

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
Artificial Intelligence
podlajak@fel.cvut.cz

January 2015
Supervisor: Ing. Petr Křemen, Ph.D.

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 předloženou práci vypracoval samostatně a že jsem uvedl veškeré informační zdroje v souladu s Metodickm pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 5.1.2015

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 Mo- tivation	1
1.2 Use Cases	3
1.2.1 Use Case 1 – basic ex- ample case.....	3
1.2.2 Use Case 2 – National Heritage Institute	4
1.2.3 Use Case 3 – Air Ac- cidents Investigation Institute	6
1.2.4 Use Case 4 – National Transportation Safety Board.....	8
1.3 Current solution crOWLer	9
1.4 Proposed Solution and Methodology	10
1.5 Specific goals of the thesis	10
1.6 Work structure	10
2 Principles and technologies	11
2.1 Technology of Semantic Web..	11
2.2 Linked Data	12
2.3 RDF and RDFS.....	12
2.3.1 URI.....	13
2.3.2 RDF and RDFS vo- cabulary	13
2.4 OWL	13
2.5 RDFa	13
2.6 SPARQL	14
2.7 RDF/XML syntax	14
2.8 Turtle syntax.....	14
3 Existing solutions	15
3.1 Semantic and non semantic crawlers.....	15
3.1.1 Advantages and pit- falls of Semantic crawlers	15
3.2 Analysis of crOWLer	16
3.3 Strigil.....	18
3.3.1 What problem does it solve?	18
3.3.2 Architecture of Strigil platform	19
3.3.3 What inspiration it brings for crOWLer	20
3.4 Finding platform for frontend ..	20
3.4.1 InfoCram 6000 – ExtBrain	20
3.4.2 Selenium	21
3.5 Libraries for crOWLer	24
3.5.1 WebDriver.....	24
3.5.2 Apache Jena	24
3.6 Libraries for SOWL.....	24
3.6.1 jQuery	24
3.6.2 jOWL.....	24
3.6.3 rdfQuery	24
3.6.4 aardvark	25
3.7 Early implementation.....	25
4 Program design	26
4.1 Workflow	26
4.1.1 Main line	26
4.1.2 Scenario creation.....	26
4.1.3 Additional branches to Scenario Creation	26
4.1.4 crOWLer scraping	27
4.2 User Interface	27
4.2.1 SOWL user interface....	27
4.2.2 crOWLer user interface..	27
4.3 Model.....	27
4.3.1 SOWL model.....	27
4.3.2 crOWLer model	28
5 Program Implementation	29
5.1 Parsing Ontologies in JavaScript	29
5.2 Scenario format	29
5.2.1 Strigil/XML	30
5.2.2 Adaptation of Strig- il/XML format	31
5.2.3 SOWL/JSON	31
5.2.4 Consequences of con- version to JSON format .	32
5.3 SOWL implementation	32
5.4 crOWLer implementation	32
5.4.1 architecture	32
5.4.2 JavaScript and events support	33
5.4.3 Implemented and unimplemented capa- bilities	35
6 Results and Tests	36
7 Conclusion	37
References	38

A Assignment	42
B Abbreviations	43
C RDF and RDFS vocabulary	44
D Example of RDF/XML syntax ..	45
E Configuration component of original crOWLer	46
F Selector component of original crOWLer	47
G crOWLer architecture.....	48
H SOWL/JSON scenario solving Use Case 1.....	49

Tables / Figures

C.1. RDF and RDFS vocabulary ...	44
1.1. A screenshot of an example main and detail page for the basic use case.....	3
1.2. An activity diagram of the general workflow of the stack	4
1.3. Partial view at data on National Heritage Institute webpage.....	5
1.4. Preview of HTML analysis on National Heritage Institute webpage	5
1.5. Partial view at data on National Heritage Institute webpage.....	6
2.1. Logo of Semantic Web	11
3.1. General architecture of the original crOWLer implementation	16
3.2. Core classes of original crOWLer implementation.....	17
3.3. Overall Architecture of Strigil .	19
3.4. Components of Data Application part of Strigil	19
3.5. Components of Download System part of Strigil	19
3.6. Example deployment structure of Strigil	20
3.8. Image of Selenium IDE	23
5.1. Overview of the whole stack and files exchanged.....	29

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
- Smart-phones, 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 is only several 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, sometimes, terrifying technology or with popularization of a new phenomena. The Semantic Web technologies have been described, standardized and implemented for several years now¹⁾ 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. When put together this practices are called the Linked Data.

The intention of this work is to bring the Semantic Web technologies closer to users. Specifically it focuses on the process of creation of semantic data. We will propose a methodology for extracting and annotating data out of unstructured web content, along with a design of a tool, to simplify the process. The design will be supported by implementation of a prototype of the tool. 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. When possible, Google presents not only the list of pages corresponding to the searched term but also snippets of information scraped directly from the content of the pages such as menu fields parsed from CSS annotation or HTML5 tags, contact information or opening hours. When applicable Google also adds data from their own internal ontology, the Knowledge Graph[2].

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

What options are there to bring real semantic into a webpage? One direction to go is to annotate data on “the server side”, i.e. at the time it is being created and/or published. When we are in position of owner of the data or server, we can help not only Google or DuckDuckGo¹⁾ to understand our website. To avoid confusion, this part is not focused on SEO – the search engine optimization[3], even though the topic overlaps in many ways. SEO primarily focuses on increasing the ranking of a webpage in the eyes of a search engine whereas pure semantization focuses on best describing the meaning of the pages content no matter how good or bad it appeals to anyone as long as it is valid according to standards of Semantic Web[4] and Linked Data[5].

In order to perform semantization on the server side, 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.

In the simplest form the HTML5[6] brings in tags for clearer specification of the page structure (such as `nav`, `article`, `section`, `aside`, and others).

Microformats[7] 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 does not impose any extra syntax and can simply embed an existing page source. As the community around microformats states “(Microformats are) designed for humans first and machines second.”

Last but not least, we can use joined power of HTML and RDFa [8] to annotate data on a webpage with an actual ontology. This technology is part of the Semantic Web stack and we will describe it closer in next chapter 2.

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 new results based on merged knowledge obtained from multiple sources. Providing data in such a form makes server a part of Linked Data cloud. For completeness let us mention some examples of utilities for extracting and testing or scraping structured data:

- Google Structured Data Testing Tool (i.e. rich snippets) [9]
- RDFa Play – tool for visualisation and extraction of RDFa content [10]
- LDSpider – a semantic data crawler [11]

Unfortunately, it is not always possible or desired by the web owner to embed semantics into their data and support it. Vast majority of the web holds plain text data without any machine readable meaning given to it, leaving it on human readers to understand it.

To bypass the gap between unstructured data present on the web on one side and rich, linked, meaningful ontologies on the other, we can take the opposite direction to the one described so far. We can take the unannotated data already present on the web and retrieve them in a form, defined by some ontology structure.

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 to manually spot and select the data on a web page. Currently there are not many tools allowing this kind of operation. The ideal implementation and the vision of result of this thesis 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.

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 re-

¹⁾ An anonymous search alternative to Google <http://duckduckgo.com>

sources 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 Use Cases

In a general case our goal is to “obtain data from a webpage in a semantic form.” We have a webpage and optionally an ontology as an input and annotated set of data as an output.

For start, we will focus on data having a structure defined in HTML. The data might be structured as a table or set of paragraphs or any other set of HTML tags, and we will handle it on the level of these tags. Some text handling might be performed using regular expressions but usually we will simply select a HTML tag and use its content along with some annotation.

In the “friendliest” cases the data we want to scrape are formed in some repetitive form, most often a table. This is the best case as we can simply define structure on one row of the table and repeat the same pattern over and over. Sometimes the table spreads over several pages, so we need to define a way of advancing to the next page and start over.

Following sections contain description of several use cases that shall be solvable using design proposed in this thesis.

1.2.1 Use Case 1 – basic example case

<http://www.inventati.org/kub1x/t/>

The first use case is the simplest task that will be covered by the implemented prototype. As you can see on the picture 1.1 it consists of table holding values about people, and link to a detail page for one of them. On the detail page there is a field with “nickname”.

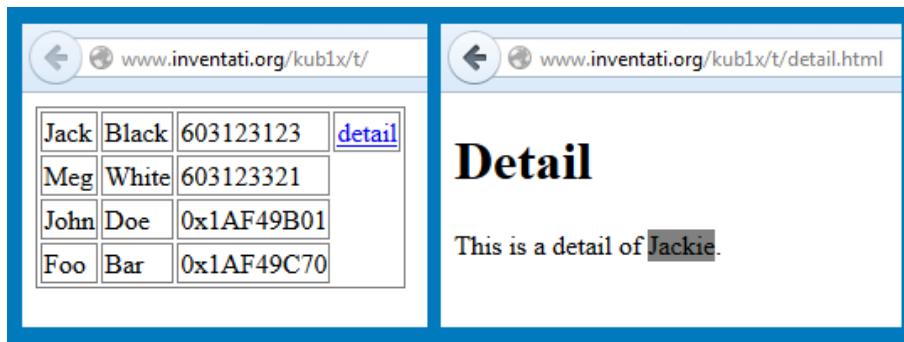


Figure 1.1. The example main page and detail page for the basic use case.

In order to fulfill this use case SOWL shall support following operation:

- Load the FOAF ontology that contains resources to describe data about people.
- Create scenario with two templates: init and detail.
- Save this scenario to a file.

CrOWLer shall be able to perform following tasks.

- Accept and parse scenario created by SOWL.
- Follow this scenario while scraping data from the page.

