

Master's thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science and Engineering

Platform for semantic extraction of the web

Jakub Podlaha

January 2015

/ Declaration

Prohlašuji, že jsem se neflákal.

Abstrakt / Abstract

Tento dokument je pouze pro potřeby testování.

Překlad titulu: Platforma pro sémantickou extrakci webu

This document is for testing purpose only.

Contents /

1 Introduction	1
1.1 Problem Statement and Motivation	1
1.2 Current solution crOWler	3
1.3 Proposed Solution and Methodology	3
1.4 Specific goals of the project	3
1.5 Work structure (XXX)	3
2 Knowledge base, principles and technologies	4
2.1 Technology of Semantic Web	4
2.2 Linked Data	4
2.3 RDF and RDFS	5
2.3.1 URI	5
2.3.2 RDF and RDFS vocabulary	5
2.4 OWL	6
2.5 RDFa	6
2.6 dalsi	6
2.7 automatická extrakce dat	6
3 Existing solutions	7
3.1 Semantic and non semantic crawlers	7
3.2 Advantages and pitfalls of Semantic crawler and linked data	7
3.3 crOWler	8
3.3.1 zavislosti	8
3.3.2 Classes of CrOWler	9
3.3.3 Run configuration	9
3.4 Research - existující řešení - platforma	9
3.4.1 InfoCram 2000 - Jirka Mašek	9
3.4.2 iMacros	9
3.4.3 Selenium	9
3.5 Strigil!	11
3.5.1 What problems it solves? (Use cases)	11
3.5.2 Architecture of Strigil platform	11
3.5.3 What inspiration it brings for crawler	11
4 Program design	12
4.1 Use Cases	12
4.2 Workflow	12
4.2.1 Main line	12
4.2.2 Scenario creation	12
4.2.3 Additional branches to Scenario Creation	12
4.3 Model	13
4.4 Implementation	13
4.5 Issues - solved and unsolved ...	13
5 Program Implementation	14
5.1 Libraries XXX	14
5.1.1 rdfQuery	14
6 Results and Tests	15
6.1 Data	15
6.1.1 Pamatky	15
7 Conclusion	16
Literatura	17
A Abbreviations	19

Tables /

2.1. RDF and RDFS vocabulary	5
------------------------------------	---

Chapter 1

Introduction

During past few years the Web went through bigger or smaller revolutions.

- WEB 2.0 and tag cloud
- HTML5 and semantic tags
- Smartphones, Tablets and mobile web everywhere,
- The run out of IPv4 addresses, nonexistent boom of IPv6,
- Cloud technologies and BigData,
- Bitcoin, Tor, anonymous internet,
- WikiLeaks, NSA, Heartbleed and security concerns
- Google Knowledge Graph, Facebook Open Graph, ...

That's only few examples of some of the biggest recent issues on the web in general. We live in an age, where so little can mean so much. The environment online is dramatically changing, mostly on a wave of some new, useful or frightening technology. The Semantic Web technologies have been described, standardized and implemented for several years now (XXX Example with linked data, rdf) and their tide seems to be near, though yet to come.

Semantic Web itself relates to several principles (along with their implementation) that allow users to add meaning to their data. This meaning brings not only a standardized structure, but also, as a consequence, the possibility to query and reason on data from multiple sources. Once given the structure, similar data can be joined in a form of a bigger cloud. This phenomena is called Linked Data.

In this work we'd like to bring the Semantic Web technologies closer to users. The approach is to propose a methodology for extracting structured data out of unstructured ones, designing and implementing an appropriate tool, to simplify the process of annotating (yet anonymous) data on a webpage, i.e. to bring structure and meaning into it.

1.1 Problem Statement and Motivation

Giving meaning, i.e. semantization of web pages gets more popular. Probably the most obvious example can be seen in the way the Google search engine serves it's results. Presenting not only the resulting pages but as well snippets of information scraped directly from the page content such as menu fields parsed directly from HTML5, contact information or opening hours, or even visualizing data from their own internal ontology, the Knowledge Graph.

XXX https://en.wikipedia.org/wiki/Google_Knowledge_Graph

XXX Strigil - <http://delivery.acm.org/10.1145/2540000/2539170/p453-starka.pdf>

What are the options for bringing semantic into a web?

