

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



## Acknowledgement / Declaration

I'd like to thank my parents and family for enormous support, my supervisor for endless patience and guidance and my friends for not letting me go insane.

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 /

<b>1 Introduction</b> .....	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 .....	4
<b>2 Knowledge base, principles and technologies</b> .....	5
2.1 Technology of Semantic Web .....	5
2.2 Linked Data .....	5
2.3 RDF and RDFS .....	6
2.3.1 URI .....	6
2.3.2 RDF and RDFS vocabulary .....	7
2.4 OWL .....	7
2.5 RDFa .....	8
2.6 SPARQL .....	8
2.7 RDF/XML syntax .....	8
2.8 Turtle syntax .....	8
<b>3 Existing solutions</b> .....	9
3.1 Semantic and non semantic crawlers .....	9
3.2 Advantages and pitfalls of Semantic crawler and linked data .....	9
3.3 Analysis of crOWler .....	10
3.4 Finding platform for frontend .	11
3.4.1 InfoCram 6000 – ExtBrain .....	11
3.4.2 Selenium .....	13
3.5 Strigil .....	15
3.5.1 What problems it solves? (Use cases) .....	15
3.5.2 Architecture of Strigil platform .....	15
3.5.3 What inspiration it brings for crawler .....	15
3.6 Early implementation .....	15
<b>4 Program design</b> .....	17
4.1 Use Cases .....	17
4.1.1 Use Case 1 – basic example case .....	17
4.1.2 Use Case 2 - NPU .....	17
4.2 Workflow .....	17
4.2.1 Main line .....	18
4.2.2 Scenario creation .....	18
4.2.3 Additional branches to Scenario Creation .....	18
4.3 Model .....	18
4.4 Implementation .....	18
4.5 Issues - solved and unsolved ...	18
<b>5 Program Implementation</b> .....	19
5.1 Libraries XXX .....	19
5.1.1 rdfQuery .....	19
5.1.2 aardvark .....	19
5.2 Scenario format .....	19
5.2.1 Strigil/XML .....	19
5.2.2 Adaptation of Strigil/XML format .....	20
5.2.3 SOWL/JSON .....	21
<b>6 Results and Tests</b> .....	24
6.1 Data .....	24
6.1.1 Pamatky .....	24
<b>7 Conclusion</b> .....	25
<b>References</b> .....	26
<b>A Abbreviations</b> .....	27

## Tables / Figures

**2.1.** RDF and RDFS vocabulary .....7

**3.3.** Image of Selenium IDE ..... 14

# Chapter 1

## Introduction

During past few years the Web has undergone several bigger or smaller revolutions.

- WEB 2.0 and tag cloud
- HTML5 and semantic tags
- Smartphones, tablets, responsivity 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 technology booms and issues on the global network. So little can mean so much in such a global environment. The environment online is constantly changing, usually on a wave of some new, useful or frightening technology or with popularization of a new phenomena. The Semantic Web technologies have been described, standardized and implemented for several years now <sup>1)</sup> 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 originating from multiple sources. Once given the structure, similar data can be joined in a form of a bigger bulk. Presenting this data publicly creates a virtual cloud. This phenomena is called Linked Data.

In this work we'd like to bring the Semantic Web technologies closer to users. We will propose a methodology for extracting and annotating data out of unstructured web content, along with design and implementation of a tool, to simplify the process. Results will be confronted with real life use cases.

### 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 its 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 CSS annotation or HTML5 tags, contact information or opening hours, or even visualizing data from their own internal ontology, the Knowledge Graph <sup>2)</sup>.

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

What options do we have to bring semantic into a webpage?

<sup>1)</sup> One of the most recent standards – OWL2 – was released in 2008 [1]

<sup>2)</sup> [https://en.wikipedia.org/wiki/Google\\_Knowledge\\_Graph](https://en.wikipedia.org/wiki/Google_Knowledge_Graph)

One direction to go 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 have to use the right tool and put some time and effort giving the data the appropriate annotation. There are standards covering this use case. HTML5 brings in tags for clearer specification of the page structure (such as `nav`, `article`, `section`, `aside`, and others). Microformats <http://microformats.org/> define specialized values for HTML `class` attribute to bring standardized patterns for several basic use cases with fixed structure, such as *vCard* or *Event*. The microformat approach is easy to implement as it doesn't impose any extra syntax and can simply embed an existing page source. Last but not least, we can use RDFa to annotate data on a webpage with an actual ontology. This technology is part of the Semantic Web stack and we'll describe it closer in further chapter (TODO link).