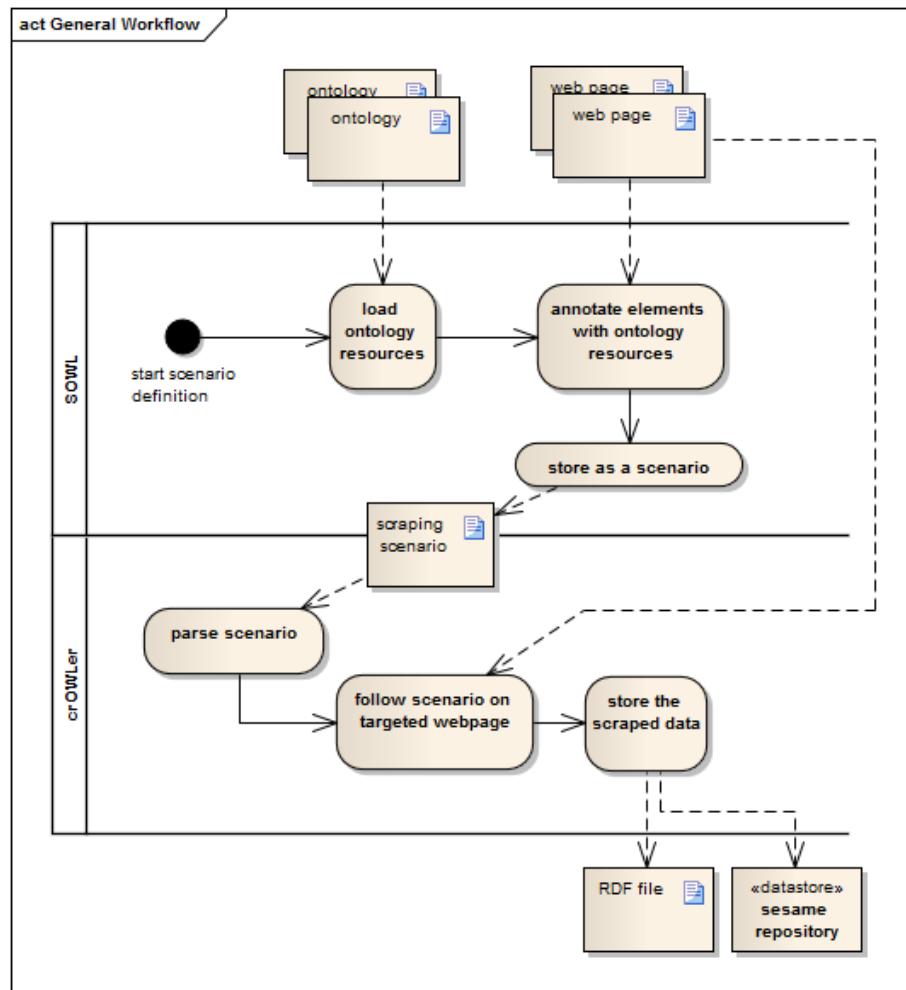


Figure 1.2. Diagram of general workflow as derived from presented use case

- Store results into RDF files.

This use case defines the simplest functionality that have to be implemented by both programs. It covers resources handling, scenario creation and running and finally storage of the results. It helps to define the proper behavior of the program as it's written in simple, valid HTML5 code without any JavaScript and all elements can be simply targeted by CSS or XPath selectors.

■ 1.2.2 Use Case 2 – National Heritage Institute

<http://monumnet.npu.cz/pamfond/hledani.php>

The webpage of National Heritage Institute of Czech Republic 1.3 gives a public access to a table of damages of national monuments. This is of interest for project MONDIS¹⁾ partially developed on our school. Its main purpose is a documentation and analysis of damages and failures of cultural heritage objects.

The data were successfully crawled by the original implementation of crOWLer. The goal of following development is to replicate the behavior with new implementation using scenario driven crawling process instead of process driven by hardcoded configuration.

¹⁾ <https://mondis.cz>

MonumNet						Nemovité památky	pro tisk: stránka celý výběr	do Excelu: stránka celý výběr
Nalezeno: 40203 je chráněno, příručky od 03.05.1958 do 10.12.2013								
Stránka 1 / 1609 ⇠ ⇢						1 2 3 4 5 6 7 8 9 10 11		
Cílej režistru	Uz	Název okresu	Sídlení útvar	Cást obce	čp.	Památka	Ulice,nám./území	
20339 / 1-1971	■	Praha hl.m.	Praha	Běchovice	čp.1	zájezdni hostinec Na Staré poště	Praha 9, Českobrodská	
104764	■	Praha hl.m.	Praha	Benice		zvonice		
40604 / 1-1569	■	Praha hl.m.	Praha	Bohnice		kostel sv. Petra a Pavla	Praha 8, Bohnice	
54973 / 1-1628	■	Praha hl.m.	Praha	Bohnice		výšinné opevněné sídliště - hradiště Zámka, archeologické stopy	Praha 8, na ostrově nad Vltavou	
54974 / 1-1571	■	Praha hl.m.	Praha	Bohnice	čp.1	venkovská usedlost Vraných	Praha 8, Bohnická	
44366 / 1-1572	■	Praha hl.m.	Praha	Bohnice	čp.4	fara	Praha 8, Bohnická	
54975 / 1-1573	■	Praha hl.m.	Praha	Bohnice	čp.12	činžovní dům - hospoda Štrasburk	Praha 8, Bohnická	
40605 / 1-1570	■	Praha hl.m.	Praha	Bohnice	čp.91	nemocnice - psychiatrická léčebna	Praha 8, Ústavní, Bohnická	
44368 / 1-1347	■	Praha hl.m.	Praha	Braník		kostel sv. Prokopa	Praha 4, Školní, Nad kostelem	
44369 / 1-1713	■	Praha hl.m.	Praha	Braník	čp.15	Maroldova vila	Praha 4, Stará cesta	

Figure 1.3. Partial view at data on National Heritage Institute webpage

The main challenge of this use case lays in JavaScript. Each row of the data table has the `onclick` attribute defined. Unlike the classical “link” (also known as the anchor or `a` tag) the `onclick` attribute does not contain URL, but rather a JavaScript function content, that handles the click event. After closer investigation 1.4 we can observe that in this case, the function advances to the detail page of the clicked record by modifying a value of a hidden `input` tag and by submitting a `form` parametrized by the value.

```
> <table width="100%"></table>
> <table width="100%" border="0"></table>
> <table class="list" cellspacing="1" cellpadding="2" border="0">
>   <form action="list.php" name="listpf" method="GET"></form>
>   <input type="HIDDEN" name="IdReg" value="131164"></input>
>   <input type="HIDDEN" name="Uz" value="B"></input>
>   <input type="HIDDEN" name="PrirUbytOd" value="03.05.1958"></input>
>   <input type="HIDDEN" name="PrirUbytDo" value="02.01.2015"></input>
>   <input type="HIDDEN" name="Limit" value="25"></input>
> <tbody>
>   <tr></tr>
>   <tr class="list" onclick="document.listpf.IdReg.value='131164'; document.listpf.submit(); onmouseout="this.className='list'">
>     <td class="list" align="left"></td>
>     <td class="list"></td>
```

Figure 1.4. A preview of HTML source analysis on National Heritage Institute webpage

If possible, we would simply simulate the user “click” action to advance to the detail page and the “back” action (usually performed by the Back button of browser or Alt+left keyboard shortcut) to get back and follow on next line. This approach will be analyzed further in this work.

If the stated approach can't be implemented to give the expected results the original approach will be simulated by the new scenario driven structure. This means crOWLer will be getting the content of the `onclick` attribute, parsing it using regular expression and combining it with a predefined pattern into an URL to be directly called using `call-template`.

Additionally this use case hides one more pitfall that, this time, challenges the selector creation. The web page uses JavaScript to colorize table rows when user hovers them with mouse cursor. Using a deeper analysis, we can figure out, that table lines are given additional CSS on certain mouse events. This is often a sign of poor web practices as the same behavior can be achieved by `:hover` CSS selector without a need of additional class, but it is an example of a challenge that our tool need to overcome. In this very case, we probably won't be able to generate selectors using CSS classes and will rely only on tag names, positions and other identifiers, if applicable.

Additional requirement on SOWL to those in Use Case 1 1.2.1:

- allow manual resources creation
- record the `click` event

Nemovit  pam tky
pro tisk: str nka cel  v yb r do Excelu: str nka

��lo rejestruku	uz	N��ev okresu	Sideln� utvar	��st obce	Pam�tka	Ulice,n��m./um�st�n�	Cor	HZ	R	F	IdReg
40604 / 1-1569		Praha hl.m.	Praha	Bohnice	kostel sv. Petra a Pavla	Praha 8, Bohnice					152682

Pam tka : kostel sv. Petra a Pavla
Ochrana stav/typ uzav ren  : zaps n  do st tn ho seznamu p ed r.1988
Pam tka od : 3.5.1958
  lo rejestruku USKP : 40604/1-1569
N  ev okresu : Praha hl.m.
Sideln  utvar (m sto/ves) : Praha
  st obce : Bohnice
Katastráln   zem  : Bohnice
Ulice,n  m./um st n  : Praha 8, Bohnice
  lo popisn  :
  lo orienta n  :
M  stsk    st : Praha 8
Stavebn    rad : Stavebn    rad -   rad m  stsk    sti Praha 8
Finan n    rad : Finan n    rad pro hl  avn  m  sto Prahu,   zem n  pracovi t  pro Praha 8
Historick  zem  :   echy
Identifik tor z  namu (IdReg) : 152682

Parcely:

f.	parc.	dil	%pl.	omezen� pam�tkov� ochrany:	specifikace/pozn�mka
				Katastráln� �zem�: Bohnice	
	1	100			
	2	100			h��bitov, ohradn� ze� s bran�kou, schodi�t�

Figure 1.5. View on detail page on National Heritage Institute webpage.

- OR
- access the `onclick` attribute
- enable string handling using regular expressions
- record a `call-template` on the resulting URL

Additional requirements on crOWLER

- simulate the `click` event
- OR
- handle the attribute according to the string filters
- do `call-template` on the result as URL

The outcome of this use case and its analysis brings an important message. In many cases we will have to dive into the implementation of the processed webpage to find out how it behaves. In a vast majority of these cases it will require a web developer or coder to correctly and exhaustively define the scraping scenario.

■ 1.2.3 Use Case 3 – Air Accidents Investigation Institute

http://www.uzpln.cz/cs/ln_incident

This is basic use case with a table, a detail page and a pagination. Everything is present in a clear HTML form without any interruption by JavaScript.

In this case we might consider replacing repetitive values by an object instance carrying the information. For example the table shows column “Event type” (in Czech original: “Druh ud  losti”). It contains constant values of “Incident”, “Flight accident” and several more. A resource can be created to denote these types of accidents. The resource corresponding to the string scraped from table would than be used as a value of object property instead of the original string literal. The original literal is assigned to this resource as a “label”.

For example we can use (in turtle syntax 2.8):

```

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
@prefix rlp: <http://kub1x.org/dip/rlp#>
<rlp:event-xFuHbjA5> a <rlp:event>;
    <rlp:hasEventType> <rlp:flightAccident>.

<rlp:flightAccident> <rdfs:label> "Letecká nehoda"@cs.

```

Instead of:

```

@prefix rlp: <http://kub1x.org/dip/rlp#>
<rlp:event-xFuHbjA5> a <rlp:event>;
    <rlp:hasEventType> "Letecká nehoda"@cs.

```

Motivation for the previous instantiation lays in the following use case. As it uses the same domain – flight accidents – it might use some of the resources previously defined here. For the *event type* it would probably use exactly the same instances, and would only add the English label to them.

This should not be much of a problem as long as we can specify an URI identifier when creating an instance of an ontological object. In the example above the identifier is: <rlp:flightAccident>. Another identifier in the example is the URI of the event: <rlp:event-xFuHbjA5>. This one was chosen from an URL of a PDF file on the page.

