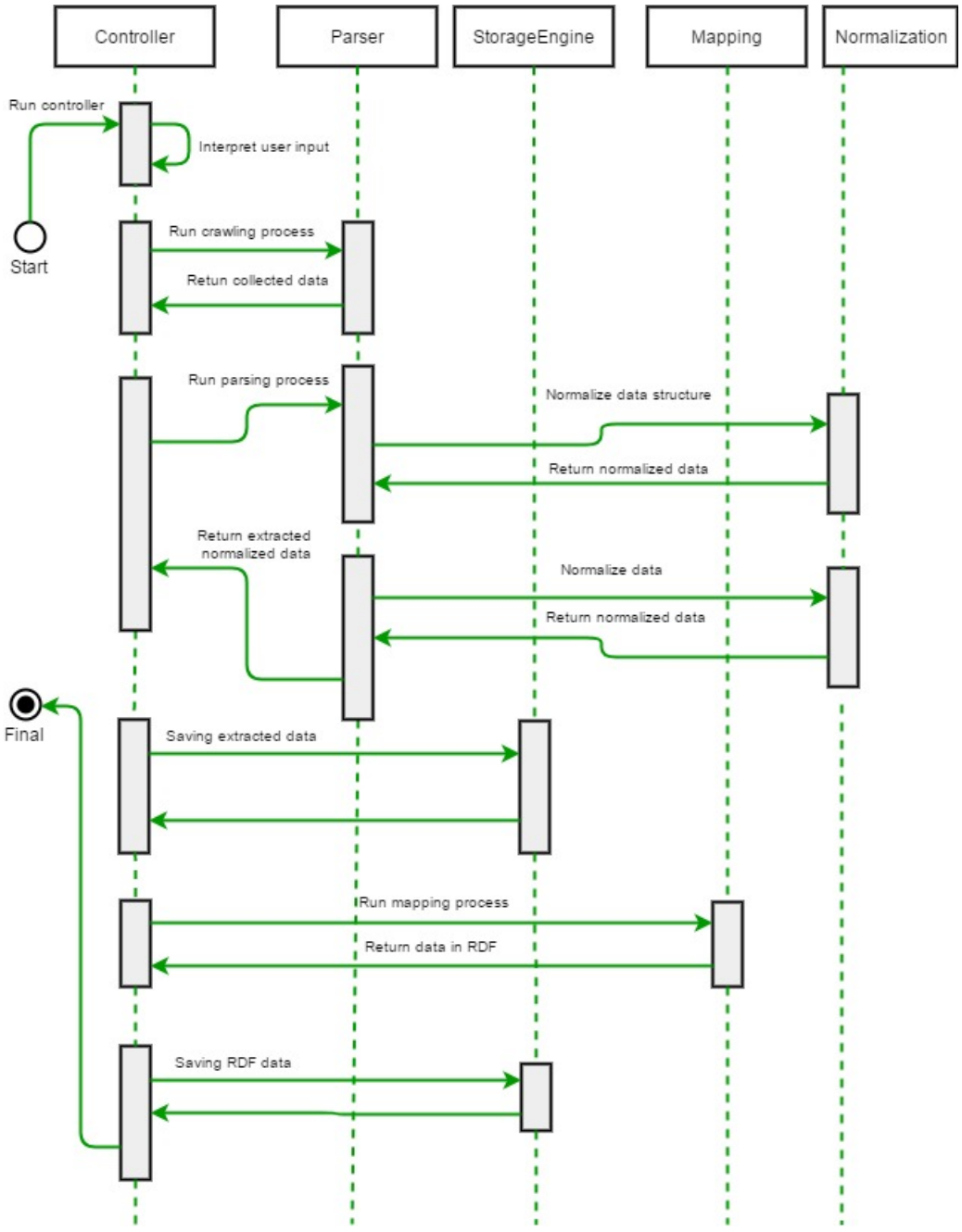


Figure 3.3: Components interaction diagram

3.2.1 Controller

As were mentioned above, the controller is a component which helps to orchestrate the application. The controller controls workflow of parser, mapping and integration components.

The controller's sequence of actions:

1. **Parser creating**
   The controller creates parser and passes to the runParser method for execution.

2. **Crawling**
   The runParser method executes crawling process and collects links from a data source.

3. **Parsing**
   After successful links collecting, the controller executes parsing method for Web API description extraction.

4. **Saving**
   Extracted Web API descriptions need to be saved for further processing.

5. **Mapping**
   The controller executes mapping process based on RML mapping language. The RML transforms extracted data using predefined RML rules from JSON format to RDF format.

6. **Storing**
   Finally, the controller saves RML mapping results to the appropriate data format.

3.2.2 Parser

The parser is a component responsible for collecting and extracting information from data sources.

The current application contains six parsers. Each parser is responsible for collecting and retrieving information from the following data sources: ProgrammableWeb APIs, ProgrammableWeb mashups, Apis.guru, Apis.io, Apifor-that and Exicon.

As you can see on the figure 3.4, PWEB_Mashup-Parser, PWEB_Parser, APIFOR_THAT_Parser are inherited from HtmlParser and APIS_Parser, EXICON_Parser, APIS_Parser are inherited from JsonParser.

HtmlParser and JsonParser are differ only by methods which are responsible for document extracting. HtmlParser extracts HTML documents, JsonParser extracts JSON documents from web.

