

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!	12	
3.5.1 What problems it solves? (Use cases)	12	
3.5.2 Architecture of Strigil platform	12	
3.5.3 What inspiration it brings for crawler	12	
4 Program design	13	
4.1 Use Cases	13	
4.1.1 Use Case 1 – basic example case	13	
4.1.2 Use Case 2 - NPU	13	
4.2 Workflow	13	
4.2.1 Main line	14	
4.2.2 Scenario creation	14	
4.2.3 Additional branches to Scenario Creation	14	
4.3 Model	14	
4.4 Implementation	14	
4.5 Issues - solved and unsolved	14	
5 Program Implementation	15	
5.1 Libraries XXX	15	
5.1.1 rdfQuery	15	
5.2 Scenario format	15	
5.2.1 Strigil/XML	15	
5.2.2 Evolution of XML format	16	
5.2.3 SOWL/JSON	16	
6 Results and Tests	19	
6.1 Data	19	
6.1.1 Pamatky	19	
7 Conclusion	20	
Literatura	21	
A Abbreviations	23	

Tables / Figures

2.1. RDF and RDFS vocabulary5

3.1. Image of Selenium IDE 11

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.crawler.configurations.npu.ImmovableHeritageConfiguration
file results
crowler cz.sio2.crawler.configurations.kub1x.KbxConfiguration file re-
sults
crowler cz.sio2.crawler.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