From previous paragraph, we define another useful functionality: conditioning on string literals and specifying URIs of instances directly in the scenario either as a constant string or obtained by combining other string values probably in a form of a pattern.

Zprávy o LN a Incidentech

Zobrazit podle kategorie

[Vše](#) | [<2250 kg](#) | [2250 – 5700 kg](#) | [>5700 kg](#) | [Letouny](#) | [Vrtulníky](#) | [Kluzáky](#) | [Sportovní letecká zařízení](#) | [Para](#) | [Balóny a Vzducholodě](#)

Zobrazit podle data

Od: Do: Podle data události ▾ [>>](#)

Funkci pro zobrazení dle data vložení události lze použít od 1 května 2012.

Datum události	Druh zprávy	Místo události	Druh události	Druh provozu	
2014-11-02	Závěrečná zpráva	Bukovice	Letecká nehoda	Ostatní	více
2014-08-23	Závěrečná zpráva	Racková	Letecká nehoda	Rekreační a sportovní létání	více
2014-08-20	Závěrečná zpráva	LKPL	Letecká nehoda	Ostatní	více
2014-08-10	Závěrečná zpráva	LKHB	Letecká nehoda	Ostatní	více
2014-07-27	Závěrečná zpráva	LKPS	Letecká nehoda	Ostatní	více
2014-07-26	Závěrečná zpráva	LKCH	Letecká nehoda	Ostatní	více
2014-07-25	Závěrečná zpráva	LKMB	Letecká nehoda	Ostatní	více
2014-07-19	Závěrečná zpráva	LKKM	Letecká nehoda	Ostatní	více
2014-07-06	Závěrečná zpráva	Kunčice pod Ondřejníkem	Letecká nehoda	Ostatní	více
2014-07-06	Závěrečná zpráva	LKPM	Letecká nehoda	Ostatní	více

> [>>](#)

Figure 1.6. View on list page on Air Accidents Investigation Institute.

- specifying a pattern for creation of URI of each instance
- adding language tag to all string values
- possible usage of geographical ontology
- possible usage of enumeration

Průvodní formulář k předběžné a závěrečné zprávě

Datum události:	2014-11-02
Druh zprávy:	Závěrečná zpráva
Místo události:	Bukovice
Druh události:	Letecká nehoda
Hmotnostní kategorie MTOM:	<2250 kg
Druh provozu:	Ostatní
Druh letadla / SLZ:	Kluzák
Typ letadla / SLZ:	NIMBUS 2
Zdravotní následky události:	Se zraněním
PDF dokument:	

Popis:

Dne 2. 11. 2014 ÚZPLN obdržel oznámení letecké nehody kluzáku Nimbus 2 v prostoru obce Bukovice. Pilot prováděl let do vlnového proudění v rámci mezinárodní plachtařské akce „Vlnový kemp 2014“. V prostoru vypnutí z aerovleku ani v blízkém okolí nenavázal na vlnové proudění. Protože neměl dostatečnou výšku k doletu na LKMI, pokusil se neúspěšně vyhledat stoupavý proud nad svahy v blízkosti nouzové plochy Bukovice. V malé výšce pak zahájil přiblížení na přistání, ale v průběhu přistávacího manévr zachytil o vodič nadzemního elektrického vedení 22 kV. Kluzák narazil v malé vzdálenosti za vedením do země a do betonových sloupů a pletiva plotu. Pilot utrpěl lehké zranění. Kluzák byl nárazem významně poškozen.

Přičinou byl pozdě zahájený manévr na přistání, jehož důsledkem byla nedostatečná výška a náraz do vodiče elektrického vedení. Spoluúčastník faktorem pravděpodobně bylo, že vodiče nebyly zřetelně vidět proti terénu ve směru přistání šikmo proti slunci.

Figure 1.7. View on detail page on Air Accidents Investigation Institute.

■ 1.2.4 Use Case 4 – National Transportation Safety Board

<http://www.ntsb.gov/investigations/AccidentReports/Pages/aviation.aspx>

This use case serves mainly to demonstrate usage of the same ontology vocabulary on two different data sources. Additionally we might fill default values in place of missing ones in this table. For example the country value isn't specified for majority of the event records, but we can determine by the "State" field, that they happened in United States.

We will have to deal with JavaScript again. As we can see from the URL of the site (having the ".aspx" suffix), we are dealing with Active Server Pages, created by ASP.NET server. The whole table with all its sorting functionality and pagination is generated by the server and defined by the framework used on the server side. The pagination is of our consideration as it loads data into the table using AJAX call. This means data are loaded dynamically and we don't have easy access to the low level network communication happening in behind.

The options we have are analogical to those in second use case 1.2.2. We can either simulate the user action of "clicking on the next page button" or deeply analyze the JavaScript behind the pagination and perform the AJAX call manually.

The situation here is slightly different from the one in UC2 1.2.2 though. If we successfully emulate the user action for both use cases, in UC2 we will have to perform it for each line in the table (thus "during" creation of consistent ontological object and within iterating the table) whereas in this use case, we only perform the "click" when we need to load completely new set of data. The difference might not seem so essential at a first glance but the devil is in the detail: user action modifies or replaces current DOM object and the original information is lost. This does not apply to regular transfer to a new page using URL because we can use completely separate REST call. Technically it is identical to clicking a link versus opening it in a new tab in your browser, only in crOWLer these operations are performed internally on lower level.

- adding default value if no content is found

Report Number	NTSB Title	Accident Date	Report Date	City	State	Country	Other	Report
MAB1422	Fire on Board Towing Vessel <i>Shanon E. Settoon</i>	3/12/2013	12/10/2014	Bayou Perot	LA	USA	29°38.03 N, 90°10.63 W	PDF
RAB1414	Collision of BNSF Railway Company and Union Pacific Railroad Trains Near Keithville, Louisiana	12/30/2012	12/1/2014	Keithville	LA			PDF
AIR1401	Auxiliary Power Unit Battery Fire Japan Airlines Boeing 787-8, JA829J	1/7/2013	11/21/2014	Boston	MA			PDF
AAR1404	Crash Following In-Flight Fire Fresh Air, Inc. Convair CV-440-38, N153JR	3/15/2012	11/17/2014	San Juan	PR			PDF
RAR1402	Collision of Union Pacific Railroad Freight Train with BNSF Railway Freight Train	5/25/2013	11/17/2014	Chaffee	MO	USA		PDF
MAB1421	Marine Accident Brief Breakaway of Tanker <i>Harbour Feature</i> from its Moorings and Subsequent Allision with the Sarah Mildred Long Bridge	4/1/2013	11/12/2014	Portsmouth	NH			PDF

Figure 1.8. View on list page on National Transportation Safety Board webpage.

Allision of Bulk Carrier *Herbert C. Jackson* with the Jefferson Avenue Bridge

Executive Summary

About 0212 on May 12, 2013, the bulk carrier *Herbert C. Jackson* was cleared for passage through the Jefferson Avenue Bridge over the Rouge River about 6 miles southwest of Detroit, Michigan, when the bridge tender, who was legally intoxicated at the time, lowered the drawbridge, striking the bulk carrier's bow. Damage to the vessel was estimated at \$5,000. The bridge, a registered historic structure, was extensively damaged and expected to remain closed until 2015 for repair and restoration. No one was injured.

Accident Location: MI. Rouge River at Jefferson Avenue Bridge, city of River Rouge, near Detroit, Michigan - 42°16.8' N, 83°07.7' W
 Accident Date: 5/12/2013
 Accident ID: DCA13LM021

Date Adopted: 10/1/2014
 NTSB Number: MAB1419
 NTIS Number:

Probable Cause

The National Transportation Safety Board determines that the probable cause of the allision of the *Herbert C. Jackson* with the Jefferson Avenue Bridge was the intoxicated bridge tender's closing of the drawbridge as the vessel began its transit through the open bridge span.

Full Report

MAB1419

Related Press Releases

Related Events

Related Investigations

More NTSB Links

- Investigation Process
- Data & Stats
- Accident Reports
- Most Wanted List

Figure 1.9. View on detail page on National Transportation Safety Board webpage.

1.3 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 particular 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 original crOWLer implementation it was necessary to fulfill the first three steps with an actual programming. In order to perform this task, we needed to have a programmer with knowledge of Java programming language, and several technologies used on the web. Moreover a knowledge of the domain of data being scraped is needed 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.

1.4 Proposed Solution and Methodology

To simplify the creation of guidelines – scenarios – for crOWLer, we will propose a tool that allows user to select elements directly on the crawled web page, with all the necessary settings, pass the scenario created to the crOWLer and obtain the results in a form of an RDF graph.

1.5 Specific goals of the thesis

- define use-cases for the semantic data creation
- create syntax for scenario used by crOWLer
- implement a web browser extension for creating these scenarios
- this extension shall
 - load and visualise ontology
 - join page structure and ontology resources in a form of scenario
 - serialize scenario and necessary ontological data
- parse the scenario by crOWLer
- run crOWLer following the scenario
- store the extracted data

1.6 Work structure

Next part of this work 2 will cover tools and technologies (and the related lingo) used in this work and in the field.

Chapter 3 will describe research on existing solutions and how they influenced results of this work.

Chapter 4 is the main part and describes the proposed design.

Chapter 5 gives details about the prototype implemented according to proposed design.

Both, design and implementation, will then be confronted against the real life use cases1.2.

Chapter 2

Principles and technologies

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

2.1 Technology of Semantic Web



Figure 2.1. Logo of Semantic Web

Wikipedia defines Semantic Web as a collaborative movement led by international standards body the World Wide Web Consortium (W3C) [4]. 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 [12]. 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 (URIs). 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 is 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[13]. 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 DBpedia¹⁾, 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.

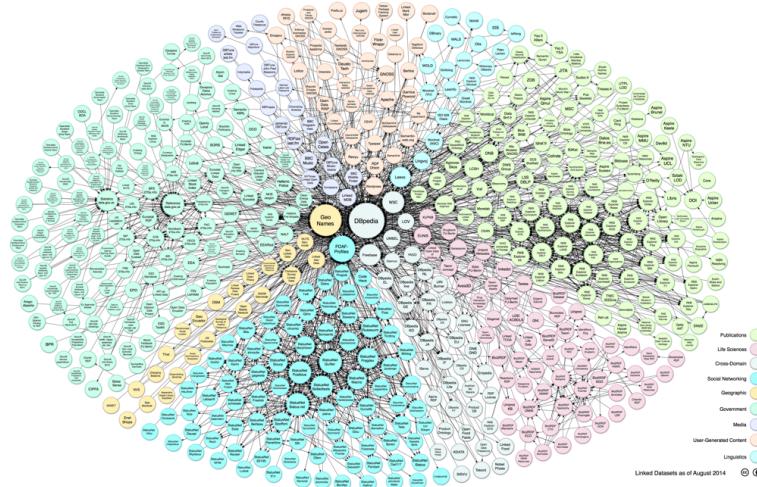


Figure 2.2. The Linking Open Data cloud diagram ²⁾

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 [14].