HtmlParser and JsonParser have a common parent class called Parser. Parser is an abstract class, which defines parser's structure. Abstract Parser

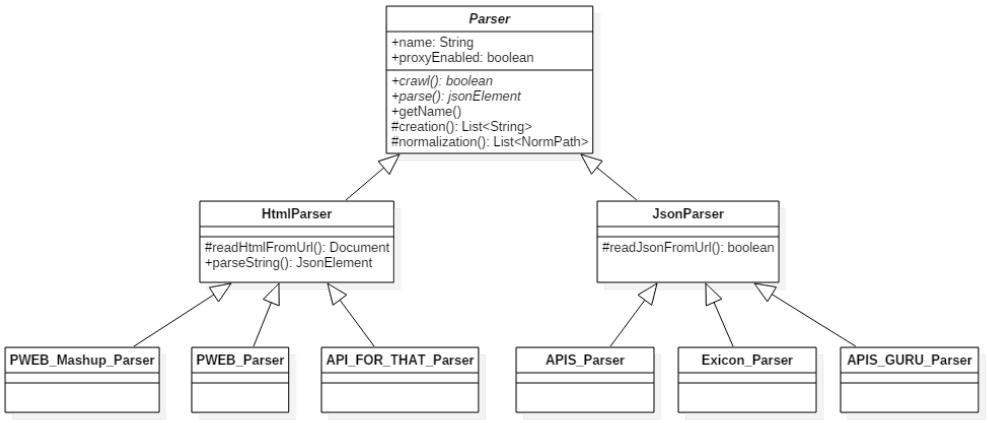


Figure 3.4: Parser's UML diagram

class contains two normalization methods. The first normalization method *creation()* is responsible for structure normalization and the second normalization method *normalization()* is responsible for data normalization.

Normalization methods define elements that should be normalized, i.e., all normalization logic is implemented in Normalization component.

Thus parsers are differ by *crawl()*, *parse()* and two normalization methods. If there is a need for implementing a parser for an additional data source, it can be easily added to the application.

3.2.3 Normalization

As were mentioned above in section 2.4.2 normalization process is a process which helps to normalize extracted data, i.e transform data to appropriate data format and data structure.

To accomplish this, the normalization component was created. The component has two methods. The first method normalizes data format, the second method normalizes data structure. More precisely, the approaches of those methods were discussed in section 2.4.2.

When extracted data is received from a parser, the application consequentially executes data format and data structure normalization methods. As a result, the component returns normalized data in JSON object.

3.2.4 Mapping

When normalization process is finished, the normalized data will be passed to mapping component. The component transforms normalized data into RDF format using RML library that was discussed before.

The transformation can be applied to different data source formats: XML, JSON, CSV, TSV. In our case, the transformation is applied for JSON data format.

The mapping component has the method called *applyMapping*, which executes the transformation process from JSON format to RDF format. The method applies mapping rules which were defined in an external file written in RML mapping language.

As a result, the mapping component returns RMLDataset object which contains generated RDF data.

3.2.5 Storage

The application consists of the storage component called TriplesEngine. The component is used for saving extracted information into JSON file. The JSON file is used then by RML library directly in predefined mapping rules.

The TriplesEngine component is also used for saving transformed RDF data into files in RDF format. Files can be imported into OpenLink Virtuoso database engine.

As a result, the transformed data becomes available as a file in RDF format.

3.2.6 Server configuration

3.2.6.1 Configuration

As were mentioned before the application is written in Java language and is platform independent. It can be run on Linux operations system as well as Windows operation system. The system must have pre-installed Maven automation tool to build the application.

To interpret and run the application, the system must have pre-installed Java JDK library and has at least 1GB of RAM to run the application properly. It is also recommended to have a Git control system installation for easier application deployment. Thus here are the system requirements:

1. Java JDK 1.4 and above.
2. Maven build automation tool.
3. Git control system.
4. Minimum of 1GB RAM.

The application is a console type application and can be run in command line. For installation and running instructions see Appendix section D.

The current application is run on Linux VPS machine on DigitalOcean hosting provider.

3.2.6.2 Application dependencies

Before deployment, the application must be built using Maven tool with the dependencies listed in pom.xml file. The POM file contains necessary information about the project, as well as dependencies used in the project.

```
<dependencies>

    <!-- https://mvnrepository.com/artifact/org.jsoup/jsoup -->
    <dependency>
        <groupId>org.jsoup</groupId>
        <artifactId>jsoup</artifactId>
        <version>1.10.2</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/com.google.code.gson/
    gson -->
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.8.0</version>
    </dependency>

    <!-- Self-added RML.io library -->
    <dependency>
        <groupId>rml</groupId>
        <artifactId>rml</artifactId>
        <version>0.3</version>
    </dependency>

    <!-- Apache language library -->
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.5</version>
    </dependency>

</dependencies>
```

Listing 3.1: Project dependencies

As mentioned on listing above, the project needs to have JSOUP, GSON, RML and Apache Language dependencies to be built properly.

3.2.7 Initialization

The first step of application running is initialization. Initialization is the assignment of initial values of application, which are contained in configuration file env.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <rdf_format>Turtle</rdf_format>
    <timeout>10000</timeout>
```

