UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Andres Viikmaa

Web Data Extraction For Content Aggregation From E-Commerce Websites

Master's Thesis (30 ECTS)

Supervisor: Timo Petmanson, MSc

Tartu 2016

Veebiandmete eraldamine tooteinfo agregeerimiseks e-poodidest

Lühikokkuvõte: Internetist on saanud piiramatu andmeallikas. Läbi otsingumootorite on see andmehulk tehtud kättesaadavaks igapäevasele interneti kasutajale. Sellele vaatamata on seal ikka informatsiooni, mis pole lihtsasti kättesaadav olemasolevateotsingumootoritega. See tekitab jätkuvalt vajadust ehitada aina uusi otsingumootoreid, mis esitavad informatsiooni uuel kujul, paremini kui seda on varem tehtud. Selleks, et esitada andmeid sellisel kujul, et neist tekiks lisaväärtus tuleb nad kõigepealt kokku koguda ning seejärel töödelda ja analüüsida. Antud magistritöö uurib andmete kogumise faasi selles protsessis. Esitletakse modernset andmete eraldamise süsteemi ZedBot, mis võimaldab veebilehtedel esinevad pooleldi struktureeritud andmed teisendada kõrge täpsusega struktureeritud kujule. Loodud süsteem täidab enamikku nõudeid, mida peab tänapäevane andmeeraldus süsteem täitma, milleks on: platvormist sõltumatus, võimas reeglite kirjelduse süsteem, automaatne reeglite genereerimise süsteem ja lihtsasti kasutatav kasutajaliides andmete annoteerimiseks. Eriliselt disainitud otsi-robot võimaldab andmete eraldamist kogu veebilehelt ilma inimese sekkumiseta. Töös näidatakse, et esitletud programm on sobilik andmete eraldamiseks väga suure täpsusega suurelt hulgalt veebilehtedelt ning tööriista poolt loodud andmestiku saab kasutada tooteinfo agregeerimiseks ning uue lisandväärtuse loomiseks.

Võtmesõnad: andmekaeve, veebiandmete eraldamine, veebipoed, tootekataloogid

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Web Data Extraction For Content Aggregation From E-Commerce Websites

Abstract: World Wide Web has become an unlimited source of data. Search engines have made this information available to every day Internet user. There is still information available that is not easily accessible through existing search engines, so there remains the need to create new search engines that would present information better than before. In order to present data in a way that gives extra value, it must be collected, analysed and transformed. This master thesis focuses on data collection part. Modern information extraction system ZedBot is presented, that allows extraction of highly structured data form semi structured web pages. It complies with majority of requirements set for modern data extraction system: it is platform independent, it has powerful semi automatic wrapper generation system and has easy to use user interface for annotating structured data. Specially designed web crawler allows to extraction to be performed on whole web site level without human interaction. We show that presented tool is suitable for extraction highly accurate data from large number of websites and can be used as a data source for product aggregation system to create new added value.

Keywords: Data mining, Web Scraping, Web Data Extraction, Focused Crawler, Wrapper generation, E-Commerce, Product Catalogs

CERCS: P170 Computer science, numerical analysis, systems, control

Contents

1 Introduction

In today's modern world people are used to find information on-line. Search engines have become axiomatic tools for every computer user, they are the entrance to the Internet. Most eveyday internet searches are done using search engines such as Google, Bing, Baiduu and others. These universal search engines were designed to index entire Web as textual data. Over the years universal search engines have become better on delivering accurate search results to users, but the results are not presented in easy manner and are not organized in comparable form. Therefore usually the method from old days - opening search results in new browser tab/window or bookmarking search results - is needed for further review of search results.

This has lead to creation of vertical search engines that focus only on specific topic (consumer goods, books, flight tickets, real estate ads, scientific articles, etc.). Most well know are SkyScanner[1] that allows you search plane tickets and Google Scholar[2]. But also product price aggregation sites like PriceGrabber[3] or Google Shopping[4]are getting more popular. Consumer product search engines (aggregators) are designed to present search results in e-shop like manner, providing product name with image and usually lowest price. Each search result links to detail product page with its full specification, reviews and list of point of sales (POS). POS can be either an on-line shop or physical store. For each POS the price and stock information is displayed and "Buy" button that directs user to web shop. Majority of these aggregators focus on price comparison ordering shops based on price. This allows users to find cheapest locations where to buy the item very easily.

In order to build such system, detail product information must be gathered and systematized. One way to get this information is to ask manufactures or wholesale companies. Contacting and asking each company for data is time consuming and updating these datasets is hard to maintain, if it is not done by the companies themselves. But usually no one want additional workload and expense from their side. The second option is to buy this data from commercial service provider such as IceCat[5]. The problem with this approach is a) it's expensive for startup companies, b) dataset is usually limited to specific category (ex. Consumer Electronics) c) only covers few languages. Because of these limitations the easiest way to get data for start-up company or individual is to use web scraping technologies to gather publicly aviable data from Word Wide Web. Coping data form other sites might not be seen as absolutely legal but copyright does not protect factual content and product description can be considered as factual data. In detail analysis of legal aspects can be found in [Vel13], [TVE14] and [O'R06].

This master thesis presents modern platform for medium scale structured data extraction from semi structured web pages called ZedBot. First part gives overview about existing work in the area of Web data mining and is divided into two separate topics: web data extraction and web crawling. Next chapter describes the implementation of built system. This is followed with evaluation of the system together with discussion and finally on last chapter with conclusions and future work ideas are presented.

[1]http://www.skyscanner.net
[2]http://scholar.google.com
[3]http://www.pricegrabber.com/
[4]http://www.google.com/shopping
[5]http://www.icecat.biz

2 Overview of web data extraction methods

Web search engines have been present more that two decades and have evolved from simple full text search engines into complex systems that analyse web page content together with links that point into them. But still even most modern web search engine has three main components[YG14]:

- **Web crawling and data acquisition** – Downloads pages for offline use and extracting link and text data,
- **Data storage and offline processing** – Data cleanup and aggregation and reverse search index creation,
- **Query processing** – Querying search index and ranking results based of search terms.

As mentioned in introduction, this work will focus on web content extraction, therefore only web crawling and data acquisition is covered in this section. Other parts (data aggregation, product classification task, duplicate matching) required to build fully functional product search engine are not discussed as these are implenented as separate system and not described in this thesis. In next subsections we look content extraction and web crawling in more detail.

2.1 Semantic Web

Semantic web was designed to make web conent machine readable and by that allowing information shared beyond originated website. Semantic web allows to define not only relations inside single web page but also link different websites together in a meaningful way and by that creating internet of knowlege. Embedding semantic information to web pages can be done in different ways. Most popular is microdata which is shown in Figure 1. Other ways to include semantic meaning are RDFa (Resource Description Framework in Attributes) and JSON-LD (JSON Linked Data) embeddings.

```
<p>Rose Tyler was sponsored by Sarah Jane
   Smith in the membership process.</p>
```

```
<p itemscope itemprop="Person" itemtype="http://schema.org/Person">
  <span itemprop="name">Rose Tyler</span> was sponsored by
  <span itemscope itemprop="sponsor" itemtype="http://schema.org/Person">
    <span itemprop="name">Sarah Jane Smith</span>
  </span> in the membership process.
</p>
```

Figure 1: HTML markup without microdata (top) and with microdata (bottom).