Annotating data on the server side enables users to use tools to highlight data they are specifically interested in, extract them and reason on them. Services can use annotated data, combine them and offer results from multiple sources.

Some examples of utilities for extracting and testing or scraping structured data:

- <http://www.google.com/webmasters/tools/richsnippets>
- <http://rdfa.info/play/>
- <https://code.google.com/p/ldspider/>
- <http://ldodds.com/projects/slug/>

To bypass the gap between unstructured data present on the web on one side and rich, linked, meaningful ontologies on the other, we can go the opposite direction as well. We can take the unannotated data already present on the web and retrieve them in a form, that is defined by some ontology structure. This process can be performed either automatically or manually.

When coming to the automated crawling, we're mostly interested in improving ranking of search results. Ontology is used to help crawler to find relevant pages to a keyword or to finetune the ranking metrics <sup>1)</sup>.

While on well structured, simple and/or partially annotated pages this process can be very successful and produce useful results, on pages with unorganized data the confidence on results produced by this approach might drop to minimum. Unfortunately many online webpages and services are poorly structured. Pages containing many unrelated data, in form of advertisements or other external content might confuse such an engine. Old servers present their content in poorly structured or even invalid HTML usually in a form of multiple nested tables that serve for structuring only and makes the whole structure of the web unreadable. Social aspect of web brings in almost complete randomness making it even harder to automatically reason on page's content if it can't be distinguished and potentially left aside. We've mentioned few, both potential and real threads that prevent us from automatically annotate all data on web with confidence.

In some use cases the ontology of the desired data is yet to be created and the user is aware of the data structure and capable of manually spot and select the data on a web page. Currently there isn't many tools allowing this kind of operation. The ideal implementation and the vision here will allow user to partially identify the structure of a webpage while leaving the repetitive tedious work on crawler following the same procedure repeatedly on all data of the page.

<sup>1)</sup> [http://www.researchgate.net/publication/220830610\\_An\\_Ontology-Based\\_Crawler\\_for\\_the\\_Semantic\\_Web](http://www.researchgate.net/publication/220830610_An_Ontology-Based_Crawler_for_the_Semantic_Web)



For such a process we need to create tools that allow users to address previously unstructured content, link it to resources of existing ontology and/or create these resources on-the-go. By using existing ontologies we would not only give the meaning to our data, but also create 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. Crawler called crOWLer serves the needs of extracting data from web. It follows the workflow of scraping data using manually created scenario with given structure and user-defined set of ontological resources.

In previous 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

In the initial crOWLer implementation it is necessary to fulfill the first three steps with an actual programming. In order to perform this task, we need to have a programmer with knowledge of Java programming language, and several technologies used on the web, along with knowledge of the domain of data being scraped in order to correctly choose appropriate resources for annotation. There is also a huge overload in preparation of development environment and learning time of the crOWLer implementation. The need of more elegant and generic solution is evident.

## 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 RDF graph.

## 1.4 Specific goals of the project

- design the semantic data creation use-cases
- create syntax for scenario for crOWLer
- implement a web browser extension for creating scenario for crOWLer
- this extension shall
  - load and visualise ontology
  - join page structure and ontology resources in a form of scenarion
  - serialize scenario and necessary ontological data
- parse the scenario by crOWLer
- run crOWLer following the scenario
- visualize the extracted data (feedback)

## 1.5 Work structure

Next part of this work will cover tools and technologies related to the work. Chapter XXX will describe research on existing solutions and how they influenced results of this work. XXX program design. XXX program implementation. At the end we'll evaluate results of this work against proposed use cases.

## Chapter 2

# Knowledge base, principles and technologies

In following chapter I'll provide basic information about technologies of Semantic Web, and Knowledge Representation. The terminology often used in the field will be defined and used to help full understanding before we proceed to the design and implementation.

## 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) [2]. 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 [3]. 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 are well standardized and will be described in following sections of this chapter.

As a general logical concept of the technology, languages of Semantic Web are designed to describe data and metadata, give them unique identifiers – so that we can address them – and form them into oriented graphs. The metadata part define a schema of types (or classes) and properties that both can be assigned to data and also relations between this types and properties themselves. Wrapped together this metainformation is being presented 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.