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.

¹⁾ <http://dbpedia.org>

■ 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 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 UUID¹⁾ 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 in appendix C 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.

■ 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 an OWL ontology, it allow us to use unions or intersections of classes or cardinality of properties. All this capabilities comes in with well defined 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
- <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 a 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 data²⁾.

¹⁾ https://en.wikipedia.org/wiki/Uniform_resource_identifier

²⁾ <http://rdfa.info/play/>

2.6 SPARQL

Is a semantic query language for data stored in RDF format [15]. Using SPARQL syntax we define a pattern of the RDF graph using triples and as a result we obtain such 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 ¹⁾²⁾.

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 forms into which we can serialize our RDF data [16]. 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. D

2.8 Turtle syntax

Turtle syntax is another popular syntax for expressing RDF. It allows an RDF graph to be completely written in a compact and natural text form, with abbreviations for common usage patterns and datatypes [17]. Its syntax suits more naturally to RDF data as it conforms the triple pattern. Follows an example about author of this work.

```
@base <http://kub1x.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<#me> a foaf:Person;
      foaf:name "Jakub Podlaha".
```

¹⁾ <http://dbpedia.org/sparql> Dbpedia SPARQL endpoint

²⁾ <http://linkedgeodata.org/sparql> LinkedGeoData SPARQL endpoint

Chapter 3

Existing solutions

In this chapter we'll describe the research made on existing solutions for given task (scraping and annotating data from a web). The performed search was focused on tools directly targeting the problem, as well as libraries and technologies that could be included in the 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. The tool 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 retrieved from the documents and annotated with resources from the ontologies.

3.1.1 Advantages and pitfalls of Semantic crawlers

To properly target the benefits the semantification of the scraped data brings to the user, let us quickly follow an evolution from the most primitive technologies for scraping data to the advanced ones. The ultimate goal is to effectively search in data and maximally utilize the knowledge it carries.

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 previous case. Such data is static, but can be queried over and over and possibly re-retrieved when becomes obsolete. Its 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 break structure of an ontology. In its state from April 2014 DBpedia states, there is 3.64 million resources, out of which 1.83 million are classified in a consistent Ontology [18]. That is only half of the data being arguably consistent with each other. That does not say the rest is bad. Only that it might cause a inconsistency and prevent us from reasoning if we include a wrong subset of the data.

Just like with "hardcoded" crawling technique, the semantic crawling is tightly bound to the structure of the crawled web. The web is being matched against some pattern described by selectors and the matching element, when found, is 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 partially or completely invalid.

Many web pages load their data dynamically using AJAX queries. Some pages simply changes it's content frequently (e.g. news pages, forums, user content pages, like video or music servers and social web applications). Crawling content on such servers would require almost constant crawling and would cause growth into massive ontology of, oftentimes, questionable quality.

The semantic crawling is an useful way to effectively obtain and query on data from the web, but it still have it's challenges to overtake.

3.2 Analysis of crOWLer

A thorough analysis of the current program shall precede creation of the final design. We will focus on architecture, dependencies and components that will have to be reimplemented.

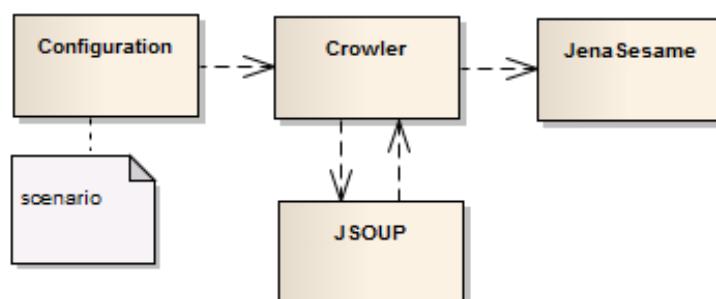


Figure 3.1. General architecture of the original crOWLer implementation

Original implementation is a prototype of console Java application. It uses Apache Jena library [19] for handling ontological data and JSOUP library [20] for accessing web-pages and addressing elements. Instead of scenario file, crOWLer accepts Java .class files containing an implementation of ConfigurationFactory class. In appendix E you can see definition of the configuration components of crOWLer. The class diagram in appendix F describes the InitialDefinition and the Selector classes along with their connection and usage of crOWLer. Configuration defined using this structure specifies all the information needed for crawling process:

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

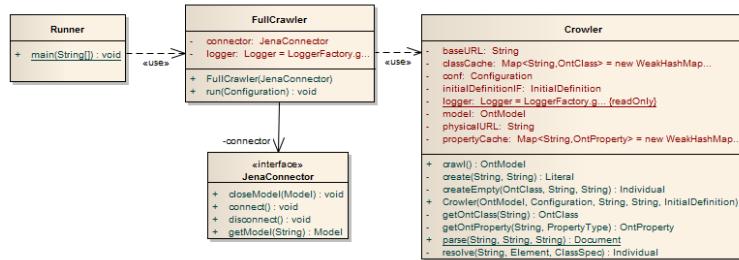


Figure 3.2. Core classes of original crOWLer implementation

The main program flow of crOWLer lays upon few core classes. The pair **FullCrawler**, **Crowler** form the crawling process loop. In this loop **FullCrawler** fetches the source web pages and passes them one by one to the **Crowler**. The **NextPageResolver**, which defines list of pages to be crawled is structure implemented within the configuration and thus is specific for given problem instance. Results are stored in the outer loop after each scraped page.

The inner loop performed by **Crowler** finds a set of HTML elements as defined by the **InitialDefinition** class, which also comes with the configuration. Each of these elements serve as a root for a tree of ontological individuals linked by their properties. The tree is in configuration defined using **ClassSpec** and **PropertySpec** classes that hold definition of type of the individual and the assigned property respectively. The spec-classes also carry information about selector, used to find the corresponding HTML element. A collection of **Selector** classes is available and can be extended. JSOUP selector handling is implemented as well as selector chaining or resolving data from a link target.

In Crowler an individual of an ontological object is created after all his defined properties values are scraped within the inner loop, as the URI of the individual can be formed using one or more of these values. This way we can refer to the same object if we create individual of the same URI on two different pages for example.

3.2.1 Issues of crOWLer configuration

From deeper analysis of the original crOWLer source we can observe, that the whole scraping process rely on the configuration defining it – a set of Java classes, implementing the predefined interfaces and using the API provided.