One direction to go (XXX better) is to annotate data on **the server side**, i.e. at the time it is being created and/or published. The person or engine creating the data

on-the-go. By using existing ontologies we not only give the meaning to our data, but also valuable connection to any other dataset annotated using the same ontology.

1.2 Current solution crOWLer

The suggested base-technology is being developed on our faculty XXX. Crawler called crOWLer serves the needs of extracting data from web. In current implementation, both, the scenario, followed by the crawler, and the ontology structure/schema are hard-coded into the crOWLer code. This requires unnecessary load of work for each separate use case, whilst in practice all the use cases share the same workflow.

1. load the ontology
2. add selectors to specific resources from the ontology
3. implement the rules to follow another page
4. run the crawling process according the above

1.3 Proposed Solution and Methodology

To simplify the creation of guidelines, or scenarios for crOWLer, we propose a tool that allows user to select all the element directly on the web page being crawled, with all the necessary settings, pass the scenario created to the crOWLer and obtain the results in a form of a graphical feedback.

1.4 Specific goals of the project

- design the semantic data creation use-cases
- implement extension for a browser
- load and visualise ontology
- create scenario for crOWLer
- serialize scenario and ontology
- parse it by crOWLer creating it's configuration
- run crOWLer
- visualize the extracted data (feedback)

1.5 Work structure (XXX)

XXX TBD

Chapter 2

Knowledge base, principles and technologies

2.1 Technology of Semantic Web

Wikipedia defines Semantic Web as a collaborative movement led by international standards body the World Wide Web Consortium (W3C). [1] W3C itself defines Semantic Web as a technology stack to support a **Web of data**, as opposed to **Web of documents**, the web we commonly know and use (XXX <http://www.w3.org/standards/semanticweb/>). Just like with **Cloud** or **Big Data** the proper definition tends to vary, but the notion remains the same. It is collaborative movement led by W3C and it does define a technology stack. It also includes users and companies using this technology and the data itself. Technologies and languages of Semantic Web such as RDF, RDFa, OWL, SPARQL (XXX) are well standardized and will be described in following chapters.

As a general logical concept of the technology... (XXX) Technology of Semantic Web is used to take data and metadata, give them unique identifiers and form them into oriented graphs. The metadata part define a schema of types and properties that can be assigned to data and also relations between this types and properties possibly in a form of ontology. When some data are annotated by resources from such an ontology we gain power to reason on this data, i.e. resolve new relations based on known ones, and also to query on our data along with any data annotated using the same ontology.

In RDF(S) the is defined in a form of triples. Triple consists of subject, predicate and object, wich all are simply **resources** listed by their identifiers. In this very general form we can express basically any relationship between two resources. On a level of classes and properties, we can for example assign a type to an individual, or set a class as a domain of some property. On a level of ontologies we can specify author and date it was released. (XXX DELME)

2.2 Linked Data

Wikipedia defines Linked Data as a **term used to describe a recommended best practice for expo**. Just like Semantic Web it's a phenomena, a community, a set of standards created by this community, tools and programs implementing these standards and people willing to use these programs. Linked data strives to solve problem of XXX unreachability of majority of data on the web, as they're not accessible in machine readable, by defining standards and supporting implementation of those standards.

- <http://linkeddata.org/guides-and-tutorials>
- <http://linkeddatabook.com/editions/1.0/>
- <http://lov.okfn.org/dataset/lov/>

2.3 RDF and RDFS

RDF is a family of specifications for syntax notations and data serialization formats, meta data modeling, and vocabulary used for it.

XXX https://en.wikipedia.org/wiki/Resource_Description_Framework

We will look closely on URI, the resource identifier, vocabularies and semantics defined by RDF, RDFS, and OWL, and serialization into Turtle and RDF/XML formats.

2.3.1 URI

In order to give each resource an unique identifier a Uniform Resource Identifier is used. This is mostly in a form of URL as we commonly know it as **web address** (e.g. <http://www.example.org/some/place#something>). In some cases URI can be a URN as well. URN is a complementary syntax for URL that allow us to identify resources without specifying their location. This way we can for example use ISBN codes when working with books and records, or UUID identifier a Universally Unique Identifier widely used to identify technically any data instance.