```
    <rdf_print>1</rdf_print>
    <qty>0</qty>
    <isProxy>false</isProxy>
    <proxy>data/proxy.txt</proxy>
    <delay>4000</delay>
    <requestNumber>10</requestNumber>
</configuration>
```

Listing 3.2: Configuration file env.xml

The configuration listed above initializes and configures the application.

Rdf_format attribute sets up an output format of RDF triples. The attribute takes the following values: Turtle, NTriples, Binary, JSONLD, N3, NQUADS, RDFA, RDFJSON, RDFXML, TRIG, TRIX.

Timeout attribute sets up an HTTP request/response timeout interval in milliseconds. If the application does not receive any response on page request from a server within an interval timeout, it discards the request and continues to work with the next page in the queue.

Rdf_print attribute can accept two value, 1 or 0. The attribute indicates the RDF output to console.

Qty is an attribute which is usually used for testing. The value of attribute means the number of pages to process. Value 0 means unlimited, the application will process all available pages.

Delay attribute indicates the delay between HTTP requests.

IsProxy attribute can accept two values, true or false. The attribute indicates whether proxy servers should be used for avoiding blocking problems. The path to the proxy list file can be found in the proxy attribute.

RequestNumber attribute indicates the amount of requests per proxy.

3.3 Deployment

The application is written in Java language using Maven build automation tool. The project uses several technologies such as GSON library, JSOUP library, RML mapping library and OpenLink Virtuoso. The whole deployment process can be separated into three steps:

- **Installing the application environment**
  At first, the application environment should be installed to build and run the application. The following technologies should be installed:

  1. Java JDK
  2. Maven build automation tool

- **Downloading the application**
  The application can be downloaded using git version control system

- **Adding RML dependency**
  RML library does not exist in Maven central repository. Therefore the library cannot be automatically downloaded and built using Maven. To add the RML library, it needs to be downloaded from the repository using git and manually added to Maven's pom.xml file.

- **Application building**
  Finally, the application needs to be built using Maven building tool. Maven automatically downloads all necessary libraries and builds the application.

The full deployment instructions can be found in appendix section D.

3.4 Implementation issues

This chapter describes several problems which were encountered during the application development.

3.4.1 HTML Parsing

The application works with five Web API directories and one mashup directory. Three of them represent data in HTML format. The application extracts data from HTML structures using DOM traversal.

When the structure of HTML documents is changed, the extraction process is violated and data cannot be extracted or extracted with a data integrity error.

To avoid this problem, HTML parsers need to be updated every time the DOM structure of API directories is changed.

Much better to implement a mechanism that will check DOM structure on each extraction. If the mechanism identifies a DOM structure changes, it notifies a user/developer. The user/developer can modify the application to avoid violations.

3.4.2 Blocking problems

The ProgrammableWeb data source is the most largest Web API directory. The directory has more than 20 thousands of APIs and mashups. The application needs to parse a huge amount of pages to get all data. Thus a number of HTTP requests are almost the same as a number of pages.

The ProgrammableWeb website has a preventing mechanism against frequent requests. When amount of requests from client's side reach some point, the site blocks further communication. To bypass the blocking, the application has two solutions, delay injection and proxy injection.

3.4.2.1 Delay injection solution

The blocking can be bypassed simply adding a delay between HTTP requests. The most suitable delay is 4000 milliseconds. The application will consider normal behavior and will not raise red flags.

3.4.2.2 Proxy injection solution

The second solution is a proxy injection. As we know, the website sets block after a certain number of requests for each client. The block is bounded to a client's IP address.

Despite this, the blocking can be bypassed by changing the client's IP address. It can be achieved by using third-party proxy servers. Thus, the ProgrammableWeb will see requests from different IP addresses and will not block any requests.

3.4.3 Transformation problems

One of the main transformation problems is an inappropriate data format. Extracted data is usually not in an appropriate data format.

The second problem is an inappropriate data structure. As were mentioned in previous chapters, the application stores all extracted data in JSON objects. Each object has its structure based on document's structure that was extracted. For example, one document has a particular set of attributes, another document has almost the same set of attributes, but without several attributes, thus data structures of extracted documents are different.

Without solving these problems, data cannot be transformed into semantic representation.

The solution to above problems is normalization. The normalization of data format should be handled by RML mapping library. But the library is still new and has no ability to do a data normalization.

Thus the normalization component was created for solving these problems. It was discussed in previous section 3.2.3.

3.4.4 Extraction errors

When the application processes a huge amount of pages, sometimes extraction problems occur. There are two reasons why it happens: bad client's internet connection and malfunctioning of a website (Web API directory).

To decrease the number of extraction errors, double extraction mechanism can be applied. The double extraction mechanism sends the additional request every time the application catches an exception on request.