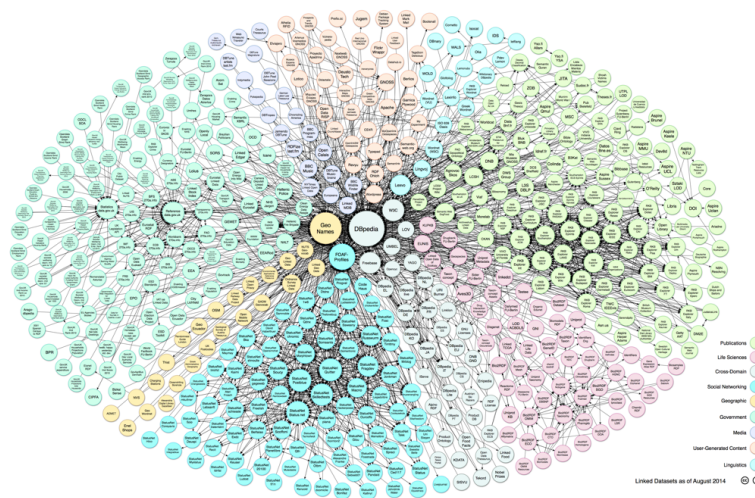
On low level of the implementation we deal with simple *oriented graph*. The graph structure is defined in a form of *triples*. Each triple consists of three parts: *subject*, *predicate* and *object*, which all are simply *resources* listed by their identifiers (URI's). In this very general form we can express basically any relationship between two resources. On a level of classes and properties, we can define hierarchies, or set a class as a domain of some property. On lower, more concrete level we can assign a type to an *individual*. On a level of ontologies, in a way a “meta-meta” level, we can specify for instance an author, description and date it was released. Each of the relations is described using triples and together form one complex graph.

## 2.2 Linked Data

Wikipedia defines Linked Data as “a term used to describe a recommended best practice for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs and RDF.” 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 tools and, of course, the data being

presented. Linked data effort strives to solve the problem of unreachability of majority of the knowledge present on the web, as it is not accessible in machine readable form, doing so by defining standards and supporting implementation of those standards.

To imagine current state of the Linked Data we can take a look on the Linking Open Data cloud diagram <sup>1)</sup>. The visualisation contains a node for each ontology and shows known connections between ontologies. The data originate from <http://datahub.io>, a popular web service for hosting semantic data. Current diagram visualises the state of linked data cloud in April 2014. As we can see in the center, many data resources are linked to dbpedia <sup>2)</sup>, the semantic data extracted from Wikipedia. This best describes the notion of Linked data. When two datasets relate to the same resource, they can be logically linked together through this connection, as this way they state, they relate to the same thing.



**Figure 2.1.** The Linking Open Data cloud diagram <sup>3)</sup>

Some additional resources on Linked Data:

- <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 [4].

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>). This literally specify address of

<sup>1)</sup> <http://lod-cloud.net>

<sup>2)</sup> <http://dbpedia.org>

resource and in many cases can be directly accessed in order to obtain the related data. In some cases we can use URN as well. URN as opposed to URL allow us to identify a resources without specifying it's location. This way we can for example use ISBN codes when working with books and records, or UUID <sup>1)</sup> a Universally Unique Identifier widely used to identify data instances of any kind.

### ■ 2.3.2 RDF and RDFS vocabulary

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

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. This is the most general standard that lays under every ontology out there.

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 <b>r</b>
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

## ■ 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, symetricity and other logical capabilities of properties. When querying an OWL ontology, it allow us to use unions or intercections of classes or cardinality of properties. All this capabilities comes in with well defined

<sup>1)</sup> [https://en.wikipedia.org/wiki/Uniform\\_resource\\_identifier](https://en.wikipedia.org/wiki/Uniform_resource_identifier)

semantics. Usage of each feature brought in by OWL semantics extends requirements on resolver being used for reasoning on our ontology and brings in necessary computational complexity.

Including some more readings on OWL:

- <http://www.w3.org/TR/owl2-primer/>
- [https://en.wikipedia.org/wiki/Web\\_Ontology\\_Language](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 a concept of embedding content of a web document defined in HTML with resources from some ontology. Technically we create an invisible layer of annotations over the data that turns our content into machine readable record. This is accomplished by embedding the original HTML with custom attributes. Tools can be used to visualise this data <sup>1)</sup>.

## 2.6 SPARQL

Is a semantic query language for data stored in RDF format [5]. Using SPARQL syntax we define a pattern of the RDF graph using triples and as a result we obtain such a nodes that form a subgraph of the original graph that match the given pattern. So called SPARQL endpoints are the main entry points through which users can obtain data from openly available datasets <sup>2)</sup><sup>3)</sup>.