This reveals the issue being addressed. Writing of 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 example of actual configuration code of original crOWLer implementation. It uses NPU class as simple static storage for URI's used in our ontology. It creates a *monumnetRecord* object for each table row as defined by the "initial definition". The second part create *district* object with it's label (found in third table column denoted by the `td:eq(2)` JSOUP 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));
```

XXX Direct GET (POST) calls no javascript -> WebDriver

XXX Reference to uc4 – javascript pagination (ajax) wouldn't work at all in original
XXX fix following paragraph after previous is written

Previous example more or less defines requirements on scenario for semantic crawler. To fully satisfy the current implementation of crOWLer, 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.

3.3 Strigil

Strigil is an ontological scraping system developed at Faculty of Mathematics and Physics of the Charles University in Prague ¹⁾. It represents an easily configurable tool that enables one to retrieve data from textual or weak structured documents. [21]

It consists of webserver and backend service. The webserver offers frontend for configuring the crawling process. The backend then follows the configuration and handles downloading, scraping the data and storing results. Strigil strongly focusses on the download process. Components of the backend conform in a structure of Download-Manager, Downloaders and Proxy servers that help to distribute the load of data being transferred.

The frontend part serves user interface for handling ontological data on top of a web being scraped. It internally creates its scraping script (will be referred to as

¹⁾ <http://xrg.ksi.ms.mff.cuni.cz/software/ld/ldi.html#strigil>

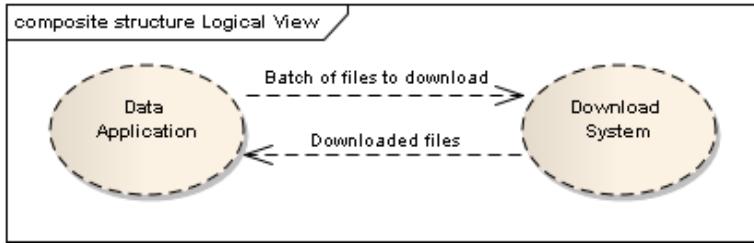


Figure 3.3. Overall Architecture of Strigil

Strigil/XML) which strongly inspired format for scenario used in the actual implementation later in this work and will be closely analyzed in next chapter 4.

■ 3.3.1 What problem does it solve?

The architecture of Strigil is tailor made for parallel processing of documents. The installation of Strigil requires working Apache2 web server with PHP5, Tomcat, PostgreSQL database, OpenMQ service and several other components before the actual deployment of Strigil into the environment. The system is designed for processing many requests on targeted server, heavy loads of data and long running tasks. It's complicated architecture and installation process prevents it from being effectively used in occasional simple, yet non trivial, scraping tasks.

Moreover it's download system fetches only the main HTML data (XXX like the original crawler implementation) and treats it as static document. This way it can't properly handle dynamic content and temporal changes in documents performed by JavaScript.

■ 3.3.2 Detailed architecture of Strigil platform

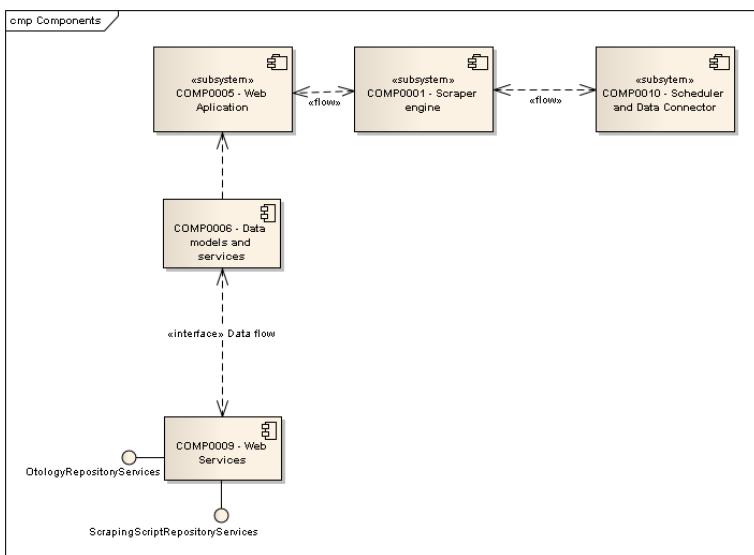
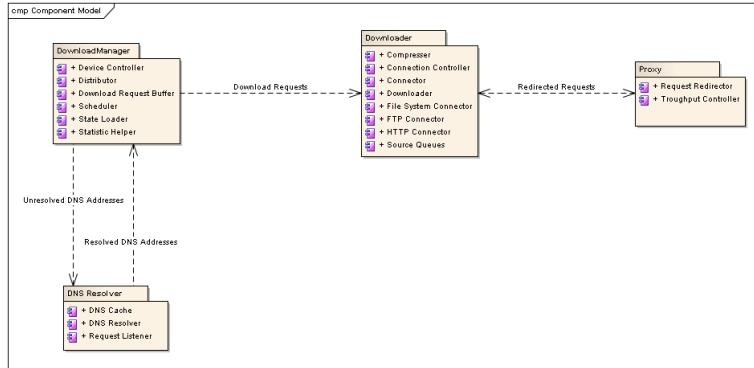
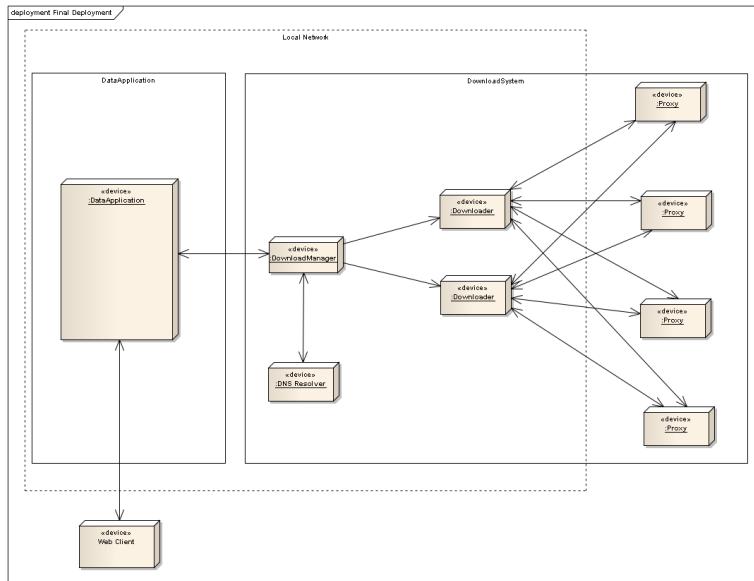


Figure 3.4. Components of Data Application part of Strigil

■ 3.3.3 What inspiration it brings for crOWLer

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

**Figure 3.5.** Components of Download System part of Strigil**Figure 3.6.** Example deployment structure of Strigil

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 does not reach the richness of Strigil/XML format.

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 later chapters.

3.4.1 InfoCram 6000 – ExtBrain

InfoCram 6000 is part of project ExtBrain¹⁾ that is developed here on Department of Computer Science. This specific part was implemented by Jiří Mašek and is described as “prototype of user interface for visual definition of extraction rules for ExtBrain Extractor”. Its intended usage is very close to the usage of SOWL. It is a 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.

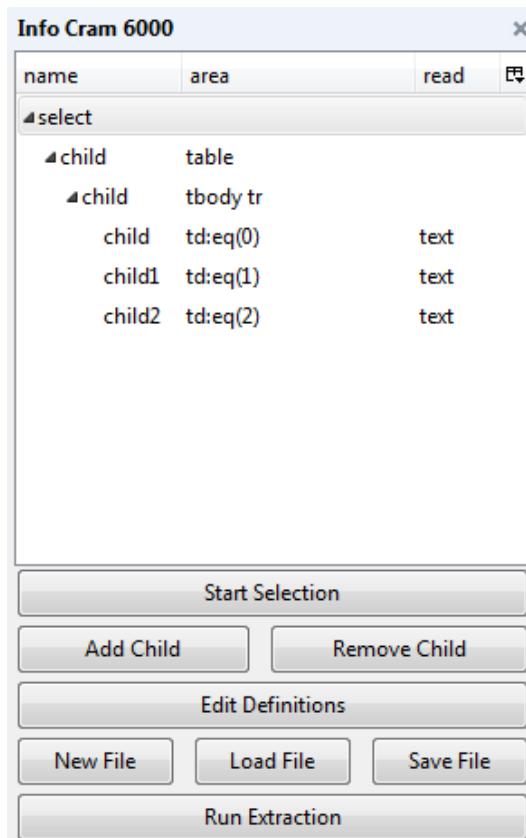


Figure 3.7. Main window of InfoCram 6000

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. Its implementation is based on Aardvark¹⁾, a Firefox extension that addresses this issue using mouse selection and several keyboard commands.

InfoCram does not use simple CSS or XPath selectors, but includes Sizzle library to handle selectors for it. Sizzle is a very popular library for handling selectors, which also defines its own selectors like :eq(), or :first. It's simpler and more expressive than CSS. Its popularity is mainly based on its 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

¹⁾ <http://www.extbrain.net>

¹⁾ <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. These 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 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 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 allows 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 development of this plugin. Selenium IDE, as being plugin itself, implements its own plugin system, through which it allows other developers to extend its 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 system using which a new layer is defined and layered over existing part of application layout while extending or modifying it. The overlay system itself comes with Mozilla stack and can be used on IDE by default.

The functionality of IDE is, however, linked with its layout 3.8 and has to be taken into 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.

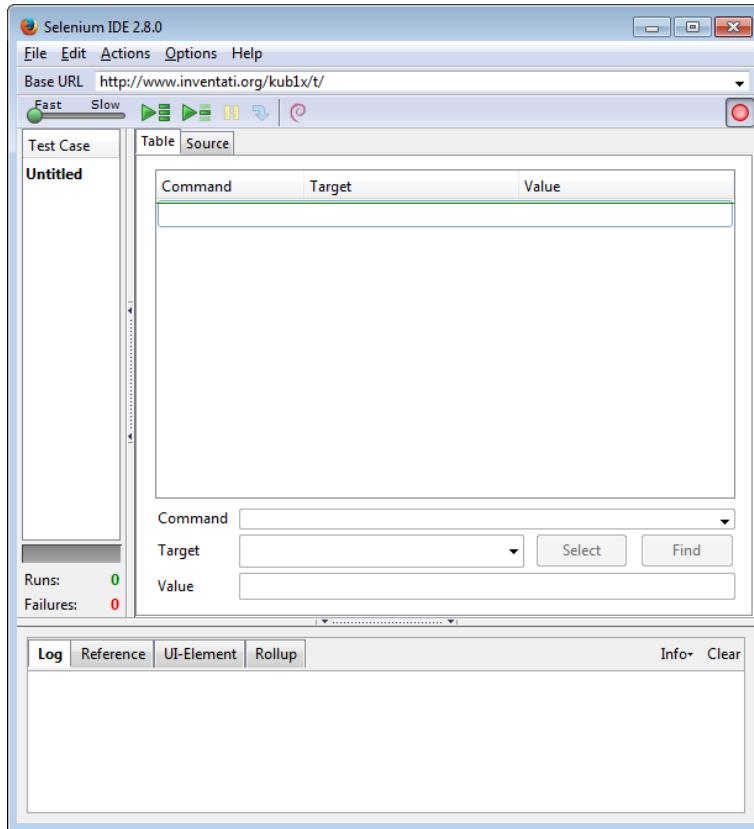


Figure 3.8. GUI of Selenium IDE showing the Command, Target and Value fields.

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 assertion 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 its 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 Selenium 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)`¹⁾ 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 Libraries for SOWL

3.5.1 jQuery

jQuery[22] is widely used JavaScript library that simplifies general tasks like DOM manipulation or event handling. A simplified selectors can be used to target DOM elements as jQuery internally uses Sizzle[23] library for selector handling. Compared to Vanilla JavaScript[24], jQuery produces more compact and coherent code.

Developers can extend the jQuery library with their own plugins. This is the case for two most promising JavaScript libraries handling RDF and OWL data, and so jQuery will be necessary if we decide to use either jOWL3.6.2 or rdfQuery3.6.3.

3.5.2 jOWL

The jOWL library is a jQuery plugin for navigating and visualising OWL-RDFS documents[25]. It can parse and handle RDF files, store them in its internal storage and query on them using subset of QUERY-DL language[26]. The library was last updated in 2008¹⁾.

3.5.3 rdfQuery



rdfQuery[27] is a JavaScript library for RDF-related processing. It supports parsing RDFa, RDF, OWL formats for loading data. It can dynamically embed HTML webpage with RDFa data. rdfQuery is written as a jQuery plugin. The intended use of the rdfQuery library is to write queries over data stored in rdfQuery internal datastore in similar way as DOM objects are queried using jQuery. Moreover the whole concept is based on SPARQL and design in a manner that make the resulting JavaScript code look familiar when compared to native SPARQL query.

TODO example

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

¹⁾ <https://code.google.com/p/jowl-plugin/>

3.5.4 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. It has been released in two forms: as a bookmarklet and a Firefox extension. The later was used in a modified form in the InfoCram 60003.4.1 and later in one of SOWL (SelectOWL) prototypes3.7.

3.6 Early implementation

TODO

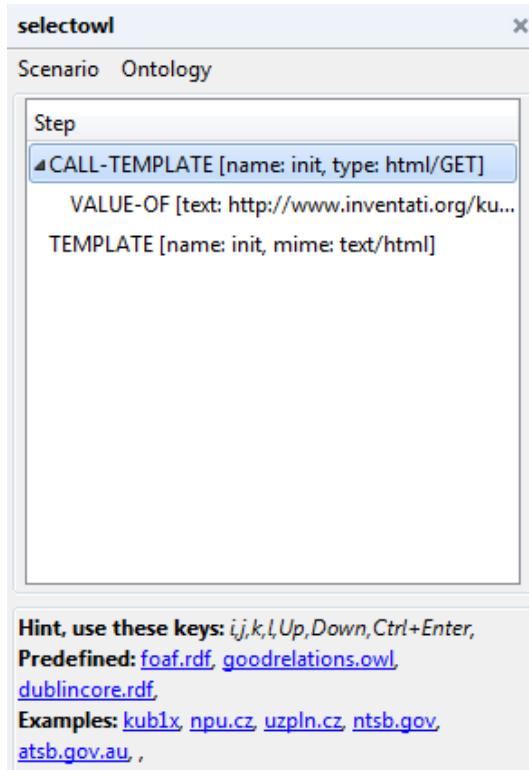


Figure 3.9. View at original SelectOWL sidebar implementation.

TODO Aardvark and Infocram

The InfoCram specific functionality was carefully removed and replaced with procedures following the SOWL's workflow.

The added/modified features are namely: the selector creation feature, the drag/drop event handling, the messaging, and modification of user commands.

XXX describe each feature here or later in this chapter...

Chapter 4

Program design

This chapter defines the overall behavior of the program derived from requirements as well as the proposed design.

4.1 Workflow

Derived from previous use case definitions we can derive the general workflow for both SOWL and crOWLer part of the implementation.

4.1.1 Main line

- user loads/creates ontology using SOWL
- user opens webpage with data
- user creates scenario using SOWL
 - user adds selectors to scenario steps
 - user adds resources to scenario steps
- SOWL sends scenario to crOWLer
- crOWLer crawls the web according to scenario and stores results in a file or repository

4.1.2 Scenario creation

- user starts scenario creation in SOWL
- loop until finished:
 - user creates a step in scenario
 - user selects an element on page, a selector is generated if applicable, on the step
 - user selects a resource, resource is updated on the appropriate field of the step, if applicable

4.1.3 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)

■ 4.1.4 crOWLer scraping

- user runs crOWLer passing it the created scenario
- crOWLer parses the scenario
- crOWLer scrapes data from the webpage following the scenario
- crOWLer stores the results in file or repository

■ 4.2 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.

■ 4.2.1 Strigil/XML

Strigil, 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 its 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

¹⁾ <https://drive.google.com/file/d/0B4On-1Gb38CgWlAyZDhGbDV2TFk/edit> Scraping script documentation

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 does not 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.

■ 4.2.2 Adaptation of Strigil/XML format

Strigil creates its scraping script internally hidden under GUI and leaves user unaware of its 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 data types 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 does not 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 allow us to handle webpages the same way as user sees them.

Usage of actual full-stack web browser with JavaScript engine long 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 parameters and contain it's code. To execute this function we would call `function-call` and identify it by it's 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 it's intended use for distributed downloader and it lacks some functionality. In SOWL we would almost necessarily modify it's formal definition and thus it is of consideration if we can't make use of more appropriate format.

■ 4.2.3 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 Strigi/XML format using hardcoded XML snippets and string formating – 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 is not hard to implement, but brings in poor maintainability and with additional complexity it looses 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. XXX not true anymore ;)

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

H

■ 4.2.4 Consequences of conversion to JSON format

XXX We don't have `text content` like XML elements can, but Strigil does not really use that

XXX We don't have child nodes.. so we have to keep them in an array, like in `steps`

XXX But hey, we can use any key to store a substep, not only the array, we practically use different approach here.. instead of tree structure we have tree of hash maps.

The `onto-elem` command benefits exactly from this difference between XML and JSON. In original Strigil/XML the `onto-elem` tag allow us to specify URI of the resulting individual (as commonly denoted by the `about` property), by taking it from it's "first child" which is expected to be `value-of` tag. Needles to say, this specification lowers robustness as the position in the XML file is not enforced by the syntax and can be easily unintentionally broken by accidental swap of two elements, although it

wouldn't invalidate the files syntax and thus won't be captured by the script parser. In the JSON format we lack the notion of child elements. Even when we simulate it as mentioned before, we'd only cause the same indetermination. So instead, we simply reserve a property named `about` exactly for the described use. The same technique is used to specify URL for a `call-template` on it's `valueof` property.

XXX TODO might rename these

XXX call-template might need simple steps array property, as it might follow a list of URLs, not only one

4.3 JavaScript and events support

Special attention have to be payed when dealing with direct interaction with DOM elements and script execution. WebDriver supports injection and execution of JavaScript as well as simulation of user interactions like "click" on element or "back" and "forward" navigation. Even though it brings great power there are considerations and great limitations to be taken in account.

WebDriver supports execution of JavaScript directly on webpage loaded in the driver. This is done by calling `executeScript` or `executeAsyncScript` function on the `driver` object. First argument of these functions is string defining content of JavaScript function we want to execute. Header and actual call of this function will be added for us before it gets attached to the webpage. We can pass any number of accepted arguments to these functions and they will be accessible through standard `arguments` object in on the JavaScript side. Types, corresponding to standard JavaScript types are supported as arguments: number, boolean, String, WebElement or List of any combination of the previous¹⁾. The second – asynchronous version returns immediately with a `response` object. It provides callback as additional argument to the JavaScript call. This callback is used for synchronization when accessing the result on the `response` object from Java.

XXX TODO better example this one is from here²⁾