InfoCram 2000 is part of project ExtBrain XXX

- 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

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

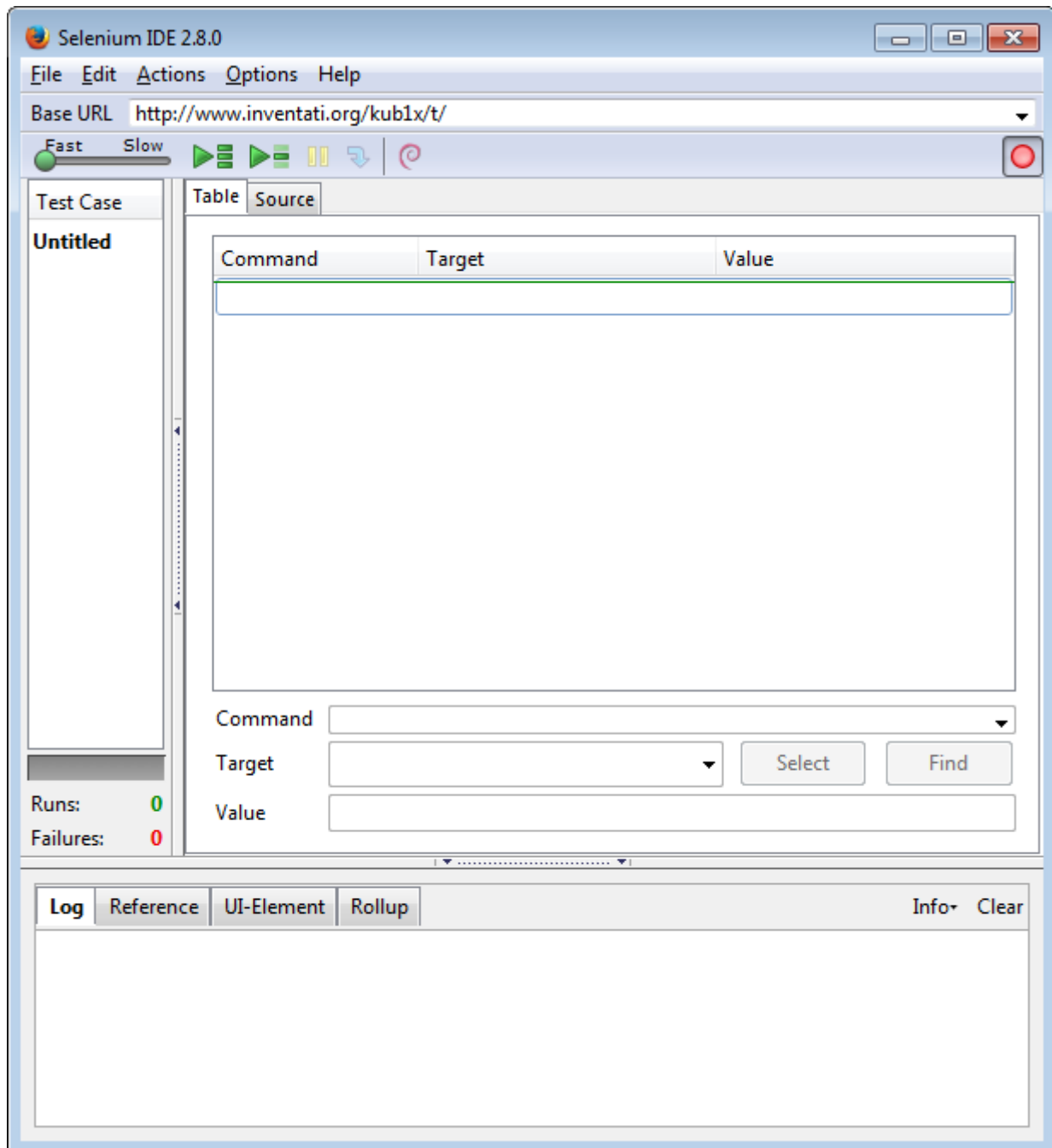


Figure 3.1. GUI of Selenium IDE showing the Command, Target and Value fields.

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)

Strigils 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

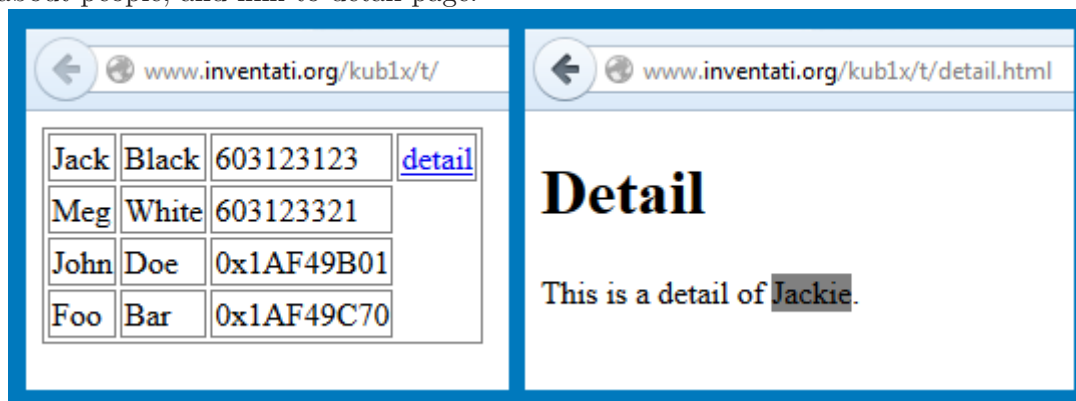
Program design

4.1 Use Cases

In following part I'd like to describe several use cases that should be solvable by implementation this work XXX

4.1.1 Use Case 1 – basic example case

I've created sample general use case on webpage <http://www.inventati.org/kub1x/t/>. This use case can be seen on picture below. It consists of table holding values about people, and link to detail page.



In order to fulfill this usecase SOWL should support following operation:

- load the foaf ontology that describe data bout people
- create scenario with two templates: init and detail
- save this scenario to a file

crOWLer should be able to:

- accept and parse scenario created by SOWL
- follow this scenario while scraping data from the page
- store results into rdf files

4.1.2 Use Case 2 - NPU

- 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/>

5.2 Scenario format

One of main tasks of this work was to create format for scenario generated by SOWL and consumed by crOWler. This scenario will describe information necessary for the crawling process: what operation to do (create ontological object, assign property to such an object, perform task with webpage).

This task is closely related to implementation peculiarity of semantic crawler: we're dealing with two separate contexts at the same time, the ontological and the web context. Ontological context holds current object (individual) to which we assign properties, web context hold current webpage along with currently selected element on that webpage. Scenario have to support operations to change each context separately and/or both at the same time.

5.2.1 Strigil/XML

Scrigil, the scraping platform in order to solve similar problem as crOWler introduces it's own XML based Scraping Script format. It's documentation can be found here XXX ¹⁾.