XXX https://en.wikipedia.org/wiki/Uniform_resource_identifier

2.3.2 RDF and RDFS vocabulary

In order to work with data properly RDF(S) vocabulary defines several basic URIs along with their semantics.

resource	description
rdf:type	a property used to state that a resource is an instance of a class a commonly accepted qname for this property is r
rdfs:Resource	the class of everything; all things described by RDF are resources
rdfs:Class	declares a resource as a class for other resources
rdfs:Literal	literal values such as strings and integers property values such as textual strings are examples of RDF literals literals may be plain or typed
rdfs:Datatype	the class of datatypes rdfs:Datatype is both an instance of and a subclass of rdfs:Class each instance of rdfs:Datatype is a subclass of rdfs:Literal
rdf:XMLLiteral	the class of XML literal values; rdf:XMLLiteral is an instance of rdfs:Datatype (and thus a subclass of rdfs:Literal)
rdf:Property	the class of properties
rdfs:domain	(of an rdf:predicate) declares the class of the subject in a triple whose second component is the predicate
rdfs:range	(of an rdf:predicate) declares the class or datatype of the object in a triple whose second component is the predicate
rdfs:subClassOf	allows to declare hierarchies of classes
rdfs:subPropertyOf	an instance of rdf:Property that is used to state that all resources related by one property are also related by another
rdfs:label	rdf:Property used to provide a human-readable version of a resource's name
rdfs:comment	rdf:Property used to provide a human-readable description of a resource

Table 2.1. RDF and RDFS vocabulary

These are the basic building blocks of our future RDF graphs. The semantics defined in the specification and slightly described here 2.1 allow us to specify class hierarchy,

properties with domain and range as well as use this structure on individuals and literals.

■ 2.4 OWL

Additionally to RDF and RDFS the OWL – Web Ontology Language, is a family of languages for knowledge representation. OWL extends syntax and semantics of RDF brings in notion of subclasses and superclasses, distinction between datatype properties and object properties, defines transitivity, symmetricity and other logical capabilities of properties. When querying, OWL allow us on unions or intercections of classes or cardinality of properties. All this capabilities comes in with well defined semantics and brings in necessary computational complexity when reasoning on OWL ontology.

XXX

- <http://www.w3.org/TR/owl2-primer/>
- https://en.wikipedia.org/wiki/Web_Ontology_Language
- <http://www.w3.org/TR/2012/REC-owl2-quick-reference-20121211/>

■ 2.5 RDFa

RDFa technology defines concept of embedding data on the web document in HTML by semantics. This is accomplished by adding custom attributes containing the RDF related information, such as resources, properties.

- <https://www.sio2.cz/web/psiotwo/publications>
- <http://rdfa.info/play/>

■ 2.6 dalsi

- <https://en.wikipedia.org/wiki/SPARQL>
- [https://en.wikipedia.org/wiki/Turtle_\(syntax\)](https://en.wikipedia.org/wiki/Turtle_(syntax))

■ 2.7 automatická extrakce dat

TODO in next section

Chapter 3

Existing solutions

3.1 Semantic and non semantic crawlers

By researching existing solutions, there is currently no open source or openly available solution to solve this task. Rumor goes there is proprietary tool in IBM.

Existing tools named as **Ontology-based Web Crawlers** refer mostly to crawlers that **rank** pages being crawled by guess-matching them against some ontology. In those programs user can't specify data that are being retrieved. Moreover, there is no way to get involved in the crawling process. It is solely used to automatically rank the relevance of documents. They are solving different task where input is several documents and possibly an ontology and output is the best matching document.

In case we are trying to solve the input is one or more documents and one or more ontologies and the result is data obtained from the documents and annotated with resources from the ontologies.

3.2 Advantages and pitfalls of Semantic crawler and linked data

The simplest approach is manual searching for keywords, or even simple browsing the web. That might be useful in some cases, but when there is a lot of data, it becomes exhausting.

Crawling data using simple tools like 'wget -mirror' allows us to load data and then write a program or script to retrieve a relevant information. This approach takes a lot of energy for one time only solution of a given problem.