```
List<WebElement> labels = driver.findElements(By.tagName("label"));
List<WebElement> inputs = ((List<WebElement>) ((JavaScriptExecutor)driver).executeScript(
    "var labels = arguments[0], inputs = []; for (var i=0; i < labels.length; i++){" +
    "inputs.push(document.getElementById(labels[i].getAttribute('for')));"
} return inputs;", labels);
```

In simple case we can use JavaScript to extend functionality of crOWLER. It might be used as a complex string formatter, parser for nontrivial values etc. In following example it is used to condition on attribute value of an anchor tag.

```
WebElement el = driver.findElement(By.cssSelector('a.detail'));
String result = (String) ((JavaScriptExecutor)driver).executeScript(
    "var elem = arguments[0];"+
    "var href = elem.getAttribute('href');"+
    "return (href ==='#' ? window.location.href : href);", el);
```

Previous example is simple, yet if we wanted to cover this use case with our scenario implementation we would bring a lot of problem-specific XXX balast XXX into the scenario syntax. We would have to use special syntax for obtaining current URL and for conditioning on values. Following code demonstrates how this functionality might look if

¹⁾ <http://goo.gl/Hhwq3l> Selenium JavaScriptExecutor documentation

²⁾ http://docs.seleniumhq.org/docs/03_webdriver.jsp#using-JavaScript

it was covered only by scenario syntax without usage of JavaScript. The `getCurrentUrl` function is inspired by Strigil.

```
{
  command: "condition",
  condition: "eq",
  param: "#",
  value: {
    command: "value-of",
    selector: "a.detail",
  }
  onfalse: {
    command: "function",
    value: "getCurrentUrl",
  }
}
```

We've declared the `condition` command with implementation of `eq` operator (and probably several other un/equality operators) and the `function` command with implementation of `getCurrentUrl` which, again, probably is not the last function to be implemented. All this would require update of the scenario parser, the implementation for commands and all their attributes and thus update of the whole backend every time, new functionality is needed. The advantage is, that user does not have to know JavaScript and understand how it is called in WebDriver. It is discutable if XXX

With use of JavaScript it might look as follows:

```
{
  command: "value-of",
  selector: "a.detail",
  exec: "var href = elem.getAttribute('href');" +
        "return (href ==='#' ? window.location.href : href);"
}
```

In this case we embedded only the `value-of` with a single attribute that takes JavaScript. From there we have technically unlimited power for extending the functionality of the crOWLer without need of changing the Java implementation.

Note that the first line of the original JavaScript was omitted:

```
var elem = arguments[0];
```

It can be automatically prepended every time, we exec JavaScript on a single DOM element. It is a simple helper and does not interfere with anything XXX (as we can redefine variable as many times as we want). Similarly we could predefined variable `value` when passing a string or number to a JavaScript function.

But with great power comes a great current squared times resistance ¹⁾ XXX. With usage of JavaScript as suggested in previous paragraphs we have to take in account two major considerations.

Firstly, JavaScript function can accept any number of parameters and return an arbitrary value. In both cases, the parameters and the return value, we can be of any of the allowed types (as JavaScript is not strongly typed language). We thus have to specify what exact parameters are being passed to a function and what result is expected. We also have to implement a robust way of controlling and properly define a fallback-on-error behavior. This is especially important as we might want to use

¹⁾ <http://www.xkcd.com/643/>

JavaScript function not only as a string filter, but also for example as a universal selector where we struggle with classical selectors. Any additional use have to be described separately before it can be universally used.

More importantly, there is the second consideration. Any DOM element is accessible from the JavaScript function. When this element is modified or even removed, it becomes invalid from the Java context. The same applies for operations on the whole page. When a link is followed, the original DOM tree is dropped and all references are lost.

To specify the behavior during this issue, below you can see a simple test. When link is clicked, the webdriver follows the link in current window and the reference to the original DOM is lost.