By that, the double check mechanism can significantly decrease an amount of errors and increase the success rate of the extraction process, see the next chapter 4.

3.4.5 Identical APIs

One of the most difficult problems of API extraction for different data sources is identical APIs. Different data sources may have the same APIs in their databases. To make data more consistency and structured this problem needs to be avoided. However, this problem is not a trivial and required many efforts.

The current work does not solve this kind of problem. It is a topic for further exploring and research. But the problem can be solved if identifying mechanism will be invented and injected.

When the application extracts new API, the mechanism compares extracted API with APIs that already extracted. The mechanism calculates the similarity index between two APIs. If the similarity index is more than a predefined constant index, then very likely two APIs are similar and can be merged.

The predefined constant index is calculated experimentally based on a lot of processed data.

It is hard to say how the identification process should work in details because it is a topic for another work. But definitely, it is possible. Similar mechanisms are already applied in aggregation portals shopstyle.com [25], lyst.com[26] and many others.

[25] `http://shopstyle.com`
[26] `http://lyst.com`

Chapter 4

Validation and experiments

4.1 Coverage

Based on information provided by API directories we get the official information about amount of APIs each API directory provides. The information is provided in the table 4.1.

Thus API directories contain 21,736 APIs descriptions and 7,881 mashup descriptions and totally 29,617 APIs descriptions and mashups.

As were mentioned above the application extracts information from several data sources such as ProgrammableWeb, Apis.io, Apis.guru, Api-for-that and Exicon and transforms to Web API dataset. The amount of extracted API and mashup descriptions is represented in the table 4.1.

Thus the application has extracted 21,211 APIs descriptions and 7,852 mashup descriptions and 29,063 API and mashup descriptions.

Table 4.1: Provided and extracted APIs

Directories	**Provided by directories**	**Extracted by the application**
ProgrammableWeb	17,632 APIs	17,161 APIs
ProgrammableWeb Mashup	7,881 mashups	7,852 mashups
Apis.io	1,102 APIs	1,088 APIs
Apis.guru	482 APIs	460 APIs
Api-for-that	599 APIs	581 APIs
Exicon	1,921 APIs	1,921 APIs

Table 4.2: Extraction results

	Percent of extracted APIs	**Percent of missed APIs**
APIs	97.58%	2.42%
Mashups	99.63%	0.37%
Total	98.12%	1.87%

Based on the information provided in the table 4.1, we have calculated the percentage of extracted APIs and mashups information, as well as the percentage of missed APIs and mashups information. Calculated results are represented in the table 4.2.

It is worth noting that API information was obtained from API directories by the date of June 2017. As well as, the extraction process of Web APIs descriptions was carried out by the date of June 2017.

Based on results above, the success rate of extraction is between 97% and 100%. The success rate mostly depends on several factors: client's internet connection and proper functionality of a remote website.

Appearing of errors is a normal, it always happens on extraction process due to connection problems and errors on a server side. However, the success rate can be significantly increased.

More precisely this problem is discussed in section 3.4.4.

4.2 Quality

According to the 5-star classification system defined by Tim Berners-Lee, datasets can be classified using the following classification system:

1. Make your data available on the web under an open license.
2. Data in machine-readable structured format (e.g., Excel instead of image scan of a table).
3. Make it available in a non-proprietary open format, such as CSV, JSON, XML, etc.
4. Use URIs for identification.
5. Link your data to other LOD[27] datasets.

[27]Linked Open Data

The Linked Web APIs dataset credits four out of five stars: data provided by Linked Web APIs is open (first star), data is in machine-readable structured format (second star), it is available in non-proprietary RDF format (third star), dataset uses URIs for identification (fourth star).

4.2.1 Dataset Quality

The quality of dataset mainly depends on extraction process and quality of extracted information.

The extraction process has a straight impact on a quality of extracted data. To evaluate a quality of data, we have randomly created 100 RDF triples for each data source and make a data validation using Raptor RDF Syntax Library.

Raptor library goes through all RDF data in automatic mode and checks the syntax of each RDF triple.

As a result of Raptor execution were discovered a problem in an inappropriate data format in subject and object component of triples. Data format has the following inconsistencies:

1. Data format contains prohibited symbols such as ";", "—", "&" and a couple more.

2. Some of the data were written using not Latin symbols.

3. Some of the extracted data contain digits at the beginning of the subject, which is not allowed.

To eliminate the 1st and the 2nd problem, the normalization component was modified. Before these problems identification, the data normalization method solves normalization problems by replacing a set of prohibited symbols.

But this solution as we can see is not efficient because the set of prohibited symbols is not complete enough. To solve this problem, the normalization function was rewritten in opposite way: the function, instead of prohibited symbols, has a set of allowed symbols and generate normalized strings based on allowed symbols (strings contain only allowed symbols). Allowed symbols are a-z, A-Z and 0-9.

To fix the 3rd problem, an indication statement was added to normalization function. As a result, if the normalization function detects a string of digits at the beginning of the string, it prepends the preposition "the-".

Also, it is worth noting that data from API data sources in rare cases is not consistency. It has errors. This kind of problem cannot be solved; the data can be avoided and left as is.

4.2.2 Known Shortcomings