By storing such crawled data into database we obtain persistent database, possibly automatically obtained by the script from pervious case. Such data is static, but can be queried over and over and possibly re-retrieved when becomes obsolete. It's structure is, however, based on programmers imagination and needs to be described in order to understand and handle the data properly.

When using Ontology-based solution, tailor made for crawling and annotating data from web, we obtain several benefits **for free**. The tool designed specially for this purpose makes it easy. Once the data is annotated, we can not only query on them, but also automatically reason on them and obtain more or more specific/narrow results than with general data. The attributes and relations within ontology, that allow reasoning, are usually part of the ontology definition and as such comes, again, **for free**.

Last for benefits: using ontology from public resource as a schema for our data can give us correct structure without need for XXX making it up or building it from scratch. Also by using some common ontology, we can join together any accessible data structured according to this ontology and simply query on resulting super set. With this approach we can utilize the power of linked data cloud (XXX reference).

Semantic crawling is not a silver bullet. The technology is only finding it's place and uses and it's being shaped by the needs of it's users. In current it's mostly used on accademic field XXX.

There is always a threat of inconsistency of an ontology when some data don't fit the rules or breaks structure of an ontology. (XXX more)

Just like with **hardcoded** crawling technique, the semantic crawling is tightly connected (XXX better) with the structure of the web being crawled and selectors (XXX explain term) used for matching data on the web. Any change on a webpage structure can lead to broken selectors or links during the crawling process (XXX and make the scenario useles, more on self-repairing of scenarios?).

A lot of web pages loads their data dynamically using AJAX queries. Some pages simply changes it's content frequently (XXX typically news pages, forums: rt.com, vimeo.com, ...) which would require almost constant crawling and growth into an massive ontology (XXX any suggestions on that? =).

Stating that, the semantic crawling is an usefull way to effectively obtain and query on (otherwise anonymous) data from the web, but it still have it's challenges to overtake.

3.3 crOWLer

3.3.1 zavislosti

■ maven - apache project managing tool

- <https://maven.apache.org>
- <https://maven.apache.org/run-maven/index.html>
- <https://maven.apache.org/guides/mini/guide-ide-eclipse.html>

■ sesame

- <http://www.openrdf.org/download.jsp> ??

■ jena

- <https://github.com/ansell/JenaSesame> !!
- or <https://github.com/afs/JenaSesame> ??
- or <http://jena.apache.org/> ???
- or <http://sjadapter.sourceforge.net/> ????
- or <http://sourceforge.net/projects/jenasesame/>
- might help <http://www.iandickinson.me.uk/articles/jena-eclipse-helloworld/> ■
- little hint http://spqr.cerch.kcl.ac.uk/?page_id=130
- another hit <http://answers.semanticweb.com/questions/20865/how-to-get-the-jena-sesame->
- wiki [https://en.wikipedia.org/wiki/Jena_\(framework\)](https://en.wikipedia.org/wiki/Jena_(framework))
- jena vs. sesame flame <http://answers.semanticweb.com/questions/1638/jena-vs-sesame-is-there-a-serious-complete-up-to-date-unbiased-well-informed-side-by>

3.3.2 Classes of CrOWLer

- `ImmovableHeritageConfiguration` extends `MonumnetConfiguration` implements `ConfigurationFactory`
 - implements `Configuration`, which is parameter for `FullCrawler.run()` method
- `FullCrawler`
 - implements the whole crawling algorithm
 -

3.3.3 Run configuration

```
crowler cz.sio2.crowler.configurations.npu.ImmovableHeritageConfiguration
file results
crowler cz.sio2.crowler.configurations.kub1x.KbxConfiguration file re-
sults
crowler cz.sio2.crowler.configurations.parser.SeleniumConfiguration\
file results generated.html
```

- Class `ImmovableHeritageConfiguration` implements `Configuration` class.
- Folder `jena_con` will be created and all the rdf's will be stored in int with names derived from ontology uri

3.4 Research - existující řešení - platforma

3.4.1 InfoCram 2000 - Jirka Mašek

- zalozeny na Aardwark ¹⁾

3.4.2 iMacros

- http://wiki.imacros.net/Command_Reference
- http://wiki.imacros.net/iMacros_for_Firefox
- http://wiki.imacros.net/iMacros_for_Chrome