```
WebDriver wd = new FirefoxDriver();
wd.navigate().to("http://www.inventati.org/kub1x/t/");
WebElement a = wd.findElement(By.cssSelector("a"));
System.out.println(a.getText()); // Prints "detail"
a.click();
System.out.println(a.getText());
// throws org.openqa.selenium.StaleElementReferenceException:
// { "errorMessage": "Element does not exist in cache", ... }
```

In crOWLer, we can now distinguish between two ways of ascending to another HTML page:

1. using `call-template` command
2. using JavaScript or user event such as “click” or “back”

The `call-template` is always called on an URL and always creates new web context, keeping the original one untouched. It actually behaves like call stack, so when we return from the template call, we can follow on the original DOM tree. Just to note: compared to corresponding Strigil command, crOWLer persists the ontological context throughout this call, and so we can relate to it when assigning properties.

Direct interaction with current window in any way that changes page location will, however, irreversibly invalidate all the elements of current DOM. This does not have to mean we can't use this functionality all together. Probably the best solution would be to only allow DOM modifying operations on the bottom level of templates (i.e. within the `steps` property of the `template` command in scenario). At this place we only hold the `body` of current document and as such we can simply replace it with the newly loaded content. In the original crOWLer implementation, this would be the spot between two “Initial Definitions”.

Even though the JavaScript is sandboxed in WebDriver, it is still running in a browser in your computer and could technically submit some data on a web. Security issues haven't been considered so far, but might become a point of interest when we take in account an option of obtaining and executing scenarios from unknown sources.

4.4 User Interface

Here the required structure of user interface is described.

4.4.1 SOWL user interface

The user interface of SOWL shall be presented in a form of sidebar. The sidebar shall have two parts: a scenario editor and resources list. Scenario editor shall contain a tree

shaped structure of steps of the scenario being created along with panel for editing the general settings of the scenario. The resources list shall accept dropping of ontology files which would load it into current dataset. Addition of resources manually shall be possible using a button. The list shall show all currently loaded resources and allow textual filtering.

SOWL shall enable tag selection on the webpage being processed by clicking or other user action.

■ 4.4.2 crOWLer user interface

CrOWLer is a console application. It shall accept scenario as one of it's parameters. Following settings shall be enabled using parameters as well:

- setting of target directory for RDF files
- setting of sesame repository for the result storage

■ 4.5 Model

Presenting proposed design of the two programs the SOWL Firefox addon and the crOWLer Java application.

■ 4.5.1 SOWL model

Current recommendation of Mozilla Developer Network suggests developing new addons using their native SDK. It allows creation of restartless addons, uses new API and limits usage of older libraries or low level calls by wrapping it in consistent API.

The SDK based addons have partially predefined structure. The *background script* runs in it's own scope and uses the SDK API to control the addon's behavior. The *content script* is a JavaScript code that is injected into a webpage but runs in it's own sandboxed overlay, while having access to pages DOM and JavaScript content. In SOWL, the scenario editor will be placed into a sidebar. Sidebar holds standard HTML window object in which the JavaScript code is running. All three components communicate via textual messages using port object offered internally by Firefox.

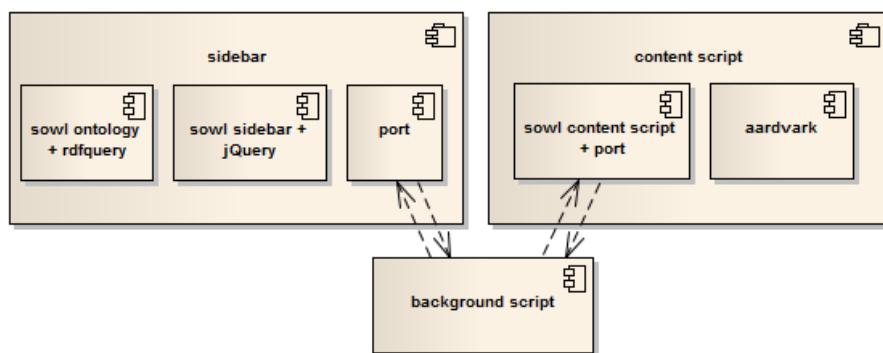


Figure 4.1. A component structure of the SOWL Firefox addon.

■ 4.5.2 crOWLer model

In the new implementation of the scraping backend the original JSOUP component will be replaced by WebDriver. WebDriver, with it's support for JavaScript, will help to handle dynamic content and brings in new possibilities for the crOWLer itself. The

original configuration component is replaced by parser for the SOWL/JSON scenario format (XXX ref). The core crOWLer is also reimplemented according to new set of instructions (i.e. commands in the scenario) and the new web interface (i.e. the WebDriver instead of the native Java Jsoup library).

The overall architecture then looks as follows:

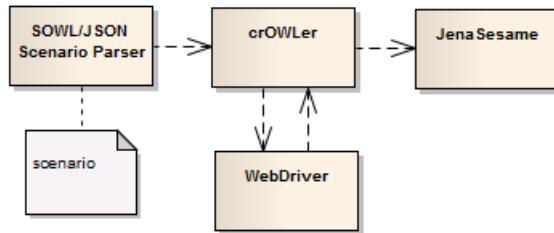


Figure 4.2. A new overall architecture of the crOWLer implementation.

Chapter 5

Program Implementation

This chapter describes the implemented prototype.

TODO

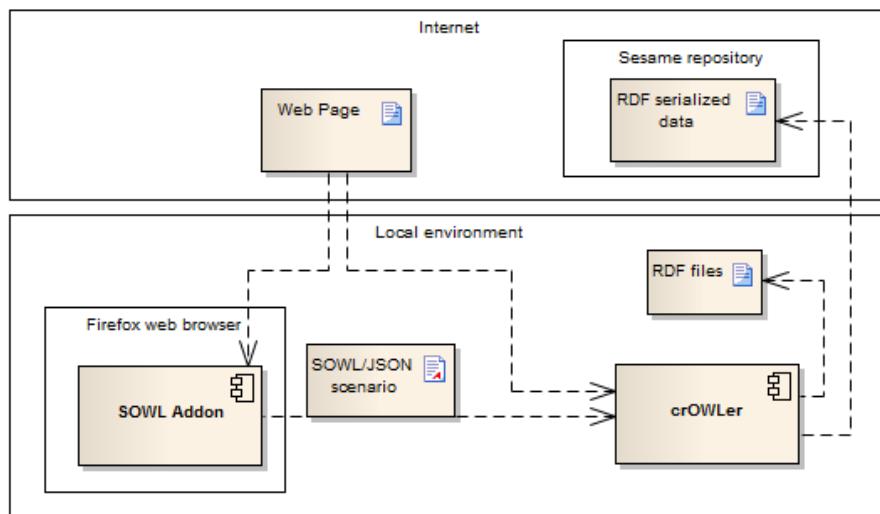


Figure 5.1. Overview of the whole stack and files exchanged

5.1 Parsing Ontologies in JavaScript

TODO jOWL vs rdfQuery

TODO aardvark

TODO TODO TODO

In current Implementation of SOWL we create different type of Firofox extension using new SDK. Moreover the aardvark code is injected directly into the webpage using the Content Script feature of Firefox SDK. According to these differences the bookmarklet version better fits the needs and is used. The code was included in the addon and extended with features necessary for SOWL.

TODO TODO TODO

5.2 SOWL implementation

XXX

5.3 crOWLer implementation

5.3.1 architecture

XXX include class diagram..?

■ 5.3.2 Implemented and unimplemented capabilities

XXX

Chapter 6

Results and Tests

XXX

- <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&PrirUbyt0d=3.5.1958&PrirUbytDo=10.12.2013>
- <http://dominanty.cz/pamatky-cihana.php>

Chapter 7

Conclusion

TBD

It is suitable to notice, that in many cases the intentions and activities of semantic web community focus on government data [28]. The common goal leads us to turn the web into an open, accessible source of knowledge and data of all kinds, linking the data together where possible. Naturally the governmental data and statistics get the most attention. Government handles, collects and is often obliged to publish in some form a lot of data and statistics. Not always this form complies with standards of semantic web. Sometimes it might even be the case of intentional presenting of malformed data or obfuscation. In the big picture, misinformation of people seems to be the major threat to democracy as we usually envision it. By supporting the creation of semantic data we are naturally taking part in this movement. The hope is to bring government data closer to people to help overcome the information gap that prevents each one of us from being adequately informed about how our resources are being spent and how our countries are truly led and offices driven. I hope this and any follow-up work will serve to support this common vision.

References

- [1] *Web Ontology Language – Wikipedia.*
https://en.wikipedia.org/wiki/Web_Ontology_Language.
- [2] *Google Knowledge Graph – Wikipedia.*
https://en.wikipedia.org/wiki/Google_Knowledge_Graph.
- [3] *Search Engine Optimization – Wikipedia.*
https://en.wikipedia.org/wiki/Search_engine_optimization.
- [4] *Semantic Web – Wikipedia.*
https://en.wikipedia.org/wiki/Semantic_Web.
- [5] *Linked Data – Connect Distributed Data Across Web.*
<http://linkeddata.org/>.
- [6] *HTML5 – Wikipedia.*
<https://en.wikipedia.org/wiki/HTML5>.
- [7] *Microformats.*
<http://microformats.org/>.
- [8] *HTML + RDFa 1.1 – Support for RDFa in HTML4 and HTML5.*
<http://dev.w3.org/html5/rdfa/>.
- [9] *Google Structured Data Testing Tool.*
<http://www.google.com/webmasters/tools/richsnippets>.
- [10] *RDFa Play – the RDFa data visualisation tool.*
<http://rdfa.info/play/>.
- [11] Robert Isele, Jürgen Umbrich, Chris Bizer, and Andreas Harth. *LDSpider: An open-source crawling framework for the Web of Linked Data*. In: *Proceedings of 9th International Semantic Web Conference (ISWC 2010) Posters and Demos*. 2010 .
<http://iswc2010.semanticweb.org/pdf/495.pdf> .
- [12] *Semantic Web – W3C.*
<http://www.w3.org/standards/semanticweb/>.
- [13] *Linking Open Data diagram.*
<http://lod-cloud.net>.
- [14] *Resource Description Framework – Wikipedia.*
https://en.wikipedia.org/wiki/Resource_Description_Framework.
- [15] *SPARQL Protocol and RDF Query Language – Wikipedia.*
<https://en.wikipedia.org/wiki/SPARQL>.
- [16] *RDF/XML – Wikipedia.*
<https://en.wikipedia.org/wiki/RDF/XML>.
- [17] *Turtle – Terse RDF Triple Language – W3C.*
- [18] *DBpedia – the Datahub.*
<http://datahub.io/dataset/dbpedia>.

-
- [19] *Apache Jena*.
<http://jena.apache.org/>.
 - [20] *JSOUP – Java HTML parser*.
<http://jsoup.org>.
 - [21] Nečaský M. Stárka J., Holubová I.. Strigil: A Framework for Data Extraction in Semi-Structured Web Documents. 2013, paper submitted to 15th International Conference on Information Integration and Web-based Applications & Services, Vienna, Austria, 2013..
 - [22] *jQuery*.
<http://jquery.com>.
 - [23] *Sizzle JavaScript selector library*.
<http://sizzlejs.com>.
 - [24] *Vanilla JS*.
<http://vanilla-js.com>.
 - [25] *jOWL – Ontology Online*.
<http://jowl.ontologyonline.org>.
 - [26] Petr Kremen. *Towards SPARQL-DL Evaluation in Pellet*. 2007.
<http://weblog.clarkparsia.com/2007/10/26/towards-sparql-dl-evaluation-in-pellet>.
 - [27] *rdfQuery – RDF processing in your browser*.
<https://code.google.com/p/rdfquery>.
 - [28] *Open Government Data*.
<http://opengovernmentdata.org/>.

Appendix A

Assignment

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science and Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Bc. Jakub Podlaha**

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: **Platform for Semantic Crawling of the Web**

Guidelines:

1. Become familiar with semantic web technologies, namely with Linked Data concept, and RDF(S), OWL 2, SPARQL languages.
2. Design a platform for creating semantic extraction scenarios of the web. The platform will provide a UI for searching a suitable ontological schema, construction of the extraction scenarios, their execution using existing tools and visualization.
3. Test the platform on the selected public data sets and evaluate its potential for semantic web authoring.

Bibliography/Sources:

- [1] Tom Heath and Christian Bizer (2011) Linked Data: Evolving the Web into a Global Data Space (1st edition). Synthesis Lectures on the Semantic Web: Theory and Technology, 1:1, 1-136. Morgan & Claypool.
[2] OWL Primer. <http://www.w3.org/TR/owl2-primer/>, cit. 19.12.2013

Diploma Thesis Supervisor: Ing. Petr Křemen, Ph.D.

Valid until the end of the summer semester of academic year 2014/2015



doc. Ing. Filip Železný, Ph.D.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, March 3, 2014

Appendix B

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

Appendix C

RDF and RDFS vocabulary

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 <code>a</code>
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
rdfs:Datatype	property values such as textual strings are examples of RDF literals literals may be plain or typed
rdf:XMLLiteral	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: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 C.1. RDF and RDFS vocabulary

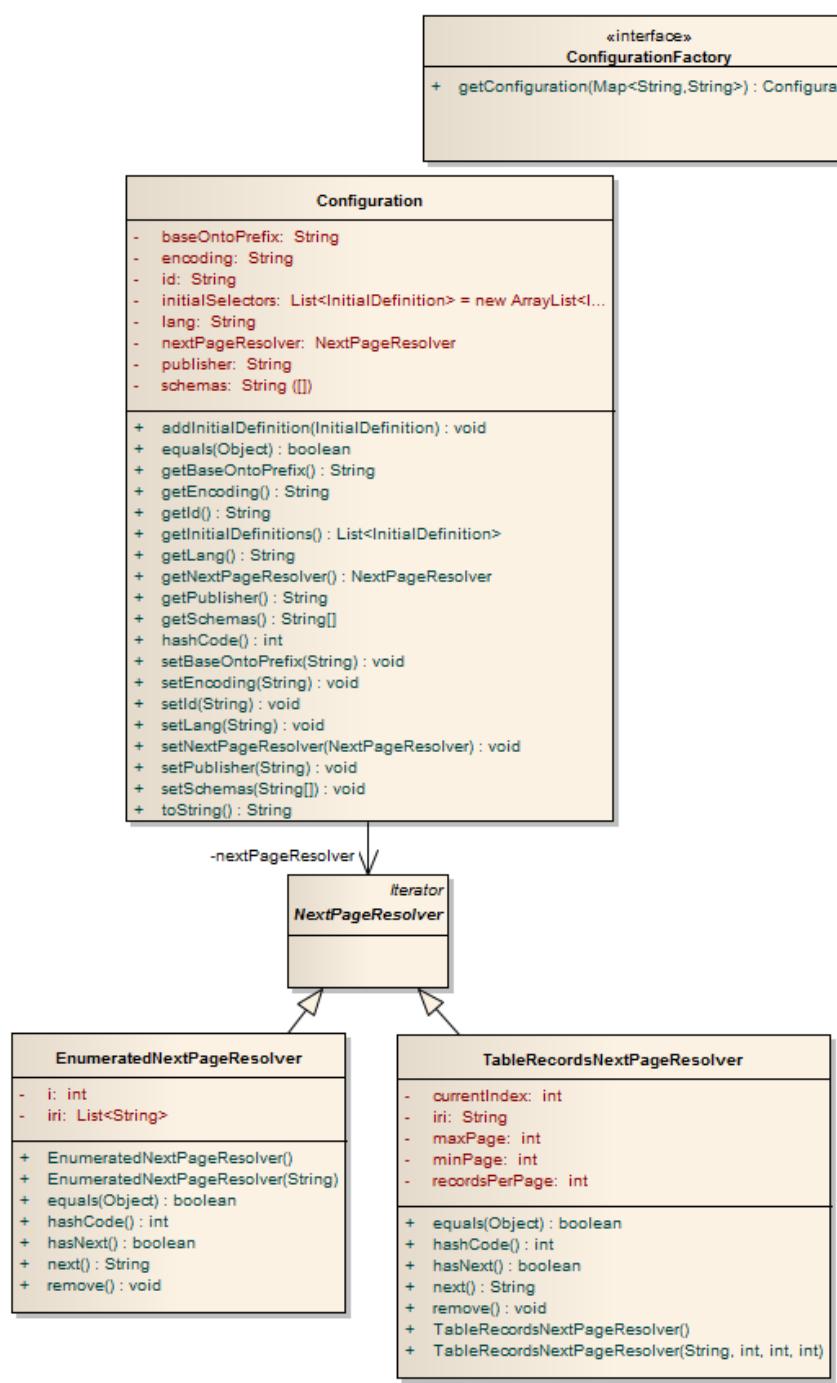
Appendix D

Example of RDF/XML syntax

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:vs="http://www.w3.org/2003/06/sw-vocab-status/ns#" xmlns:foaf="http://xmlns.com/foaf/0.1/" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <!-- Here we describe general characteristics
       of the FOAF vocabulary ('ontology'). -->
  <owl:Ontology rdf:about="http://xmlns.com/foaf/0.1/">
    dc:title="Friend of a Friend (FOAF) vocabulary"
    dc:description="The Friend of a Friend (FOAF) RDF
                    vocabulary, described using
                    W3C RDF Schema and OWL the Web
                    Ontology Language." >
  </owl:Ontology>
  <rdfs:Class rdf:about="http://xmlns.com/foaf/0.1/Person">
    rdfs:label="Person"
    rdfs:comment="A person."
    vs:term_status="stable">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
    <owl:equivalentClass
      rdf:resource="http://schema.org/Person" />
    <owl:equivalentClass
      rdf:resource="http://www.w3.org/2000/10/swap/pim/contact#Person"/>
    <rdfs:subClassOf>
      <owl:Class rdf:about="http://xmlns.com/foaf/0.1/Agent"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class
        rdf:about="http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing"
        rdfs:label="Spatial Thing"/>
    </rdfs:subClassOf>
    <rdfs:isDefinedBy
      rdf:resource="http://xmlns.com/foaf/0.1/">
    <owl:disjointWith
      rdf:resource="http://xmlns.com/foaf/0.1/Organization"/>
    <owl:disjointWith
      rdf:resource="http://xmlns.com/foaf/0.1/Project"/>
  </rdfs:Class>
  <!-- (...) -->
</rdf:RDF>
```