Although `Schema.org` is becoming de facto standard to define schemas for semantic web it is not necessary to use their schema. But consolidating and unifing schemas makes their usage easier and allows spreading. For commercial products `GoodRealations`[6] schema was created by Martin Hepp and is now partially merged into `Schema.org`. When

[6]http://www.heppnetz.de/ontologies/goodrelations/v1

defining your own schema it is advisable to use `https://purl.org` as vocabulary identifier as it will stay persistent over time.

Together with linked data, semantic elements were introduced in HTML5 standard. These are elements like `<article>`, `<figure>`, `<header>`, `<nav>` and others. These were designed to give meaning to website structure and replace HTML syntax like `<div class="header">`.

As semantic web is getting more widespread (18% of domains crawled by Common Crawl used semantic notation) and with growth of different API-s it is becoming questionable, to we need data extraction from fuzzy semi structured webpages and instead focus on structured data extraction and its meaningful analysis [WPC+15]. But unfortunately, at current time (2016) we still can't live without extracting data from non semantic web pages. Even when semantic markup is used, it is often used partially and describes only part of data made available. Also syntactical mistakes are common.

2.2 Web data extraction

Web pages contain messy data. Modern web page does not only contain main textual and image content, it has also additional content added as header, footer, side area block. These blocks contain navigation links and some times advertisements. Also web page is decorated with markup which only purpose is to improve visual appearance. Vast majority of web content is generated automatically from relational databases. These include Content Management Systems like Wordpress, web forums, online shops. This means that content in its original form is structured. This structured information is embedded into HTML template and decorated with visual information. Figure 2 shows underlying database schema (left) of popular open source e-commerce solution and `timefy.com` online store. In order to retrieve original data, the template must be removed or data must

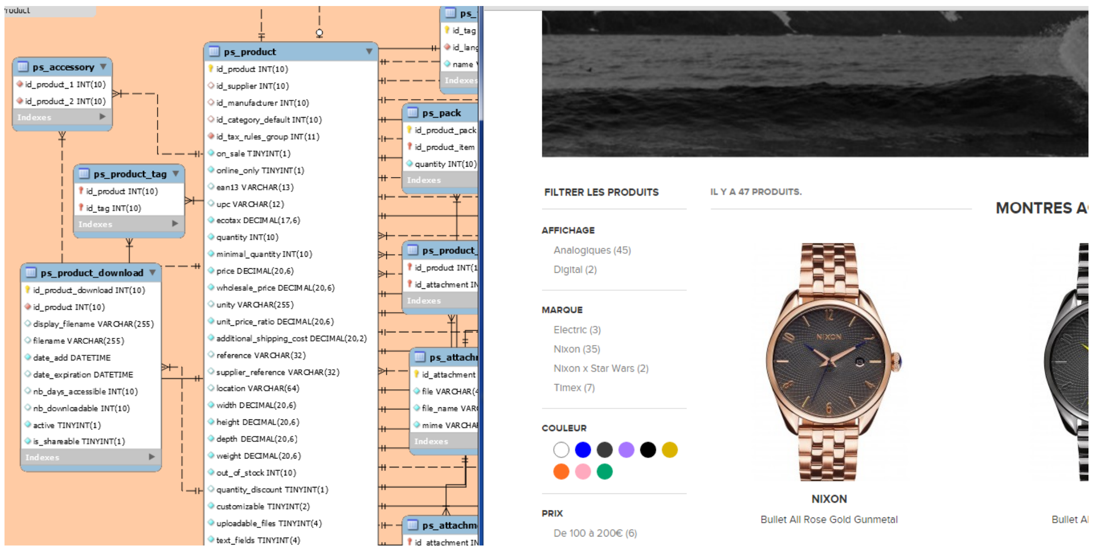


Figure 2: Prestashop database schema and sample online store.

be extracted from it. This is done by wrapper procedure.

2.2.1 Wrappers

A wrapper is a procedure that implements a family of algorithms, that finds the information that user needs, extracts this from an unstructured source and transform them into structured data.

Over the last decade large amount of research and tools have been created that help with web mining task. Multiple studies have been performed to compare existing solutions, these include [KB00], [LRNdST02], [KT02], [CKGS06], [PPPD08] and more recent [SC13], [FDMFB14], [WPC$^+$15]. Also book from Bing Liu gives in detail overview about web mining including web data extraction[Liu11]. Together with web content mining some of these surveys give overview also about other parts of web mining, such as web structure mining and web usage mining.

Data extraction methods can be divided into three segments depending on the level of automation[Liu11]:

- **Manual approach**: Human observers web page and its source code and writes down rules or program code extract data. Also tools that make the process simpler for programmers, such as pattern specification languages and user interfaces are placed to this segment.
- **Wrapper induction**: In this approach supervised learning where extraction rules are learned from a manually labeled data records.
- **Automatic extraction**: This uses unsupervised learning to find repetitive patterns on single or multiple pages. As this is fully automatic, then it can be applied in web scale.

Although it is generally accepted, that manual approach is not scalable to large number of sites, it can be viewed as labeling or annotation task for Wrapper induction or to generate test dataset for automatic extraction validation as it is sill gives most accurate results. And with visual assistance tools this process can be significantly speed up when compared to manually creating extraction rules. This approach also works well, if we have highly structured data, meaning that we can construct wrapper by only observing single web page per site.

2.2.2 Expressing extraction rules

Web pages can be treated as just as stream of characters, structured data where fields are separated using tokens (HTML tags) or as Document Object Model (DOM) that represents HTML page in tree structure. When using HTML page as text document we can use regular expressions (regex) or other standard text extraction methods to extract data we need. For example extracting article headline we can use following rules (Figure 3). When we look web page as DOM tree, then we can use XPath to represent the extraction rules (see Figure 3). Using XPath makes rules simple but we also loose ability to extract partial content inside HTML element.

When creating extraction rules we must ensure that they are general enough to extract data from all pages and specific to only extract data that this rule is designed for. For example when HTML markup for single page contains multiple `H1` tags then we must make the rule less general to pinpoint the field with higher precision.

Several programming languages or language extensions has made to make manual approach faster for programmer. These include Elog [BFG01b] (used in Lixto[BFG01a]),

```
Regex: match('/<H1>(.+)</H1>/')
XPath: evalute('H1')
Selector: querySelector('H1')
Token based: find("<h1>").copy_until("</h1>")
Microdata API: getItems("http://schema.org/Product")[0].properties["
    name"].getValues()
```

Figure 3: Rules for extracting news article headline.

```
Regex: /<div class="articleTitle">\s*<H1 itemprop="headline">(.+)</H1>/
XPath: div[@class="articleTitle"] > H1[itemprop="headline"]
```

Figure 4: More precise rules for extracting news article headline.