3.4.3 Selenium

Selenium is a collection of tools for automated testing of web pages. This tools include:

- Selenium IDE – a Firefox plugin for creating test scenarios
- WebDriver – a set of libraries for various languages capable of running tests generated from Selenium scenarios

A user of Selenium, typically a web designer, programmer or coder, would create a scenario using Selenium IDE, in order to test his web server. From this scenarion a unit test can be generated for desired programming language and in desired form (e.g. JUnit test case). Such a test can be simply included it in a set of tests for the web server project. WebDriver library needed for running these tests is available through Maven. There is also a chance to use PhantomJs no-gui web browser for

¹⁾ <https://addons.mozilla.org/en-US/firefox/addon/aardvark/>

running tests without a need for actual browser, for cases when tests are being executed automatically in background or on server environment without X server or other form of graphical interface. The capabilities of WebDriver make it one of the most popular testing platforms for web servers nowadays XXX.

- IDE - <http://www.seleniumhq.org/projects/ide/>
- plugins - <http://www.seleniumhq.org/projects/ide/plugins.jsp>
- current commands - <http://release.seleniumhq.org/selenium-core/1.0.1/reference.html>
- documentation - <http://docs.seleniumhq.org/docs/index.jsp>
- extending selenium API (blog, tutorial) - <http://adam.goucher.ca/?s=selenium&paged=2>
- randomString example - <http://adam.goucher.ca/?p=1348>

Selenium IDE is a Firefox plugin that allow us to directly record user actions on webpage such as following links, storing and comparing values, filling in and submitting forms.

An attempt was made to implement SOWL as a plugin for Selenium IDE. This plugin would have two parts:

1. an extension of graphical interface
2. a formatter that would generate scenarios for crOWler in some desired form

Certain limitations were discovered during developement of this plugin. Selenium IDE, as being plugin itself, implements it's own plugin system, through which it allows other developers to extend it's functionality. The Selenium IDE plugin API allows us to use standard Firefox techniques along with predefined API, to extend the graphical interface and the functionality of the IDE respectively.

Graphical interface is defined using XUL, the standard Mozilla XML format for defining user interface. XUL defines an overlay sysem using which a new layer is defined and layed over existing part of application layout while extending or modifying it. The overlay sytem itself comes with Mozilla stack and can be used on IDE by default.

The functionality of IDE is, however, linked with it's layout and has to be taken in account. Selenium IDE internally defines set of commands that can be used in scenarios. List of default commands can be seen in dropdown on main screen of the IDE. This list can be extended, but the use and structure of commands is implemented internally in Selenium IDE. Addition of new commands is XXX accomplished by extending the `Selenium.prototype` object in registered plugin. After the extension is processed through internal command loader, a new set of commands is added for user to use.

Commands in this system are recognized by their names as they are assigned on the prototype object the prefixes used are:

- do – the action commands – for performing user actions
- get and is – the accessor commands – for testing and/or waiting for a values on page and potentially storing it
- assert – the assertio commands – for performing actual tests

When `command` is generated the prefix is being stripped and according to type, multiple versions commands can be created. For example `do` commands have always `immediate` and `patient` version and in this principle `Selenium.prototype.doClick` will generate the `click` and `clickAndWait` command. Accessor commands are even

more complex and generate eight commands for every single method (positive and negative assertion, store method, waitFor, etc.). Implementation of the command method defines, how Selenium IDE would behave when **replying** the scenario recorded. Technically it is possible to leave the implementation empty in the IDE and use it only as a command for WebDriver unit test.

None of the original command types corresponds to format of commands for handling the semantic annotation, like adding URI to element, recording creation of individual, assigning literal to it's property etc. A new set of commands was suggested and partially implemented having the prefix **owl**. This led to changes in core sources of Selenium IDE, which itself is a bad sign as it technically creates a new branch of the program. CommandBuilder had to be extended directly in the selenimu code as it's impossible to change it's behavior through native Selenium IDE API. Unfortunately, even though the new command type was implemented, it is not possible to change the more general concept of all commands. Every command is stored as (**name**, **target**, **value**) triple and from this format everything is derived. It is technically impossible to create command for example for literal along with it's language tag as there is simply no field for it. For the same reason we can't create a command to create an ontological object of some type as a property of another object. These commands relate to each other, but such a behavior is not supported by the scenario editor in it's current architecture. There is also no way to alter editor GUI for specific command. For instance, we can't offer autocomplete for input field when user enters URI of ontological resource. Such a feature would be an essential part of SOWL's workflow, and as a consequence these limitations are critical and disallow us from properly implementing SOWL on top of the Selenium IDE.