The extraction process of Web APIs is not always flawless because of dependency on HTML structure. The HTML structure in the course of time can be modified which breaks the process of extraction.

More precisely this problem is discussed in section 3.4.1.

Conclusion

The growing number of Web APIs has a great impact on the current internet and web development. The amount of Web APIs is significantly grown in last decades. At the moment there are several Web API directories which contain APIs descriptions. But there is still no common data model which unifies Web APIs data models.

Therefore the current work has been discovered and implemented. We have implemented the system for acquisition of data for Web APIs; the system transforms descriptions of Web APIs from several data models into the unified data model. The Linked Web APIs model was taken as the basis for unified data model and furthermore was extended.

As a result, we have accomplished:

- Analyzation of existing data models for its structure and data format. We have studied each data model structure and based on this information we have created automated collection of Web APIs descriptions.
- We have implemented transformation rules based on RML. Thus we have established mappings between data models (ProgrammableWeb, Apis.io, Apis.guru, Api-for-that, Exicon) and unified data model.
- Finally, based on collected Web APIs data and established mappings, we have transformed and integrated Web APIs data into Linked Web APIs data model.

As a result, Linked Web APIs data becomes available as Open Data.
The Open Data can be used then in several cases:

1. It can be consumed by end-user.
2. API providers have abilities to increase the visibility of their APIs.
3. The data can be used for further usage by developers or applications.

The application can be found on the Bitbucket repository on the following link: `https://bitbucket.org/m1ci/semantic-web-apis-hub`

Future work

The current thesis has a lot of work that can be implemented in future.

First of all extracted and transformed data is not interlinked with external semantic databases such as DBPedia and Freebase. When this task is accomplished, data becomes available as Linked Open Data. Thus, Data becomes linked, more valuable and efficient.

Moreover, if additional Web API data sources appear, the application can be easily extended by adding new parsers. The structure of the application is already developed, so the process of adding new parsers is easy enough.

To make the application more accurate and efficient, the problem with identical APIs should be solved. When the problem is solved, data generated by the application becomes more valuable and completeness.

As were mentioned above Semantic Web Services is very interesting and perspective area of web development. The primary goals of Semantic Web Services are automatic discovery, invocation, composition and monitoring Web Services. The current work can efficiency aid the process of API discovery. Using Linked Web APIs dataset, APIs become more understandable for machines and can be efficiently discovered and consumed.

Also, data provided by the application can be imported into Virtuoso database engine to be available through SPARQL endpoint. Then, the application can be integrated with SPARQL service description ontology, which provides a mechanism by which a client can discover information about SPARQL endpoint.

Bibliography

[1] Kearn, M. *Introduction to REST and .net Web API*. 2015. Available from: https://blogs.msdn.microsoft.com/martinkearn/2015/01/05/introduction-to-rest-and-net-web-api/

[2] GlobusSoft. *An Introduction to Web Scraping*. 2014. Available from: http://globussoft.com/an-introduction-to-web-scraping/

[3] Microsoft. *Understanding WSDL*. Available from: https://msdn.microsoft.com/en-us/library/ms996486.aspx

[4] Fensel, D.; Kopecky, J. *Semantic Web Services*. 2008. Available from: http://teaching-wiki.sti2.at/uploads/b/bf/SWS-10-SAWSDL.pdf

[5] Ghent University. *RDF Mapping Language (RML)*. Available from: http://rml.io/spec.html

[6] Wikipedia. *Application programming interface*. Available from: https://en.wikipedia.org/wiki/Application_programming_interface

[7] Wikipedia. *Web API*. Available from: https://en.wikipedia.org/wiki/Web_API

[8] *Mashup (web application hybrid)*. Available from: https://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)

[9] *Information extraction*. 2017. Available from: https://en.wikipedia.org/wiki/Information_extraction

[10] Olston, C.; Najork, M. *Web Crawling*. Stanford University, 2010.

[11] Wikipedia. *Web scraping*. Available from: https://en.wikipedia.org/wiki/Web_scraping

[12] Consortium, W. *Semantic Web*. W3 Consortium, https://www.w3.org/standards/semanticweb/.

[13] Klimek, J. *Introduction to Linked Data*. FIT, CVUT.

[14] Tim Berners-Lee. *5-star deployment scheme for Open Data*. 2010. Available from: http://5stardata.info/en/

[15] W3C. *RDF*. February 2014. Available from: https://www.w3.org/RDF/

[16] W3C. *SPARQL Query Language for RDF*. January 2008. Available from: https://www.w3.org/TR/rdf-sparql-query/

[17] Wikipedia. *Semantic Web service*. Available from: https://en.wikipedia.org/wiki/Semantic_Web_service

[18] Alesso, H. P. *Preparing for Semantic Web Services*. 2004.

[19] W3C. *Web Services Description Language (WSDL) 1.1*. Available from: https://www.w3.org/TR/wsdl

[20] W3C. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. 2007. Available from: https://www.w3.org/TR/wsdl20/

[21] W3C. *Semantic Annotations for WSDL and XML Schema*. Available from: https://www.w3.org/TR/sawsdl/

[22] Wikipedia. *Virtuoso Universal Server*. Available from: https://en.wikipedia.org/wiki/Virtuoso_Universal_Server

[23] Kopecky, J.; Vitvar, T.; et al. *MicroWSMO and hRESTS*. 2009. Available from: http://sweet.kmi.open.ac.uk/pub/microWSMO.pdf

[24] W3C. *Web Service Modeling Ontology (WSMO)*. 2005. Available from: https://www.w3.org/Submission/WSMO/

[25] Swagger. *Swagger*. Available from: http://swagger.io/

[26] *Apis.json*. Available from: http://apisjson.org/

[27] Dojchinovski, M.; Vitvar, T. *Linked Web APIs Dataset*. 2015.

[28] ProgrammableWeb. *About ProgrammableWeb*. Available from: https://www.programmableweb.com/about