OXPath[SFG+11b], Ducky [KT14], Nexir[SWL+13]. Elog uses formal logic and predicates to express extraction rules (5).

```
ebaydocument(S, X) ← getDocument(S = $1, X)
ebaydocument(S, X) ← next( , S), getDocumentOfHref(S, X)
        next(S, X) ← ebaydocument( , S), subelem(S, (⋆.content, [(href, .∗),
                     (elementtext, (next page))]), X)
      record(S, X) ← ebaydocument( , S), subelem(S, .table, X)
                     before(S, X, (⋆.tr, [(elementtext, . ∗ Current.∗)]), 0, 100,  ,  )
                     after(S, X, (⋆.img, [(src, . ∗ spacer.gif)]), 0, 100,  ,  )
     itemdes(S, X) ← record( , S), subelem(S, (⋆.td. ⋆ .content, [(href, .∗)], X)
       price(S, X) ← record( , S), subelem(S, (⋆.td, [(elementtext, \var[Y].∗)]), X),
                     isCurrency(Y)
        bids(S, X) ← record( , S), subelem(S, ⋆.td, X), before(S, X, .td, 0, 30, Y,  ),
                     price( , Y)
        date(S, X) ← record( , S), subelem(S, ⋆.td, X), notafter(S, X, .td, 100)
    currency(S, X) ← price( , S), subtext(S, \var[Y], X), isCurrency(Y)
      amount(S, X) ← price( , S), subtext(S, [0 − 9]+\.[0 − 9]+, X)
```

Figure 5: Elog Extraction Program for linked eBay pages [BFG01b].

OXPath extends XPath with visual features (For example, //span.price[style::color='red']), allows the execution of user actions such as click, navigation through data paginated pages and identification of data for extraction. Figure 6. shows sample script written in OXPath language.

Although OXPath is very powerful it is not very declarative. Same functionality is provided by more declarative language Ducky with addition to post process extracted data with additional rules (see Figure 7).

Nexir rule language is similar to Ducky but is written down in XML instead of JSON and XPath rules are used instead of DOM selectors.

2.2.3 Semi-automatic tools

Semi-automatic tools, such as XWRAP[LPH00], Lixto and others try to make wrapper generation easier by providing user interface to construct refine rules. Early tools were standalone applications with embedded browser window shown side-by-side with web page source code view and separate panel for rule definition (8.a and 8.b).

As Web browsers have become more powerful such tools have been replaced with browser add ons. This allows user to browse website as it would do normally in familiar

```
    doc("http://www.kayak.com/flights")//field()[5]/{$origin}
    /following::field()[@type='text'][1]/{$destination}
    /following::field()[last()]/{click /}
    //tbody[@class~='flightresult'][1]:<flight>/tr[2]
    [td[2]/a:<price=string(.)>] [td[4]:<airline=string(.)>]
```

Figure 6: OXPath for finding the cheapest flights[SFG+11a].

```
  "scraping" : [{
    "url" : "http://www.fcbarcelona.com?/football/first team/staff",
    "selector" : "ul.widget jugador player list li > a",
    "data" : [{
      "field" : "player_name"
    }, {
      "field" : "personal_page_url"
      "attr" : "...",
      "find" : "...",
      "remove" : ["...", "...", ...],
      "replace" : [["...", "..."], ...]
    }],
    "next" : {...}
  }]
```

Figure 7: An example of data extraction from a single page with Ducky[KFT14].

environment and at same time create wrapper for it (8.c). Both these (standalone applications and browser addons) reguire users to download and install something. To avoid this some tools also provide bookmarklet[7]. This allows wrapper generation programm to be embeded into webpage itself allowing to run it in webpage context (8.d)[8]. This approach makes wrapper generation process extremely fast.

Over the time these tools are envolved form simple tools that allow programmer to write rules more easily into semi-automated tools that generate wrappers while labeling data.

2.2.4 Wrapper induction

Wrapper induction (or wrapper generation) solution was proposed by Kushmerick [Kus97] in 1997 to overcome the problem that creating wrappers manually in that era was more time consuming and error prone. Kushmeric defines wrapper induction as a system able to learn data extraction rules from a set of labeled training examples. Labeling is in most cases done manually by simply marking the data items in the training pages and examples. The learned rules are then applied to extract target data from other pages with the same mark-up encoding or the same template. Two key steps of Wrapper induction step is rule generation and rule refinement. In first step is to generate candidate rules for each extracted field. In second step candidate rules are combined together and refined such way that reusability is maximized and maitanence cost is minimized. This means that final rules are just as precise as needed but no more. We can make analogy from

[7]A bookmarklet is a bookmark stored in a web browser that contains JavaScript commands that add new features to the browser

[8]http://www.kimonolabs.com/

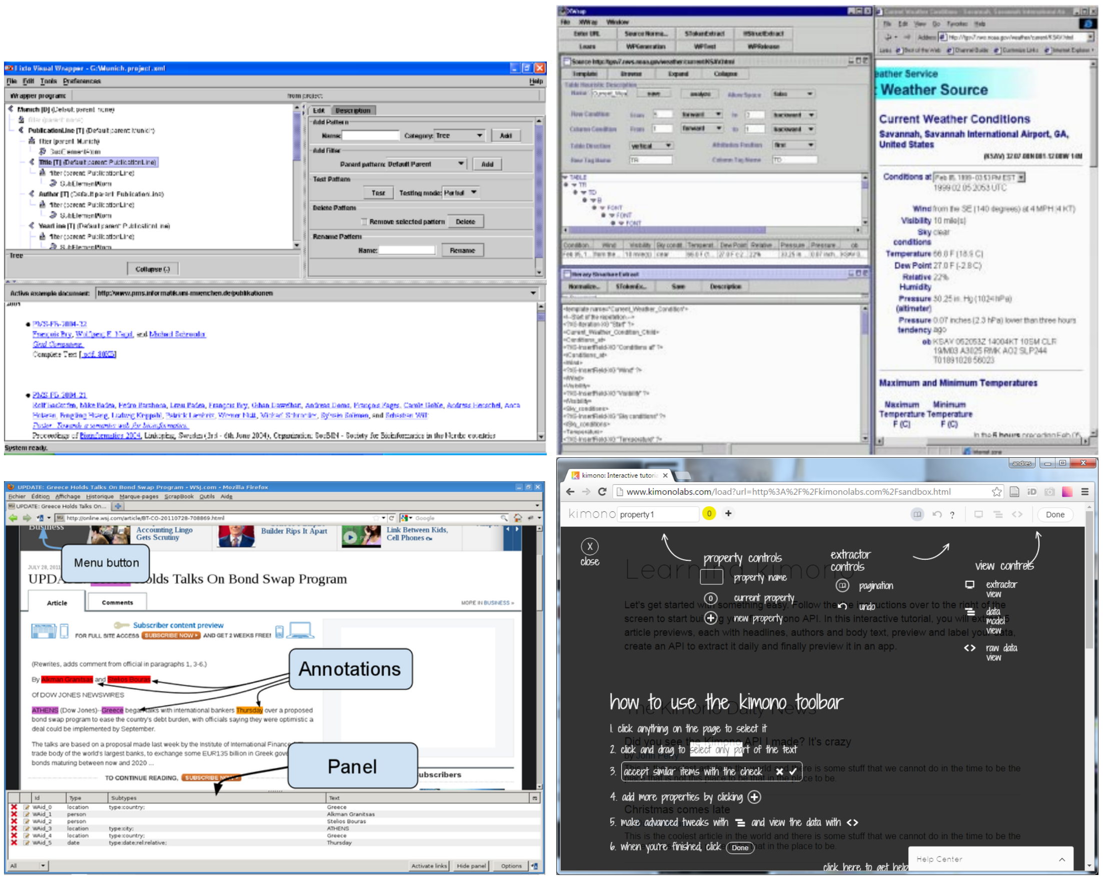


Figure 8: Evolution of Wrapper generation tools (a) Lixto, (b) XWRAP, (c) Webannotator Firefox add-on and (d) Kimono Labs Web UI.

street navigation. Rule that consists only navigation steps like "go forward", "turn left", "turn right" can guide user to destination very accurately but when new road is build or closed then the whole rule becomes useless. But replacing some detail steps with more general ones like "drive to train station" makes this rule more resistant to future changes. So final rule might be "drive to train station", "turn left".