Below you can see a simple example of a SPARQL query that returns list of all resources from database that have a `rdf:type` associated to it.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?target ?type
WHERE {
    ?target rdf:type ?type;
}
```

## 2.7 RDF/XML syntax

RDF/XML is one of formats into which we can serialize our RDF data <sup>4)</sup>. It is a regular XML document containing elements and attributes from the RDF(S) vocabulary. RDF/XML is one of the most common formats for RDF data serialization.

## 2.8 Turtle syntax

Turtle syntax is another popular syntax for expressing RDF <sup>5)</sup>. Its syntax suits more naturally to RDF data as it conforms the triple pattern.

<sup>1)</sup> <http://rdfa.info/play/>

<sup>2)</sup> <http://dbpedia.org/sparql> Dbpedia SPARQL endpoint

<sup>3)</sup> <http://linkedgeo.org/sparql> LinkedGeoData SPARQL endpoint

<sup>4)</sup> <https://en.wikipedia.org/wiki/RDF/XML>

<sup>5)</sup> [https://en.wikipedia.org/wiki/Turtle\\_\(syntax\)](https://en.wikipedia.org/wiki/Turtle_(syntax))

## Chapter 3

### Existing solutions

In this chapter we'll describe the research made on existing solutions for given task. The performed search was focused on tools directly solving the given problem (annotating data on web and crawling it), as well as libraries and technologies that would help to implement new solution or existing open source programs we could build the solution on.

#### 3.1 Semantic and non semantic crawlers

By researching existing solutions, there is currently no open source or openly available solution that would directly follow the required workflow and fulfill the requirements.

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 a triple store is used as the database in previous case we obtain one-time solution to our problem. This is technically equal to original state of crOWLer.

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 in naturally without any extra effort.

Last for benefits: using ontology from public resource as a schema for our data can give us correct structure without need for 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.

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.

For instance There is always a threat of inconsistency of an ontology when some data don't fit the rules or breaks structure of an ontology. In it's state from April 2014 DBpedia states, there is 3.64 million resources, out of which 1.83 million are clasified in a consistent Ontology [6]. That is only half of the data being arguably consistent with each other. That doesn't say the rest is bad. Only that it might cause a inconsistency and prevent us from reasoning if we include wrong subset of the data.

Just like with "hardcoded" crawling technique, the semantic crawling is tightly bound to the structure of the web being crawled. The web is being matched against some pattered described by selecors and the matching elemented, when found, are accepted for further processing. Any change on a webpage structure can lead to broken selectors or links during the crawling process and make the scenario invalid.

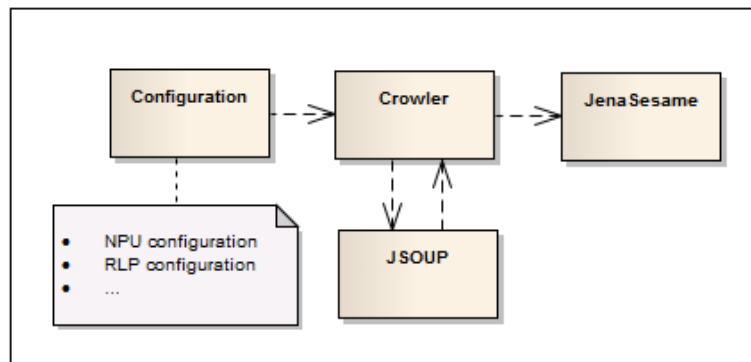
A lot of web pages loads their data dynamically using AJAX queries. Some pages simply changes it's content frequently news pages, forums, user content pages and social web applications. Crawling content on such servers would require almost constant crawling and would cause growth into massive ontology of questionable quality.

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 Analysis of crOWLer

In this section an analysis of existing impementation of crOWLer is described.

Original implementation is prototype of console Java aplication. It uses JenaSesame library for handling ontological data and JSOUP library for accessing webpages and adressing elements. As a scenerio crOWLer accepts java `.class` files containing a implementation of `ConfigurationFactory` class. This configuration specifies all the information needed for crawling process:



**Figure 3.1.** General architecture of the original crOWLer implementation.



- webpage to be crawled
- way to address data on that page using JSOUP selectors
- definition of ontology resources used to anotate the obtained data
- and definition of pagination process that brings us to next page to be crawled

Additionally the pagination and selector implementation are supported by several helper classes for chaining selectors or generation of a list of URL addresses by incrementing specific argument.

This reveals the issue being addressed. Java implemented configuration requires knowledge of Java programming language along with knowledge of RDF technologies. Programmer gets into the position of ontological engineer when designing new resources. Knowledge of WEB technologies is needed in order to properly target elements on the webpage using JSOUP selectors. This is one of the hardest task as the selectors have to be manually extracted using for example browser console.

Following code is an exapmle of actual confuguration code of original crOWler implementation. It uses NPU class as simple static storage for URI's used in our ontology. It creates a *monumentRecord* object for each talbe row as defined by the “initial definition”. The second part create *district* object with it's label (found in third table colum denoted by the `td:eq(2)` selector) and assigns it to the record using *hasDistrict* object property. The `conf` object holds the configuration being passed to the actual crawler.