[29] Wikipedia. *XML Schema (W3C)*. Available from: https://en.wikipedia.org/wiki/XML_Schema_(W3C)

[30] Apache. *Apache Maven Project*. Available from: https://maven.apache.org/

[31] Jsoup. *Jsoup: Java HTML Parser*. Available from: https://jsoup.org/

[32] W3C. *Linked Data*. July 2006. Available from: https://www.w3.org/DesignIssues/LinkedData.html

[33] APIs.io. *About APIs.io*. Available from: http://apis.io/about

[34] APIS.guru. *About APIS.guru*. Available from: https://apis.guru/openapi-directory/

[35] Wikipedia. *RDF Schema*. Available from: https://en.wikipedia.org/wiki/RDF_Schema

[36] W3C. *Vocabularies*. Available from: https://www.w3.org/standards/semanticweb/ontology

[37] OpenLink. *Virtuoso*. Available from: https://virtuoso.openlinksw.com/

[38] IBM. *Describe REST Web services with WSDL 2.0*. May 2008. Available from: https://www.ibm.com/developerworks/webservices/library/ws-restwsdl/

[39] The Dublin Core Metadata Initiative. *Dublin Core*. Available from: http://dublincore.org/documents/dces/

[40] W3C. *An Overview of the PROV Family of Documents*. Available from: https://www.w3.org/TR/prov-overview/

[41] Google. *GSON*. Available from: https://github.com/google/gson

[42] Kopecky, J.; Gomadam, K.; et al. *hRESTS: an HTML Microformat for Describing RESTfulWeb Services*. Innsbruck, 2008.

[43] Kopecky, J. *Evaluating WSMO-Lite*. STI Innsbruck.

[44] Vitvar, T.; Kopecky, J.; et al. *WSMO-Lite Annotations for Web Services*. 2008.

[45] Vitvar, T.; Kopecky, J.; et al. *WSMO-Lite and hRESTS: Lightweight Semantic Annotations for Web Services and RESTful APIs*.

[46] Serugendo, G. D. M. *Mashups*. University of Geneva.

[47] Labsky, M. *Information Extraction*. University of Economics, Prague.

[48] Arroyo, S.; Lara, R.; et al. *Semantic Aspects of Web Services*. 2003.

[49] Wikipedia. *Web Services Description Language*. Available from: https://en.wikipedia.org/wiki/Web_Services_Description_Language

Appendix A

Acronyms

WWW World Wide Web

RDF Resource Description Framework

HTML HyperText Markup Language

XML Extensible Markup Language

API Application programming interface

JSON JavaScript Object Notation

HTTP Hypertext Transfer Protocol

HTTPS Hyper Text Transfer Protocol Secure

WSMO Web Services Modeling Ontology

URL Uniform Resource Identifier

SQL Structured Query Language

DBMS Database Management System

CSV Comma-separated values

hRESTS HTML for RESTful Services

WSDL Web Services Description Language

SAWSDL Semantic Annotation for Web Services Description Language

XSD XML Schema Definition

OWL Web Ontology Language

LWAPIS Linked Web APIs

RML RDF Mapping Language

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `.gitignore` ..... file which tells git which files it should ignore
- `README.md` ..... description of the project for git repository
- `pom.xml` Maven XML file that contains information about the project and configuration details used by Maven to build the project
- `data` ..... the directory with RML rules and RDF output
- `src` ..... the directory of source codes
  - `semapi` ..... base application classes
  - `parsers` ..... application's parser classes (abstract and concrete)
    - `apiParsers` ..... application's concrete parser classes
- `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format

Appendix C

RML rules

Below is a full text of ProgrammableWeb RML rules, which apply to transform data from JSON to RDF format.

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix vcard: <http://www.w3.org/2006/vcard/ns#> .
@prefix doap: <http://usefulinc.com/ns/doap#> .
@prefix semapi: <http://linked-web-apis.fit.cvut.cz/ns/core#> .
@prefix pw: <http://linked-web-apis.fit.cvut.cz/resource/pw/> .
@prefix prov: <http://www.w3.org/ns/prov#> .

#API
<#PWebMapping>

  rml:logicalSource [
    rml:source "data/programmableweb/pweb.json";
    rml:referenceFormulation ql:JSONPath;
    rml:iterator "$";
  ];

  rr:subjectMap [
    rr:template "http://linked-web-apis.fit.cvut.cz/resource/pw/{
    name_normalized}_api";
        rr:class semapi:WebAPI;
        rr:class prov:Entity;
  ];

    rr:predicateObjectMap [
    rr:predicate dcterms:created;
    rr:objectMap
    [
      rml:reference "created";
            rr:datatype xsd:date
```

```
    ]
  ];

  rr:predicateObjectMap [
    rr:predicate foaf:homepage;
    rr:objectMap
    [
      rml:reference "url";
            rr:termType rr:IRI
    ]
  ];

  rr:predicateObjectMap [
    rr:predicate dcterms:title;
    rr:objectMap
    [
      rml:reference "name"
    ]
  ];

    rr:predicateObjectMap [
    rr:predicate rdfs:label;
    rr:objectMap
    [
      rml:reference "name"
    ]
  ];

  rr:predicateObjectMap [
    rr:predicate dcterms:description;
    rr:objectMap
    [
      rml:reference "description";
      rr:language "en-us"
    ]
  ];

  rr:predicateObjectMap [
    rr:predicate semapi:endpoint;
    rr:objectMap
    [
      rml:reference "endpoint";
            rr:termType rr:IRI
    ]
  ];

  rr:predicateObjectMap [
    rr:predicate foaf:isPrimaryTopicOf;
        rr:objectMap [
      rml:reference "homepage";
            rr:termType rr:IRI
        ]
  ];
```

```
rr:predicateObjectMap [
  rr:predicate vcard:Email;
  rr:objectMap
  [
    rml:reference "email_address";
  ]
];

rr:predicateObjectMap [
  rr:predicate semapi:docsUrl;
  rr:objectMap
  [
    rml:reference "docs_home_page";
          rr:termType rr:IRI
  ]
];

rr:predicateObjectMap [
  rr:predicate semapi:termsOfServiceUrl;
  rr:objectMap
  [
    rml:reference "terms_of_service_url";
          rr:termType rr:IRI
  ]
];

rr:predicateObjectMap [
  rr:predicate semapi:forumUrl;
  rr:objectMap
  [
    rml:reference "api_forum";
          rr:termType rr:IRI
  ]
];

rr:predicateObjectMap [
  rr:predicate semapi:supportUrl;
  rr:objectMap
  [
    rml:reference "support_url";
          rr:termType rr:IRI
  ]
];

rr:predicateObjectMap [
  rr:predicate semapi:sslSupport;
  rr:objectMap
  [
    rml:reference "is_ssl_support";
    rr:datatype xsd:boolean
  ]
];

rr:predicateObjectMap [
```

```
    rr:predicate semapi:deviceSpecific;
    rr:objectMap
    [
      rml:reference "is_device_specific";
      rr:datatype xsd:boolean
    ]
  ];

  rr:predicateObjectMap [