Wrapper Induction starts the life cycle of Wrapper program. During its liftime it might need maitenance as underlying webpage might change its layout and stucture and therefore break existing Wrappper. Kushmerick [Kus00] introduced the concept of wrapper maintenance as the process of verifying the correct functioning of the data extraction procedures [Kus00]. Wrapper maintenance has not been as well studied as induction. but there are some results by Ferrara et al. to automatically repair wrappers with re-induction[FB11].

When it comes to web scale data extraction then manually labeling examples and therefore wrapper induction approaches become to time consuming and fully automatic solutions are preferred.

2.2.5 Automatic Wrapper Generation

Much of current research effort is going into developing fully-automatic structured web data extraction techniques. These can be divided into three categories: pattern search based, visual signals aided, ontology based[GRI14].

- **Pattern mining approaches** try to generate extraction rules automatically by mining patterns from one or more web pages. In recent years, many studies focused on such approaches because such approaches can reduce human labor in the largest extent. This is the oldest and most studied approach.
- **Visual signal aided** approaches view webpage as regions. Similar and repetitive regions are clustered together and is then classified as data record. [SC13]
- **Ontology based systems** extract data using domain (such as cars, books, real estate, flights and etc.) knowledge. If we are looking for cars we know it mus have brand, model, color, type(hatchback or sedan), etc. Classification task is performed to identify these fields in web page. This approach seems most promising but its weakness are lack of domain knowledge and they are usually designed for single language. One of the best examples is the DIADEM system [FGG+12]

In general, drawbacks of fully automated approaches are limited expressive power and the large number of required example pages. In case of systems that do not rely on labelled examples the main drawback is the low percentage of correctness of the extracted data. [BEG+05]

2.3 Web crawling

World Wide Web (WWW) is huge collection of documents that are linked together using hyperlinks. According to Internet Live Stats website[9] there was over 900 million websites online as of 2014.

Web crawling is usually done by taking set of pages (manually gathered) as starting point, called seed pages. Seed pages are processed one at a time and new web pages are discovered by extracting links on them. Web page addresses (URL-s) are stored in queue. There are two types of crawlers depending of the data structure used behind the queue [Liu11]. Breath-first crawlers use FIFO and Preferential (also called Focused) Crawlers use priority queue. Breath-first crawlers tend to be biased because more popular web pages are more links into them and these are crawled first. For vertical search engine we do not want crawl the entire Web only the pages that follow our topic of interest. When main focus is content extraction using wrappers (Described in section 2.2) then crawling is usually limited to single site, but still same concerns apply. Minimizing the amount of data downloaded and indexed is always a good idea.

2.3.1 Focused crawlers

When data extrction rules are create manually using some declarative language then navigation rules are also written to guide crawler to target pages without spending much time on irrelevant pages. For example navigation rules can be specified with Ducky and Nexir. With wrapper induction and fully automated approaches navigation information is not available prior crawling and therefore heuristics that guide the crawler must be gathered during crawl time.

Different approaches to this problem have been studied and experimented. Common idea behind them is to classify pages into two groups

[9]http://www.internetlivestats.com/total-number-of-websites/

1. **Content pages** - these are pages that are targeted for data extraction and must be crawled.

2. **Navigation pages** - pages that might get crawled in order to reach target page.

Pages can classified based on their content or structure, but also by analyzing how they are linked together. Many authors [GLSRN00, DH99] have noticed that links on web pages that are close to each (share same parent node) other link to topically similar content. The same assumption holds for webpage stucture as is shown by Crescenzi et al. [CMM05]. They propose an algorithm that builds site model by crawling Web site. The structural similarity can be also determined by comparing the placements of link collections. They observe and validate that links sharing the same layout and presentation properties usually point to pages that are structurally similar. Same idea is also employed by Lin et al. to hierarchically cluster Web pages[LYHL10]. Grigalis et al. further extend these ideas. They have used XPaths of links as base for clustering and show that classification accuracy more than 90% is achievable[GC14].

Other approach use sampling and construct site map to determine optimal traversal paths to content of interest. This is basics for iRobot crawler[CYL^{+}08]. Probabilistic models such as Hidden Markov Models(HMMs) and Conditional Random Fields(CRFs) are also used and as demonstrated by Liu et al. in [LMJ04] to show good results in the decrease of number of pages crawled.

3 ZedBot information extraction system

Following section gives detail overview of information data extraction system ZedBot that enables extraction of structured data from semi stuctured webpages.

Baumgartner and Weninger have described key properties that modern data extraction systems should have. We hereby quote the list of important points given in works[BEG+05, WPC+15]:

- **High expressive power**. The system should enable the definition of complex, structurally organized patterns from Web pages and translate the data (the so-called pattern instances) into a corresponding hierarchically structured document.
- **Robustness**. Generated wrapper should remain functional to minor structural changes in target web page (such as introduction of a new banner).
- **Runtime Efficiency**. The method should provide efficient algorithms and the system should implement these algorithms efficiently such that the system becomes usable in practice and is highly scalable.
- **n-Dimensional Data Structures**. In many cases it is not sufficient to generate XML data comprising two levels, i.e. representing a relational table. In general, wrapper output shall support arbitrarily nested XML output data.
- **Semantic Web Interface**. A good wrapper generation system shall be able to populate ontologies with instance data and even extend an ontology based on concepts extracted from the Web.
- **Platform Independence**. For integration into a mediation framework supported platforms might be a decision criteria.
- **User friendliness**. It should allow a human wrapper designer to design, program, or specify wrappers in a very short time.
- **Good learnability**. The learning effort for being able to understand the method or use the system should be as small as possible. The method or system should be accessible to and usable by a non-technical content manager who is not a programmer or a computer scientist.
- **Good visual support**. It should offer the wrapper designer a GUI for specifying wrappers. Ideally, the visual user interface allows a wrapper designer to work directly on displayed sample source documents (e.g. on HTML Web pages) and supports a purely visual way of defining extraction patterns.
- **Ease of accessibility and installation**. The system should be widely accessible and should not require particular installation efforts. Ideally, the system provides an interface so that it uses a standard Web browser.

3.1 System architecture

ZedBot data extraction system was designed based on requirements presented above. The system consists of two standalone components.