```
ClassSpec chObject = Factory.createClassSpec(NPU.monumnetRecord.getURI());

conf.addInitialDefinition(
    Factory.createInitialDefinition(
        chObject,
        Factory.createJSoupSelector("table tbody tr.list")));

ClassSpec sDistrict = Factory.createClassSpec(NPU.district.getURI());
chObject.addSpec(
    Factory.createOPSpec(
        Factory.createJSoupSelector("td:eq(2)"),
        NPU.hasDistrict.getURI(),
        sDistrict));
sDistrict.addSpec(true, Factory.createDPSpec(Vocabulary.RDFS_LABEL));
```

Previous example more or less defines requirements on scenario for semantic crawler. To fully satisfy the crOWLers current implementation, we would also have to cover following hyperlinks on a page, firing javascript and browser events and functions of transforming scraped data using for example regular expressions or key-value mapping.

## 3.4 Finding platform for frontend

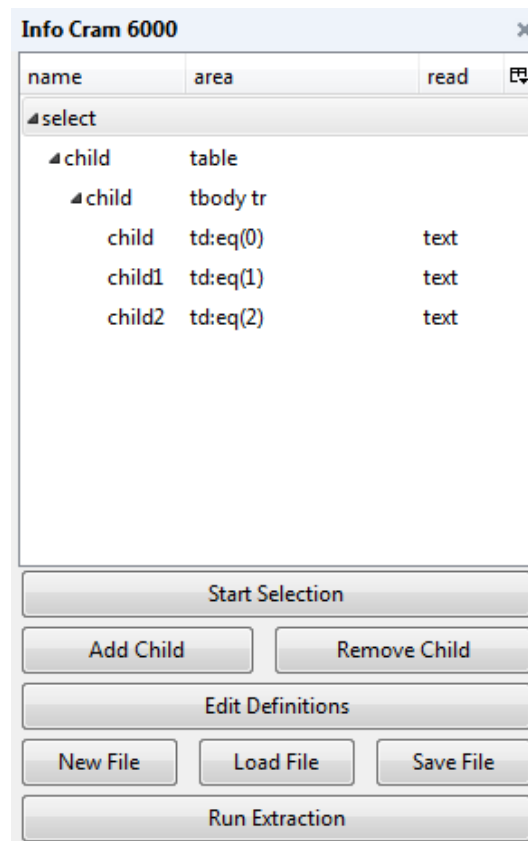
In order to develop appropriate tool for generating scenarios, several similar tools were inspected for best practices, libraries, and possible extension.

The resulted implementation is named SOWL (short for SelectOWL) and refers to Firefox addon for creating scenarios for crOWler. In following sections we'll refer to SOWL as set of requirements and a envisioned expected result of this work. The actual implementation will be covered in following chapters.

### 3.4.1 InfoCram 6000 – ExtBrain

InfoCram 6000 is part of project ExtBrain <sup>1)</sup> that is developed here on Department of Computer Science. This specific part was implemented by Ing. Jiří Mašek and is described as “prototype of user interface for visual definition of extraction rules for ExtBrain Extractor”. It’s intended usage is very close to the usage of SOWL. It is an Firefox extension that generates rules (scenario) for extractor implemented as another part of the ExtBrain project.

The ExtBrain extractor is implemented in javascript as opposed to Java in case of crOWler. It extracts data according to definitions by InfoCram 6000. The result is stored in JSON format thus not carrying semantic information, but only set of raw data in some form.



**Figure 3.2.** Main window of InfoCram 6000

Main part of the extension window shows a tree view with rules being edited. This view corresponds to required structure of scenario for crOWler.

Interesting part is an engine for selection elements of page. It’s implementation is based on Aardvark <sup>2)</sup>, a Firefox extension that addresses this issue using mouse selection and several keyboard commands.

InfoCram doesn’t use simple CSS or XPath selectors, but include Sizzle library to handle selectors for it. Sizzle is very popular library for handling selectors, which also defines it’s own selectors like `:eq()`, or `:first`. It’s simpler and more expressive than CSS. It’s popularity is mainly based on it’s involvement in jQuery library.

Being so close to required structure and workflow of SOWL, InfoCram 6000 served as the base implementation for it in the early stages. As can be seen at the end of this

<sup>1)</sup> <http://www.extbrain.net>

<sup>2)</sup> <https://addons.mozilla.org/en-US/firefox/addon/aardvark/>

chapter, the first implementation named SelectOWL carries similar user interface and make use of several modules of the InfoCram implementation.

### ■ 3.4.2 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 scenario 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 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 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.

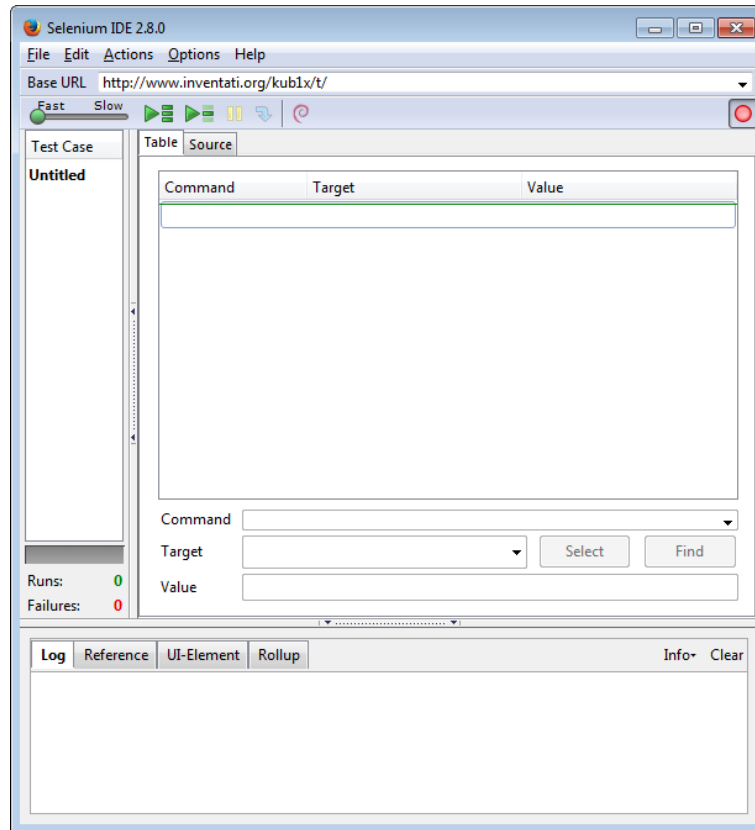
An attempt was made to implement SOWL as a plugin for Selenium IDE. This plugin would have two parts:

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 3.3 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



**Figure 3.3.** GUI of Selenium IDE showing the Command, Target and Value fields.

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 commadn 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. Accesor 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 commad 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`)<sup>1)</sup> 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)

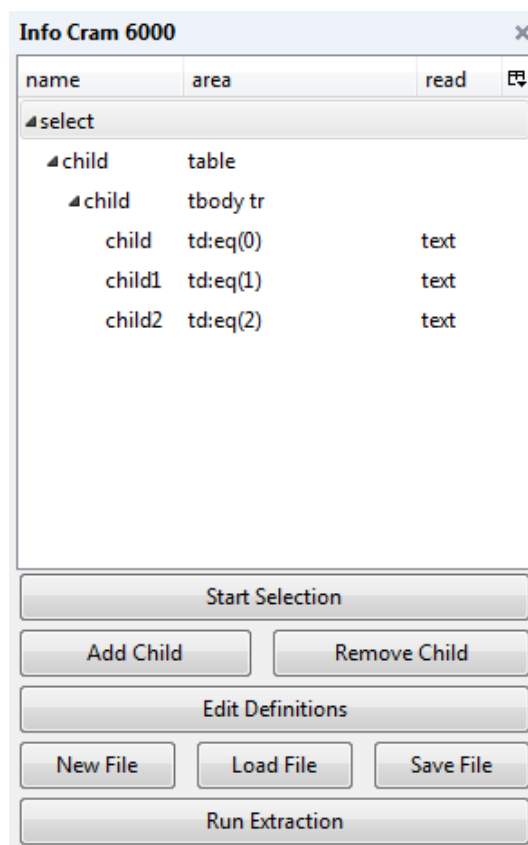
### 3.5.2 Architecture of Strigil platform

### 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.

## 3.6 Early implementation

<sup>1)</sup> <https://code.google.com/p/selenium/source/browse/ide/main/src/content/commandBuilders.js> the CommandBuilder implementation



**Figure 3.4.** Main window of InfoCram 6000

## 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.

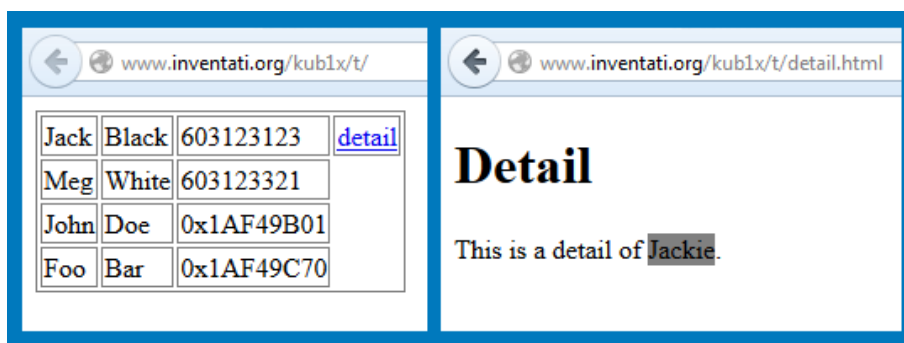


Figure 4.1. Example main page and detail page for the basic Use Case.

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

MonumNet

Nemovitá památky

pro tisk: stránka celý výběr do Excelu: stránka celý výběr

Nalezeno: 40203 je chráněno, přírůstky od 03.05.1958 do 10.12.2013

Stránka 1 / 1609 ↩ ↪

1 2 3 4 5 6 7 8 9 10 11

Číslo rejstříku	uz	Název okresu	Státní útvar	Část obce	čp.	Památky	Ulice,nám./u
20339 / 1-1971	5	Praha hl.m.	Praha	Běchovice	čp.1	zájezdní hostinec Na Staré poště	Praha 9, Českokobrodská
104764	5	Praha hl.m.	Praha	Benice		zvonice	
40604 / 1-1569	5	Praha hl.m.	Praha	Bohnice		kostel sv. Petra a Pavla	Praha 8, Bohnice
54973 / 1-1628	5	Praha hl.m.	Praha	Bohnice		výšinné opevněné sídliště - hradiště Zámka, archeologické stopy	Praha 8, na ostrohu nad Vltavou
54974 / 1-1571	5	Praha hl.m.	Praha	Bohnice	čp.1	venkovská usedlost Vraných	Praha 8, Bohnická
44366 / 1-1572	5	Praha hl.m.	Praha	Bohnice	čp.4	fara	Praha 8, Bohnická
54975 / 1-1573	5	Praha hl.m.	Praha	Bohnice	čp.12	čínžovní dům - hospoda Štrasburk	Praha 8, Bohnická
40605 / 1-1570	5	Praha hl.m.	Praha	Bohnice	čp.91	nemocnice - psychiatrická léčebna	Praha 8, Ústavní, Bohnická
44368 / 1-1347	5	Praha hl.m.	Praha	Braník		kostel sv. Prokopa	Praha 4, Školní, Nad koletem
44369 / 1-1713	5	Praha hl.m.	Praha	Braník	čp.15	Maroldova vila	Praha 4, Stará cesta

Figure 4.2. Partial view at data on National Heritage Institute's webpage.

## 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/>

##### 5.1.2 aardvark

Aardvark is a javascript engine for in place modifications of a webpage. It allows user to select, delete , or highlight part of HTML page. In its production version a

#### 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 <sup>1)</sup>.

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

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

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 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 guaranteed 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` command 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 functions 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 development 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 Adaptation of Strigil/XML format

Strigil creates it's scraping script internally hidden under GUI and leaves user unaware of it's actual content. It might still serve well, at least for developers, to keep the script compact and easily readable. Addition of language tag as seen in previous chapter, is

widely used pattern that pollutes the resulting script with unnecessary overload. Suggested improvement would separate this functionality into an extra attribute of the `value-of` tag named `lang`.

The same suggestion can be applied to the data-type specification. Moreover *implicit parsing* of known datatypes would not only simplify the scraping script, but also help to clean and clear the resulting data.