    rr:predicate semapi:unofficialApi;
    rr:objectMap
    [
      rml:reference "is_unofficial_api";
      rr:datatype xsd:boolean
    ]
  ];

  rr:predicateObjectMap [
    rr:predicate semapi:hypermediaApi;
    rr:objectMap
    [
      rml:reference "is_hypermedia_api";
      rr:datatype xsd:boolean
    ]
  ];

  rr:predicateObjectMap [
    rr:predicate semapi:restrictedAccess;
    rr:objectMap
    [
      rml:reference "is_restricted_access";
      rr:datatype xsd:boolean
    ]
  ];

  rr:predicateObjectMap [
    rr:predicate semapi:consoleUrl;
    rr:objectMap
    [
      rml:reference "console_url";
      rr:termType rr:IRI
    ]
  ];

    rr:predicateObjectMap [
    rr:predicate semapi:assignedCategory;
    rr:objectMap
    [
      rr:parentTriplesMap <#FirstCategoryMapping>
    ]
  ];

  rr:predicateObjectMap [
    rr:predicate semapi:assignedCategory;
```

```
    rr:objectMap
    [
      rr:parentTriplesMap <#SecondCategoryMapping>
    ]
  ];

  rr:predicateObjectMap [
    rr:predicate semapi:requestFormat;
    rr:objectMap
    [
      rr:parentTriplesMap <#RequestFormatMapping>
    ]
  ];

  rr:predicateObjectMap [
    rr:predicate semapi:responseFormat;
    rr:objectMap
    [
      rr:parentTriplesMap <#ResponseFormatMapping>
    ]
  ];

  rr:predicateObjectMap [
    rr:predicate semapi:authModel;
    rr:objectMap
    [
      rr:parentTriplesMap <#AuthModelMapping>
    ]
  ];

  rr:predicateObjectMap [
    rr:predicate semapi:architecturalStyle;
    rr:objectMap
    [
      rr:parentTriplesMap <#ArchitecturalStyleMapping>
    ]
  ];

  rr:predicateObjectMap [
    rr:predicate dcterms:creator;
    rr:objectMap
    [
      rr:parentTriplesMap <#ApiProviderMapping>
    ]
  ];

    rr:predicateObjectMap [
    rr:predicate dcterms:publisher;
    rr:objectMap
    [
      rr:parentTriplesMap <#ApiProviderMapping>
    ]
  ];
```

```
  rr:predicateObjectMap [
    rr:predicate semapi:isDeprecated;
    rr:objectMap
    [
      rml:reference "isDeprecated";
      rr:datatype xsd:boolean
    ]
  ].



#API provider
<#ApiProviderMapping>

  rml:logicalSource [
    rml:source "data/programmableweb/pweb.json" ;
    rml:referenceFormulation ql:JSONPath ;
    rml:iterator "$"
  ];

  rr:subjectMap [
    rr:template "http://linked-web-apis.fit.cvut.cz/resource/{$.
    api_provider_normalized}_provider";
        rr:class foaf:Organization;
        rr:class foaf:Agent;
        rr:class semapi:Provider
    ];

    rr:predicateObjectMap [
    rr:predicate foaf:name;
    rr:objectMap [
      rml:reference "$.api_provider_normalized"
      ]
    ];

    rr:predicateObjectMap [
    rr:predicate dcterms:title;
    rr:objectMap [
      rml:reference "$.api_provider_normalized"
    ]
  ];

    rr:predicateObjectMap [
      rr:predicate rdfs:label;
      rr:objectMap [
        rml:reference "$.api_provider_normalized"
      ]
    ] .


#First Category
<#FirstCategoryMapping>
```

```
  rml:logicalSource [
    rml:source "data/programmableweb/pweb.json" ;
    rml:referenceFormulation ql:JSONPath ;
    rml:iterator "$.first_category_normalized"
  ];

  rr:subjectMap [
    rr:template "http://linked-web-apis.fit.cvut.cz/resource/{$}
    _category";
         rr:class semapi:primaryCategory
         ];

  rr:predicateObjectMap [
    rr:predicate dcterms:title;
    rr:objectMap [
      rml:reference "$"
    ]
  ];

    rr:predicateObjectMap [
    rr:predicate rdfs:label;
    rr:objectMap [
      rml:reference "$"
    ]
  ] .


#Second Category
<#SecondCategoryMapping>

  rml:logicalSource [
    rml:source "data/programmableweb/pweb.json" ;
    rml:referenceFormulation ql:JSONPath ;
    rml:iterator "$.second_category_normalized"
  ];

  rr:subjectMap [
    rr:template "http://linked-web-apis.fit.cvut.cz/resource/{$}
    _category";
         rr:class semapi:secondaryCategory
  ];

  rr:predicateObjectMap [
    rr:predicate dcterms:title;
    rr:objectMap [
      rml:reference "$"
    ]
  ];

    rr:predicateObjectMap [
    rr:predicate rdfs:label;
    rr:objectMap [
      rml:reference "$"
    ]
```

```
  ] .


#Request Format
<#RequestFormatMapping>

  rml:logicalSource [
    rml:source "data/programmableweb/pweb.json" ;
    rml:referenceFormulation ql:JSONPath ;
    rml:iterator "$.request_formats_normalized"
  ];

  rr:subjectMap [
    rr:template "http://linked-web-apis.fit.cvut.cz/resource/{$}
    _format";
  ];

  rr:predicateObjectMap [
    rr:predicate dcterms:title;
    rr:objectMap [
      rml:reference "$"
    ]
  ];

    rr:predicateObjectMap [
    rr:predicate rdfs:label;
    rr:objectMap [
      rml:reference "$"
    ]
  ] .


#Response Format
<#ResponseFormatMapping>

  rml:logicalSource [
    rml:source "data/programmableweb/pweb.json" ;
    rml:referenceFormulation ql:JSONPath ;
    rml:iterator "$.response_formats_normalized"
  ];

  rr:subjectMap [
    rr:template "http://linked-web-apis.fit.cvut.cz/resource/{$}
    _format";
  ];

  rr:predicateObjectMap [
    rr:predicate dcterms:title;
    rr:objectMap [
      rml:reference "$"
    ]
  ];

    rr:predicateObjectMap [
```

```
    rr:predicate rdfs:label;
    rr:objectMap [
      rml:reference "$"
    ]
  ] .


#Architectural Style
<#ArchitecturalStyleMapping>

  rml:logicalSource [
    rml:source "data/programmableweb/pweb.json" ;
    rml:referenceFormulation ql:JSONPath ;
    rml:iterator "$.architectural_style_normalized"
  ];

  rr:subjectMap [
    rr:template "http://linked-web-apis.fit.cvut.cz/resource/{$}
    _architecturalStyle";
  ];

  rr:predicateObjectMap [
    rr:predicate dcterms:title;
    rr:objectMap [
      rml:reference "$"
    ]
  ];

    rr:predicateObjectMap [
    rr:predicate rdfs:label;
    rr:objectMap [
      rml:reference "$"
    ]
  ] .


#Auth model
<#AuthModelMapping>

  rml:logicalSource [
    rml:source "data/programmableweb/pweb.json" ;
    rml:referenceFormulation ql:JSONPath ;
    rml:iterator "$.auth_model_normalized"
  ];

  rr:subjectMap [
    rr:template "http://linked-web-apis.fit.cvut.cz/resource/{$}
    _authModel";
  ];

  rr:predicateObjectMap [
    rr:predicate dcterms:title;
    rr:objectMap [
      rml:reference "$"
```

```
    ]
  ];

    rr:predicateObjectMap [
    rr:predicate rdfs:label;
    rr:objectMap [
      rml:reference "$"
    ]
  ] .
```