- **Information extraction tool** called ZedBot Scraper. It is the core component of the system, it has following functionality:
  - **Page annotation** - Allows user to click any text or image on web page and define meaning to it using predefined schema.
  - **Wrapper generation** - Given selected text or image, extraction rule for it is generated.
  - **Wrapper execution** - Executes all extraction rules for given page and returns structured JSON document that is consistent with schema used for annotation.
- **Web Crawler** called ZedBot Crawler, a component to automate extraction from multiple web pages from single site. Its main functionality is do download pages from target website, load each web page in embedded browser and attach scraper into it so that data extraction is performed. It also extracts links from loaded web page and saves them into queue for later downloading.

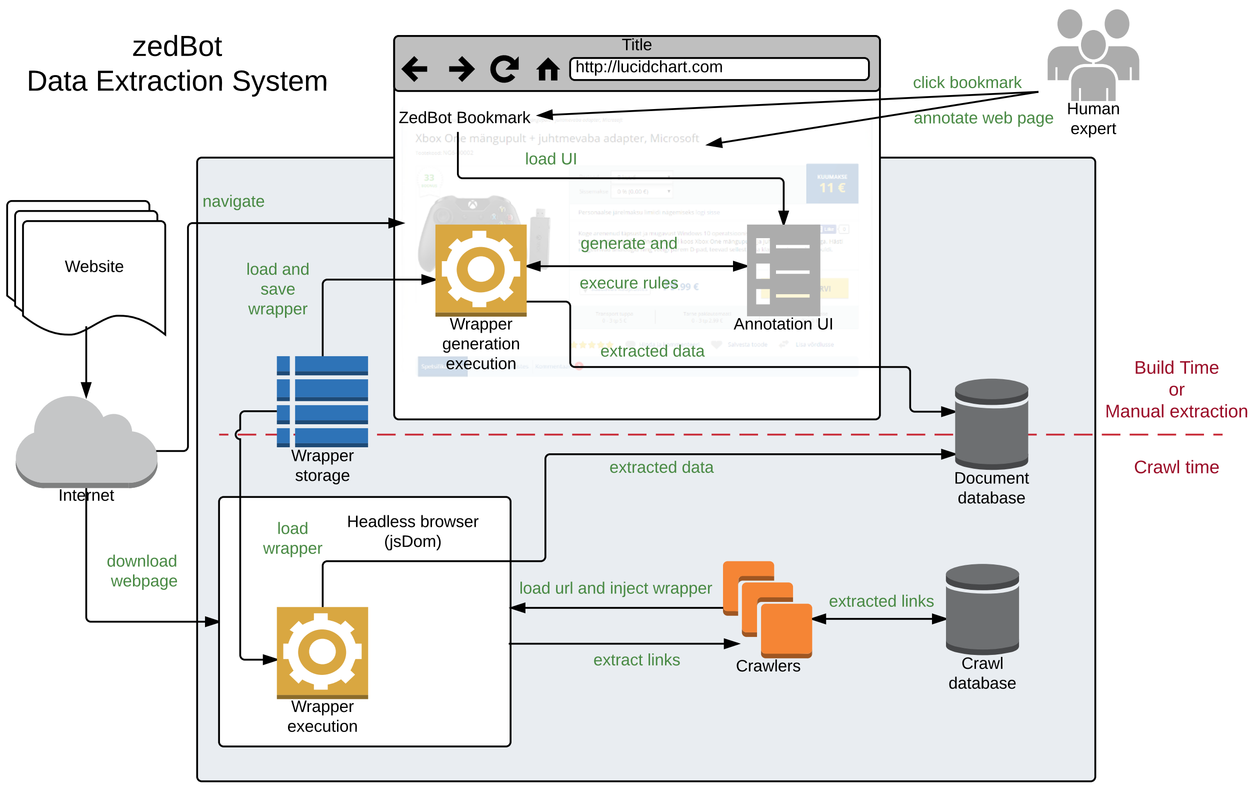


Figure 9: System components and data flow.

System components and interaction between them is shown in Figure 9. It shows two operating modes:

- **Manual mode**, where human expert creates wrapper and tests it. Or can be also used for manual execution of the wrapper.

- **Crawl mode**, this is automatic mode where wrapper is executed simultaneously while crawling.

3.2 Information extraction tool

This tool is built as Bookmarklet[10]. This allows to start the tool inside browser while browsing in target website.

Installation of the software is done by bookmarking on a link form ZedBot homepage. User will then navigate to target web page from where he wishes to extract data and click on saved bookmark.

Main user interface is displayed in Figure 10. Upon executing bookmarklet the tool is injected into partner website and UI components are loaded. First step is to create

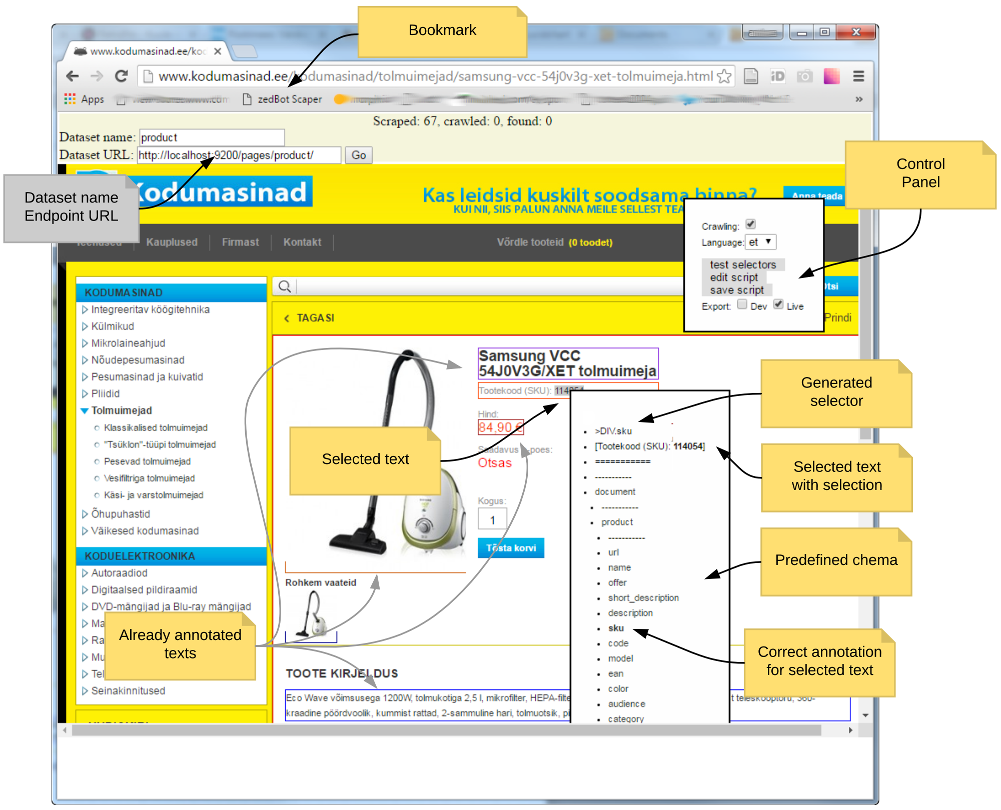


Figure 10: Annotation and wrapper generator UI.