Let's imagine hypothetical scenario of two similar tables on one page containing two sets of data in the same format. For such a case we would need to define a template on subset of DOM and call it twice with different root node. Creation of `dom-template` and `call-dom-template` tags would solve this issue and would allow scenario creator to narrow down his focus to a subpart of the scraped webpage. This would be particularly useful on complicated pages with a lot of nested HTML. `dom-template` and `call-dom-template` would be defined within a single `template` tag and unlike `call-template`, they would *keep* the ontological context of call of `value-of` within `dom-template` would assign a property to individual created by `onto-elem` wrapping the current `call-dom-template` call.

The architecture of Strigil (distributed downloader) suggests that it uses simple raw HTML pages as they were downloaded and uses JSOUP to extract data from it as JSOUP is the selector system of choice. Many webpages, or even web applications, make use of dynamic AJAX calls to fetch additional data after the presentation layer of the web is shown to the user. Strigil doesn't handle these cases by default. The internal AJAX code could be analyzed and simulated using `call-template` call, but this requires deep knowledge of the webpage being processed. In crOWler we opted to switch from JSOUP to WebDriver library and use PhantomJs, a no-GUI web browser. This technology allows us to handle webpages the same way as user sees them.

Usage of actual full-stack web browser with javascript engine along with WebDriver allows us to inject and execute arbitrary javascript code into the processed webpage. In order to make full use of this feature we can define `function-def` tag which would define javascript function with name and params and contain its code. To execute this function we would call `function-call` and identify it by its name. Return value of this function can be then used the same way as the one from `value-of` tag.

From the experience with development on Strigil/XML we can derive, that it is tied with its intended use for distributed downloader and it lacks some functionality. In SOWL we would almost necessarily modify its formal definition and thus it is of consideration if we can't make use of more appropriate format.

### ■ 5.2.3 SOWL/JSON

As all Firefox extensions, SOWL is written entirely using javascript with additional HTML defining the graphical layout. Early stages of implementation generated XML based on Strigil/XML format using hardcoded XML snippets and string formatting – approach often used on webpages with dynamically loaded content. A string holds a snippet of HTML or XML structure with placeholder. This placeholder is replaced by either a value or by another already processed snippet. This way piece by piece the whole scenario is generated. This solution isn't hard to implement, but brings in poor maintainability and with additional complexity it loses elegance, readability and can even cause performance issues.

Original data of the scenario created by SOWL are stored naturally in javascript object. Using standard javascript method `JSON.stringify()` we can immediately generate JSON serialization of such object. This way we have structure similar to the

original defined by Strigil/XML, but in flexible structure. Obviously some adaptations are necessary. Nesting is recorded using the **steps**, the header section is redesigned for the JSON structure. XXX

The original semantics of **onto-elem** and **value-of** was preserved, only limited to it's basic use. **value-of** serves solely to assign literal properties.

The final scenario for Use Case 1 XXX looks like this:

```
{
  type: "scenario",
  name: "manual",
  ontology: {
    base: "http://kub1x.org/onto/dip/t/",
    imports : [
      {
        prefix: "foaf",
        uri: "http://xmlns.com/foaf/0.1/",
      },
      {
        prefix: "kbx",
        uri: "http://kub1x.org/onto/dip/t/",
      },
    ],
  },
  creation-date: "2014-11-30 12:40",
  call-template: {
    command: "call-template",
    name: "init",
    url: "http://www.inventati.org/kub1x/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",
      selector: {
        value: [
          {
            value: "td.detail a",
            type: "css",
          },
          {
            value: "@href",
            type: "xpath",
          },
        ],
        type: "chained",
      },
    },
  ],
},
],
},
],
},
{
  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





## References

- [1] *Web Ontology Language – Wikipedia.*  
[https://en.wikipedia.org/wiki/Web\\_Ontology\\_Language](https://en.wikipedia.org/wiki/Web_Ontology_Language).
- [2] *Semantic Web – Wikipedia.*  
[https://en.wikipedia.org/wiki/Semantic\\_Web](https://en.wikipedia.org/wiki/Semantic_Web).
- [3] *Semantic Web – W3C.*  
<http://www.w3.org/standards/semanticweb/>.
- [4] *SPARQL Protocol and RDF Query Language – Wikipedia.*  
<https://en.wikipedia.org/wiki/SPARQL>.
- [5]
- [6]





## 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