Listing C.1: ProgrammableWeb RML rules

Appendix D

Deployment instructions

This appendix describes the instructions for proper application deployment. The instructions are written for Ubuntu operation system. Commands may differ for other types of Linux OS.

D.1 Requirements

For proper application use and running the following requirements needed:

1. Pre-installed Java JDK version 1.4 or above
2. Pre-installed Maven build automation tool
3. Minimum 1GB of RAM

D.2 Instructions

As were mention above pre-installed Java JDK required as well as Maven building tool.

D.2.1 Installing Java JDK

Run the following command in console to download JDK:

```
sudo apt-get install default-jdk
```

D.2.2 Installing Maven

Run the following command in console to download Maven build automation tool:

```
sudo apt-get install maven
```

When all requirements are fulfilled the application can be automatically or manually deployed.

D.2.3 Automatic deployment

For automatically application building the bash script was created. The script is given below:

```
echo "Downloading Semantic-Web-API-Hub"
git clone --recursive https://bitbucket.org/m1ci/semantic-web-apis
    -hub/

echo "Downloading RML library"
git clone --recursive https://github.com/RMLio/RML-Mapper.git
cd RML-Mapper
git submodule update --init --recursive

echo "Building RML library"
mvn clean install -DskipTests

echo "Adding RML library to Semantic-Web-API-Hub project"
mvn install:install-file -Dfile=RML-Processor/target/RML-Processor
    -0.3.jar -DgroupId=rml -DartifactId=rml -Dversion=0.3 -
    Dpackaging=jar
cd ..

echo "Building Semantic-Web-API-Hub"
cd semantic-web-apis-hub
mvn clean install

echo "Finish"
```

Listing D.1: The application bash building script

D.2.4 Manual deployment

The application can be builded using the following set of instructions.

D.2.4.1 Download a repository

The application can be downloaded from Bitbucket repository using the following command:

```
git clone --recursive https://bitbucket.org/m1ci/semantic-web-apis
    -hub/
```

D.2.4.2 Download and build RML library

The RML library can be downloaded from Github repository using following commands:

```
git clone --recursive https://github.com/RMLio/RML-Mapper.git
git submodule update --init --recursive
```

When RML library is downloaded, it should be built using following commands:

```
mvn clean install -DskipTests
```

In RML-Processor/target/ folder get RML library file RML-Processor-0.3.jar and copy into the root of the semantic-web-apis-hub folder for further adding to Maven build manager.

D.2.4.3 Adding RML dependency

RML library can be added to Maven using the following command:

```
mvn install:install-file -Dfile=RML-Processor-0.3.jar -DgroupId=
    rml -DartifactId=rml -Dversion=0.3 -Dpackaging=jar
```

D.2.4.4 Project build

The project must be built using Maven build manager:

```
mvn clean install
```

Maven build manager will download and install all required dependencies. Finally the application will be built and ready for execution.

D.3 Application commands

The application can be run using certain command set.

Extract specific API directory:

```
java -jar Sem-API-Maven-1.jar extract {pw, pw-mashup, apis-io,
    apis-guru, exicon, api-for-that}
```

Extract all API directories:

```
java -jar Sem-API-Maven-1.jar extract-all
```

Transform extracted specific API data into RDF format:

```
java -jar Sem-API-Maven-1.jar transform {pw, pw-mashup, apis-io,
    apis-guru, exicon, api-for-that}
```

Transform all extracted API data into RDF format:

```
java -jar Sem-API-Maven-1.jar transform-all
```

D.4 Application source

The application can be found and downloaded from the Bitbucket Git revision control system from the following link: `https://bitbucket.org/m1ci/semantic-web-apis-hub`