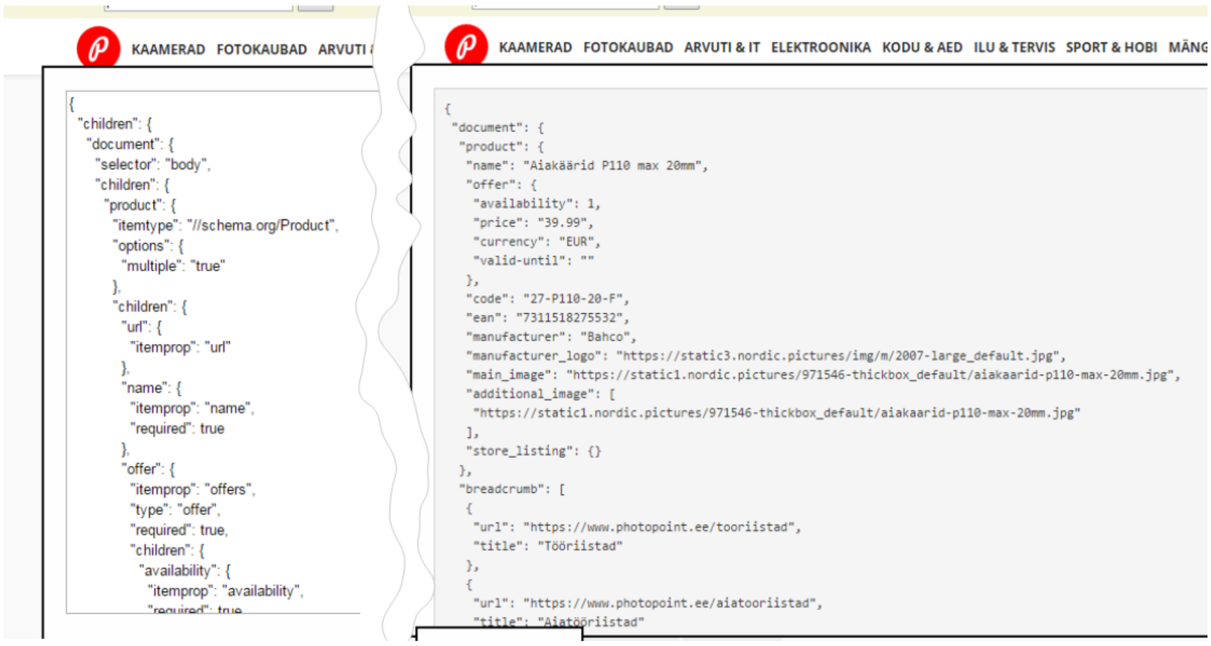
new dataset. This allows the data that is extracted from different websites to be placed in same table. Together with dataset user specifies the endpoint of database URL where extracted data is saved. Any database that supports restful API can be used. When dataset and endpoint is specified, tool is initialized. Existing predefined schemas are loaded in background.

Second step is to annotate data on target web page. User has option to select any text and images on target web page and by right clicking he or she can select correct annotation for it, again based on selected schema. After annotating a field the extraction

[10] A bookmarklet is a bookmark that contains JavaScript commands which add new features to the browser.

rule (selector) is generated and added to wrapper. Generated rule is also displayed to user (shown inside same annotation pop-up together with the text user selected). This step is repeated until all data is annotated.

At any point user can click on `<test selectors>` button in Control Panel to view extracted data (see Figure 11.a). If extracted result is not correct or optimal then user can edit the generated wrapper manually by clicking `<edit scipt>` button (see Figure 11.b). When user is satisfied with extracted results then he or she can save the generated wrapper.



KAAMERAD FOTOKAUBAD ARVUTI
KAAMERAD FOTOKAUBAD ARVUTI & IT ELEKTROONIKA KODU & AED ILU & TERVIS SPORT & HOBI MÄNG

```
{
 "children": {
  "document": {
   "selector": "body",
   "children": {
    "product": {
     "itemtype": "//schema.org/Product",
     "options": {
      "multiple": "true"
     },
     "children": {
      "url": {
       "itemprop": "url"
      },
      "name": {
       "itemprop": "name",
       "required": true
      },
      "offer": {
       "itemprop": "offers",
       "type": "offer",
       "required": true,
       "children": {
        "availability": {
         "itemprop": "availability",
         "required": true
```

```
{
 "document": {
  "product": {
   "name": "Aiakäärid P110 max 20mm",
   "offer": {
    "availability": 1,
    "price": "39.99",
    "currency": "EUR",
    "valid-until": ""
   },
   "code": "27-P110-20-F",
   "ean": "7311518275532",
   "manufacturer": "Bahco",
   "manufacturer_logo": "https://static3.nordic.pictures/img/m/2007-large_default.jpg",
   "main_image": "https://static1.nordic.pictures/971546-thickbox_default/aiakaarid-p110-max-20mm.jpg",
   "additional_image": [
    "https://static1.nordic.pictures/971546-thickbox_default/aiakaarid-p110-max-20mm.jpg"
   ],
   "store_listing": {}
  },
  "breadcrumb": [
   {
    "url": "https://www.photopoint.ee/tooriistad",
    "title": "Tööriistad"
   },
   {
    "url": "https://www.photopoint.ee/aiatooriistad",
    "title": "Aiatööriistad"
```

Figure 11: (a) Generated Wrapper and (b) Extrated data.

3.2.1 Schema definition

Annotation and rule generation is guided by schema defined by user. This assures that data extracted from variety of web pages follows exactly the same structure and user can't make spelling mistakes while labelling data. Schema must be created prior annotation. Figure 12. shows subset of product schema. Schemas can be nested together to form very complex data structures. In the example schema "offer" and "reivew" properties have defined separately. This feature is lacking in many annotation and data extraction tools and was one of limiting factors for using existing tools even for annotation. Multiple options can be specified for field in schema, these are described in "Rule generation" section.

3.2.2 Semantic Web and microdata

As seen from Figure12. `schema.org` microdata can be embedded into defined schema. This allows easily extract structured data. Microdata elements can be added as extensions to user defined fileds. This approach was chosen to allow microdata fields to be mapped to user fileds. This is mostly useful if target schema is fixed and does not perfectly align with mircodata schema.

```
{
  "itemtype":"//schema.org/Product",
  "children": {
    "url": {  "itemprop":"url"  },
    "name": { "itemprop":"name" },
    "offer":{
      "itemprop":"offers",
      "type":"offer"
    },
    "manufacturer": {   "itemprop":"brand"  },
    "manufacturer_logo": {  "itemprop":"logo" },
    "main_image": {   "itemprop":"image"  },
    "price": {},
    "review": {
      "itemprop": "review",
      "type": "review",
      "options": {
        "multiple": true
      }
    }
  }
}
```

Figure 12: Subset of product schema.

3.3 Rule generation

3.3.1 Simple rules

As mentioned in the overview there are number of ways how to specify rules for extracting. When rules are generated from DOM tree then there are two basic options, either by using selectors or XPaths. As selectors are more expressive (see Figure 13.) then this approach was choosen.

```
Selector:
  #aspnetForm > div.oi section wrap > header > div.oi section main
      navigation > nav > ul > li.oi item.oi item 3.has submenu > span >
      span
XPath:
  //*[@id="aspnetForm"]/div[3]/header/div[4]/nav/ul/li[3]/span/span
```

Figure 13: Comparison between selector and XPath.