Basis of the whole script is system of *templates*. Each template has a name and mime-type declaring type of document the template is designed for. This information is needed as Strigil supports HTML and also Excel spreadsheet files. Templates call each other using `call-template` command anywhere in the script. This command accepts URL as an argument from it's nested commands. Each template is called only with new

¹⁾ <https://drive.google.com/file/d/0B40n-1Gb38CgWlAyZDhGbDV2TFk/edit> Scraping script documentation

URL, thus on new document. Of course URL of current document can be passed as an argument, but due to nature of Strigil, this would create completely separate context.

Strigil is tailor made for parallel processing. The architecture of the Strigil system contains not only scraping processor, but also a layer for distributed download queue processing and layer of proxy servers that can be used to spread the traffic and scale the download process horizontally. As the downloads are performed asynchronously and can be even delayed due to network lags and timeouts, there is no guaranteed order in which documents will be scraped. Each of Strigil templates create it's own context when called. If we want to link data obtained from different template calls we have to use some additional techniques. For example we can assign some properly defined, non-random, unique identifiers to an object. This identifier have to be quaranteed to be the same for the same object through different template calls and potentially on different pages.

To handle ontological data manipulation the commands `onto-elem` and `value-of` are used. First one creates an individual of given type and, if nested into different `onto-elem` relates this new individual to it's parent with some property. Literals are assigned to properties of parent object using `value-of` commad with property name specified. This command is very powerful with usage regular expressions, selectors or nested calls of itself it can create arbitrary values from constants and data obtained from web page being processed.

Strigil also implements variety of functoins to help with processing of textual data. Function `addLanguageInfo`, for example, is widely used in Strigil scraping scripts to add language tags to string literals. The function call can be seen below.

```
<scr:function name="addLanguageInfo">
  <scr:with-param>
    <scr:value-of select="Hello World" />
  </scr:with-param>
  <scr:with-param>
    <scr:value-of text="en" />
  </scr:with-param>
</scr:function>
```

Similarly we can use function `addDataTypeInfo` to add datatype flag, function `generateUUID` to obtain unique identifier or function `convertDate` to convert Czech and English dates into a common `xsd:date` format and several others. Some functions, like the last one mentioned, cover task-specific issues and Strigil doesn't define a way to extend the list of functions.

In early stages of SOWL developement an attempt was made to use original Strigil/XML as a format of choice. An appropriate, consistent subset was chosen that would cover required use cases. Implementation of simple use cases revealed some pitfalls of this decision and revealed several suggestions for improvements on the approach and the format itself.

■ 5.2.2 Evolution of XML format

■ 5.2.3 SOWL/JSON

This is the format for s The final scenario for Use Case 1 XXX looks like this:

```
{
  type: "scenario",
  name: "manual",
```

```

ontology: {
  base: "http://kublx.org/onto/dip/t/",
  imports : [
    {
      prefix: "foaf",
      uri: "http://xmlns.com/foaf/0.1/",
    },
    {
      prefix: "kbx",
      uri: "http://kublx.org/onto/dip/t/",
    },
  ],
},
creation-date: "2014-11-30 12:40",
call-template: {
  command: "call-template",
  name: "init",
  url: "http://www.inventati.org/kublx/t/",
},
templates: [
  {
    name: "init",
    steps: [
      {
        command: "onto-elem",
        typeof: "http://xmlns.com/foaf/0.1/Person",
        selector: {
          value: "tr",
          type: "css",
        },
        steps: [
          {
            command: "value-of",
            property: "http://xmlns.com/foaf/0.1/firstName",
            selector: {
              value: "td:nth-child(1)",
              type: "css",
            },
          },
          {
            command: "value-of",
            property: "http://xmlns.com/foaf/0.1/lastName",
            selector: {
              value: "td:nth-child(2)",
              type: "css",
            },
          },
          {
            command: "value-of",
            property: "http://xmlns.com/foaf/0.1/phone",
            selector: {
              value: "td:nth-child(3)",
              type: "css",
            },
          },
        ],
      },
    ],
  },
]

```

```
    },  
  },  
  {  
    command: "call-template",  
    name: "detail",  
    attribute: "href",  
    selector: {  
      value: "detail",  
      type: "css",  
    },  
  },  
],  
},  
],  
{  
  name: "detail",  
  steps: [  
    {  
      command: "value-of",  
      property: "http://xmlns.com/foaf/0.1/nickname",  
      selector: {  
        value: ".nick",  
        type: "css",  
      },  
    },  
  ],  
},  
],  
},  
],  
}
```



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