■ 3.5 Strigil!

■ 3.5.1 What problems it solves? (Use cases)

Strigil's architecture is tailor made for parallel processing of documents. As such it can't properly handle temporal changes in documents and thus it is designed to manipulate with static documents without use of javascript. (XXX)

■ 3.5.2 Architecture of Strigil platform

■ 3.5.3 What inspiration it brings for crawler

XXX I tried to include Strigil/XML XXX format in SOWL, but it was XXX ridiculous. It would bring in an unnecessary workload on string-based serialization from native javascript objects into XML format. The decision was made to rather use native JSON serialization as described in XXX chapter implementation. This implementation is heavily inspired by the original Strigil/XML. Moreover it attends to improve upon readability and compactness even though it doesn't reach the richness of Strigil/XML format.

Chapter 4

Program design

4.1 Use Cases

- NPU
- RLP
- beerborec.cz
- citybee.cz

4.2 Workflow

4.2.1 Main line

- user loads/creates ontology using sowl
- user opens webpage with data
- user creates scenario using sowl
- sowl sends scenario to crawler
- crawler crawls the web according to scenario and stores results in repository
- + crawler sends data to sowl which embeds them in original web page (XXX)

4.2.2 Scenario creation

- user starts scenario creation in sowl
- loop until finished:
 - user selects an element on page
 - user select action on element (perform and record event, i.e. click on link, narrow HTML context, assign element - object or property according to situation, ...)
 - sowl records the action in scenario

4.2.3 Additional branches to Scenario Creation

- user can navigate through scenario by clicking scenario steps
- user can navigate through scenario by clicking ontological context
- user can navigate through scenario by clicking areas on webpage covered by scenario
- when user clicks on a hyperlink:
 - existing template can be assigned to the action (no need to actually follow the link)
 - new template can be created for resulting action (resulting page loaded, new template created, click through shown in breadcrumbs)

■ 4.3 Model

■ 4.4 Implementation

■ 4.5 Issues - solved and unsolved

- error handling (non existent selector, missing data, ...)

Chapter 5

Program Implementation

5.1 Libraries XXX

5.1.1 rdfQuery

rdfQuery is a javascript library for RDF-related processing. It supports RDFa, RDF, OWL for parsing data, it can dynamically embed HTML webpage with RDFa data. rdfQuery is written in similar style as jQuery, popular Javascript library. The intended use is to write queries over data stored in rdfQuery internal datastore in similar way as DOM object are queried using jQuery. Moreover the whole concept is based on SPARQL keywords, taking the best from each world XXX.



<https://code.google.com/p/rdfquery/>


Chapter 6

Results and Tests

6.1 Data

6.1.1 Památky

- <http://onto.mondis.cz/resource/page/npu/>
- <http://monumnet.npu.cz/pamfond/list.php?hledani=1&KrOk=&HiZe=&VybUzemi=1&sNazSidOb=&Adresa=&Cdom=&Pamatka=&CiRejst=&Uz=B&PrirUbytOd=3.5.1958&PrirUbytDo=10.12.2013>
- <http://dominanty.cz/pamatky-cihana.php>



Chapter 7

Conclusion

TBD



Literatura

[1] *Semantic Web – Wikipedia.*

[https://en.wikipedia.org/wiki/Semantic_Web.](https://en.wikipedia.org/wiki/Semantic_Web)



Appendix A

Abbreviations

MDN	Mozilla Developers Network
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
RDF	Resource Description Framework
RDFS	RDF Schema - set of classes and properties providing basic elements for the description of ontologies
OWL	Web Ontology Language
SPARQL	SPARQL Protocol and RDF Query Language - query language for semantic databases/triplestores
foaf	friend of a friend - a popular ontology for describing personal information and relationships