Rule generation (Algorithm 1) works recursively bottom up form selected node until parent node is reached. First candidate rules are generated. Candidate rules are generated starting from most robust one such as tag name (A, DIV, P, HEADER) and ending with more precise selectors such as DIV.row:nth-child(3). Next candidate list is traversed and first selector that matches node uniquely is selected. When selector for current node is selected then algorithm moves level up and generates new selector from parent node.

3.3.2 Data refinement rules

- **regex**. Athough DOM selectors are powerful tool for representing rules, they sometimes are not sufficient. This is mostly in case when we want to extract few words

Algorithm 1: Algorithm for creating DOM selector for selected node.

```
Function generateSelector (node, childSelector, scopeParen
    Output: selecor
    if $node == parent$ then
        return childSelector;
    end
    $selector \leftarrow ""$ ;
    $candidateSelectors \leftarrow getCandidateSelectors(node, childSelector, parent)$;;
    foreach candidateSelector in candidateSelectors do
        if testUniqueness(node, candidateSelector, node.parentNode) then
            $selector \leftarrow candidateSelector$ ;
            break;
        end
    end
    return generateSelector(node.parentNode, selector, scopeParent);
```

from sentence. For example product code node might contain some extra textual content (Figure 14.1). In this case user when selects only partial text like ”$Tootekood(SKU)$: 451180” then regular expression is generated from DOM selection object[11] from where we can read selection start and end positions. For this example following regular expression is generated $/Tootekood\backslash(SKU\backslash) : (.+)/$.

- **replace**. Replace rule allows replace text in extracted data. This is useful when creating simple regular expression is not possible. For example this can be used to convert URL on thumbnail image into URL of full size image.
- **map**. This rule allows map extracted value into predefined new value. It can be used to convert microdata values such as "InStock" and "OutofStock"[12] into numeric values "1" and "0" (see Figure 14.2).
- **use_content**. For link nodes href attribute is extracted by default. This rule allows textual content to be extracte instead (see Figure 14.3).
- **use_attribute**. Similar to previous rule this allows to extract data from specified attribute instead of the element content.

```
1) <div class="sku">Tootekood (SKU): 451180</div>
2) <link itemprop="availability" href="http://schema.org/InStock" />
3) <a href="#popup">Telia Tallinna esindus(Lasnam\\"{a}ae Centrum)</a
   >
```

Figure 14: Example data that must be extracted with complex rules.

[11]https://developer.mozilla.org/en-US/docs/Web/API/Selection
[12]https://schema.org/ItemAvailability

3.3.3 Structure transformation rules

- **textify** Dom selectors allow only select element nodes. But HTML standard allows element and text node to be interleaved (Figure 15 top). This means there is not possible to generate selector for text nodes. To overcome this problem all text nodes are automatically wrapped with span element.
- **unflatten** This is complex and powerful rule. It allows to restore hierarchical structure from flat one like is shown in Figure 16. First nodes are read sequentially and when group separator (specified in rule, H3 in the example) is observed then siblings following it are grouped together until next group separator. Groups are placed inside DIV element.

```
<p>
  <b>Photopoint, Lounakeskus:</b>
  Ringtee 75, Tartu
</p>
```

```
  <p>
  <b>Photopoint, Lounakeskus:</b>
  <span class="zedbot_text">Ringtee 75, Tartu</span>)
  </p>
```

Figure 15: Text node is wrapped with span element.

3.3.4 Executing rules

Rules are executed in top down manner while traversing the wrapper rule file. DOM tree and rules (Figure 11.a) are traversed together side-by-side. Wrapper starts with list of rules, root node of HTML document and empty object. These are passed to extractFieldBySelector function (see Algortithm 2) which recursively traverses both rule list and HTML nodes and fills initially empty object with extracted data (Figure 11.b).

When rules are applied, the total number of matched rules are counted. Together with number of total rules specified this gives us score how precise data extraction was.

$$score = \frac{\#matches}{\#rules}$$

This can be used by crawler to decide not to save documents which have score less than 0.5. Also it makes possible to detect when web page has changed and and wrapper does not work anymore. When same page is re crawled over some period of time we can compare old score with new one and if change is big enough then mark this wrapper as invalid.

3.4 Crawler

For crawling purposes a custom crawler was built. This option was chosen because creating lightweight crawler for is easier than to install and configure fully scalable crawler, such as Apache Nutch[13]. Also the requirement was that rule execution code will be

[13]http://nutch.apache.org/

```
<div>
  <h3>Kristiine Euronics</h3>
  <p>Kristiine kaubanduskeskus; Tallinn, Endla 45</p>
  <p>Avatud E P 10 21</p>
  <h3>Lasnamae Euronics</h3>
  <p>Lasnamae Centrum<br>Tallinn, Mustakivi tee 13</p>
  <p>Avatud E P 10 21</p>
  <h3>Rocca al Mare Euronics</h3>
  <p>Rocca Al Mare kaubanduskeskus<br>
  ...
</div>
```

```
<div>
  <div>
    <h3>Kristiine Euronics</h3>
    <p>Kristiine kaubanduskeskus; Tallinn, Endla 45</p>
    <p>Avatud E P 10 21</p>
  </div>
  <div>
    <h3>Lasnamae Euronics</h3>
    <p>Lasnamae Centrum<br>Tallinn, Mustakivi tee 13</p>
    <p>Avatud E P 10 21</p>
  </div>
  ...
</div>
```

Figure 16: Non-hierarchic data structure (top) is converted into hierarchic data structure (bottom).

shared between crawler and wrapper generator tool. This allows any improvements made in wrapper tool to be instantly available for crawler to use. Crawler is built also in JavaScript programming language and runs in NodeJS[14] framework. Instead on fully featured browser crawler uses headless browser jsdom[15] that implements most modern DOM and HTML standards in javacripts.

Separate crawler is launched for each site. Although each single crawler is single threaded program, but executing multiple crawlers simultaneously allows effective use of computing resources.

Each crawler has its own queue from where unvisited urls are retrieved. Urls can be processed in random order or sorted by score function (See Link extracrion subsection). Limited number of urls are proccessed at a time then process exits and is restarted by manager process if there is more work to do.

Overall process is shown in Algorithm 3. Url is loaded into browser like environment and the same bookmarklet is attached to target page. Wrapper is therefore executed in target webpage context and extracts data exactly the same way as was done when testing wrapper in its creation process. Extracted data is returned to crawler and if extraction score is over threshold then data is saved into user specified dataset. Next step is to extract additional links from webpage so that crawling would not stop at first page.