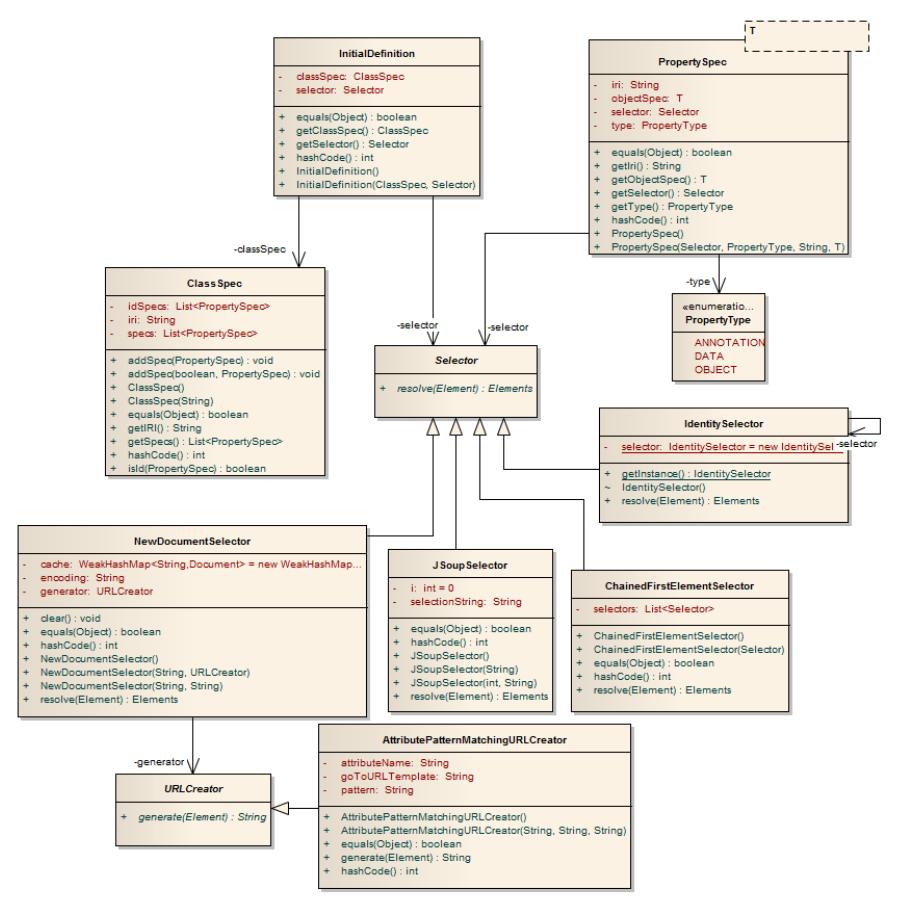
Appendix E

Configuration component of original crOWLer

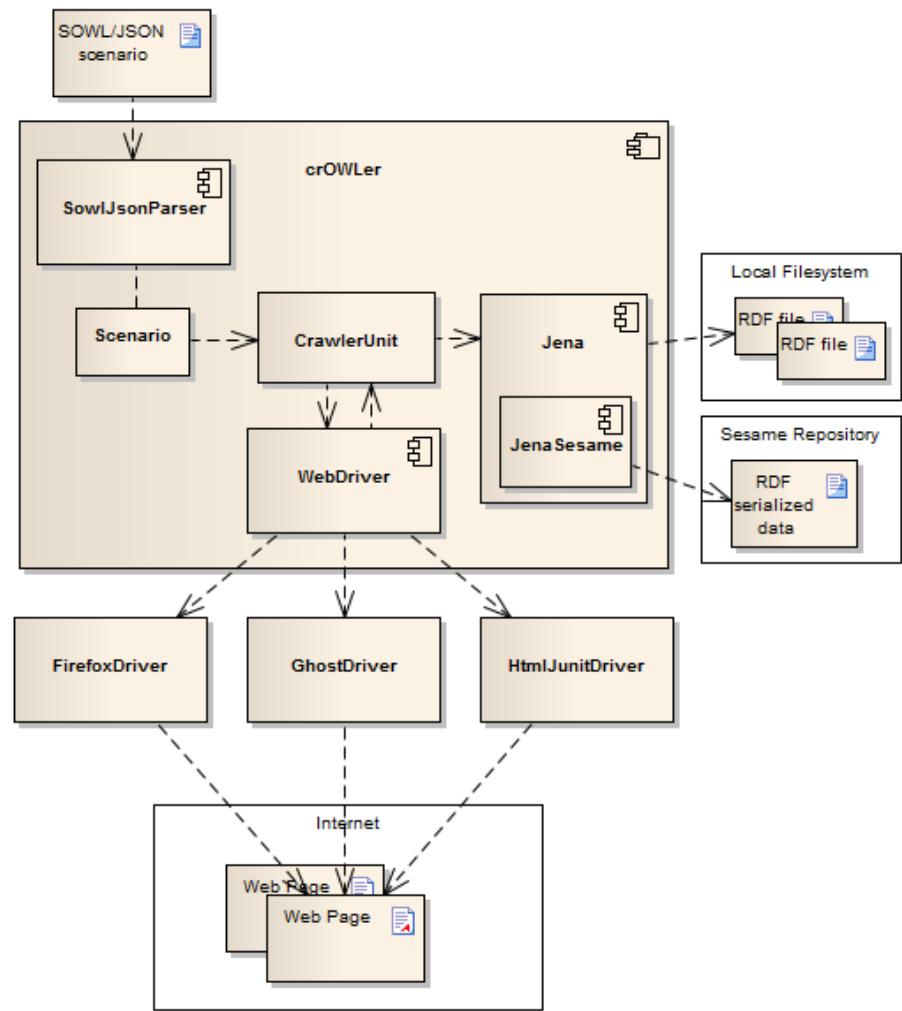


Appendix F

Selector component of original crOWLer



Appendix G crOWLer architecture



Appendix H SOWL/JSON scenario solving Use Case 1

```
{  
    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",
            },
        },
    ],
},
],
}

```