[14]https://nodejs.org/
[15]https://github.com/tmpvar/jsdom

Algorithm 2: Crawler algorithm for crawling single site.

```
1  Function extractFieldBySelector (wrapper, parent, out object)
       Input: wrapper - user defines schema with extraction rules
       Input: parent - DOM node from where selectors are applied
       Output: object - reference to parent object field where extracted data is saved
2      ;
3      foreach field in wrapper do
4          $node \leftarrow null$;
5          if field has selector then
6              $node \leftarrow querySelector(field.selector, parent)$;
7          else
8              if field has microdata property then
9                  $node \leftarrow getMicrodataNode(field.microdata, parent)$;
10             end
11         end
12         if node is not null then
13             if field has children then
14                 $object[field.name] \leftarrow \{\}$; extractFieldBySelector(field, node,
                    object[field.name]);
15             else
16                 $object[field.name] \leftarrow getNodeValue(node)$;
17             end
18         end
19     end
```

3.4.1 Link extraction

Link extraction is straight forward for universal crawler that picks urls at random. But this is not very effective when pages with extractable information is subset from all pages (Figure 17. left) or even extremely ineffective when links to non content pages are generated dynamically (Figure 17. right). For example link to add product into shopping cart or wish list. In majority of sites these kind of actions are done by form submission or handling click event in JavaScript. This will hide these actions from crawler. This is beneficiary to site owner as well as less load is generated to server by crawlers.

To solve this issue we use similar approach that was proposed by Grigalis et al. [GC14]. We exploit the fact that links that are in same location in different pages all point to similar content. We indentify links that lead to content pages. When such link is identified we save its location in referring page together with score (got from extracting current page) into hash table. When new page is processed and links are extracted we look hash table to see is score is associated with link XPath. If it is then we add all links that are nearby (they have same XPath) into crawl queue with matching score. If XPath was not found or score is below threshold we only add one link. See Algorithms 4., 5. and 6.

Algorithm 3: Data extraction function.

```
Input: siteName, datasetName
1  $links \leftarrow getUnvisitedLinks(siteName)$
2  foreach $link$ in $links$ do
3      load url in jsdom environment;
4      attach bookmarklet to loaded document;
5      $extractedData \leftarrow wrapper.extract()$ ;
6      if $extractedData.score > extractionThreshold$ then
7          saveToDataBase('pages', datasetName, extractedData) ;
8      end
9      $extractLinks(link, document)$ ;
10     link.processed = true ;
11     saveToDataBase('links', link) ;
12 end
```

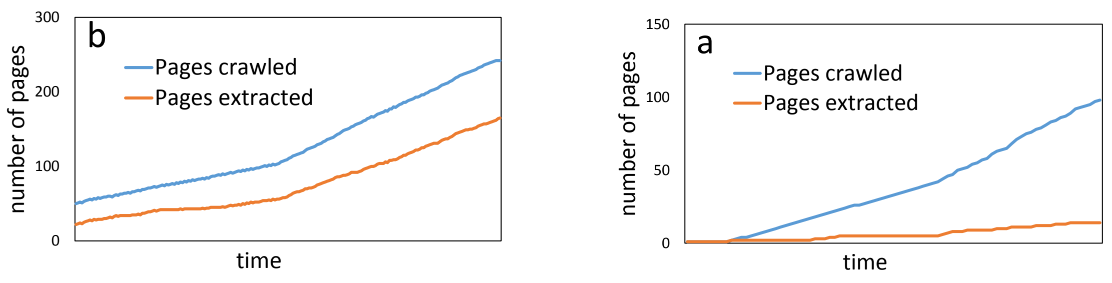
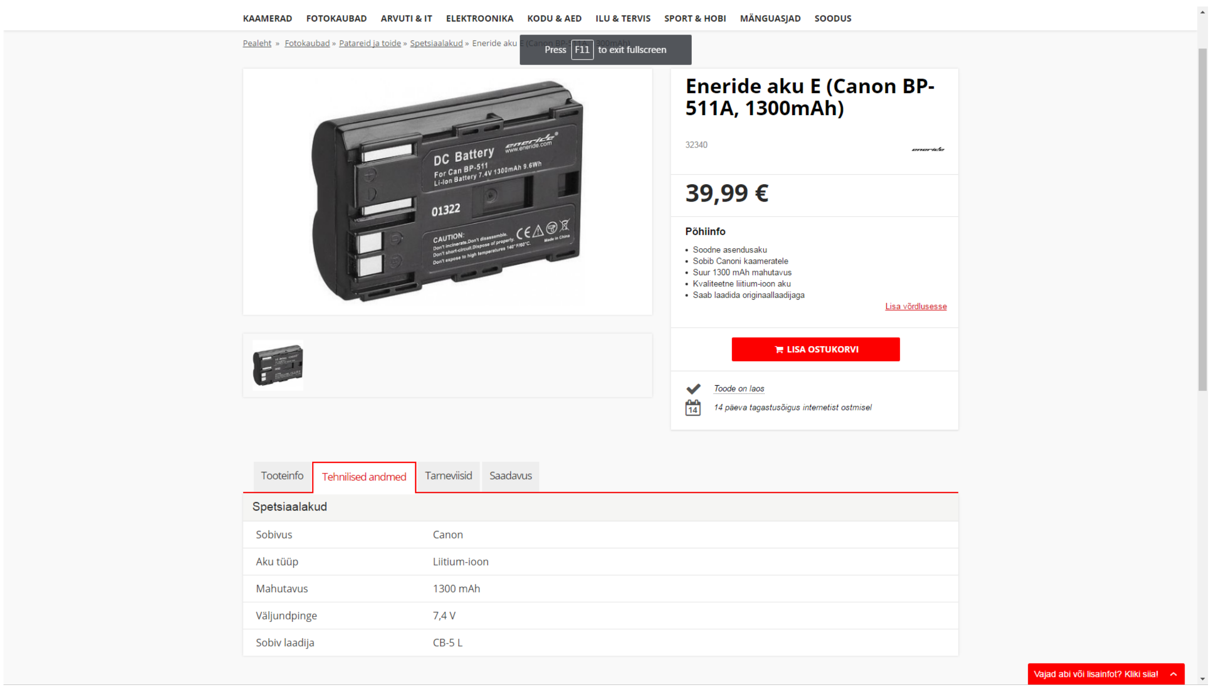


Figure 17: Data extraction with random crawler. a) Normal site, b) site with dynamic links.

4 Results and discussion

Described web data extraction tool was successfully used to gather high precision (see Figure 18.) product data from Estonian e-commerce websites. Together with detail product information, item availability in local stores were extracted. And also store locations were extracted.

Most requirements presented at the beginning of methodology section were achieved. It allows to define of complex, structurally organized rules to transform Web pages into hierarchically structured document with predefined schema. Semantic Web embeddings are automatically detected and extracted and. Created system is platform independent, only requirement is Web browser that implements modern Web standards. Wrapper generation user interface is easy to use and usable to non-technical user for extracting data with simple structure. Extraction of complex structure still requires expert user. Tool has good visual support, wrapper is designed directly on top of displayed source documents and supports visual way of defining extraction rules. System is widelyaccessible and does not require installation efforts, it uses a standard Web browser.



KAAMERAD FOTOKAUBAD ARVUTI & IT ELEKTROONIKA KODU & AED ILU & TERVIS SPORT & HOBI MÄNGUASJAD SOODUS

Pealeht » Fotokaubad » Patareid ja toide » Spetsiaalakud » Eneride aku

Press F11 to exit fullscreen

Eneride aku E (Canon BP-511A, 1300mAh)

32340

39,99 €

Põhiinfo

- Soodne asendusaku
- Sobib Canoni kaameratele
- Suur 1300 mAh mahutavus
- Kvaliteetne liitium-ioon aku
- Saab laadida originaallaadijaga

Lisa võrdlusesse

LISA OSTUKORVI

Toode on laos

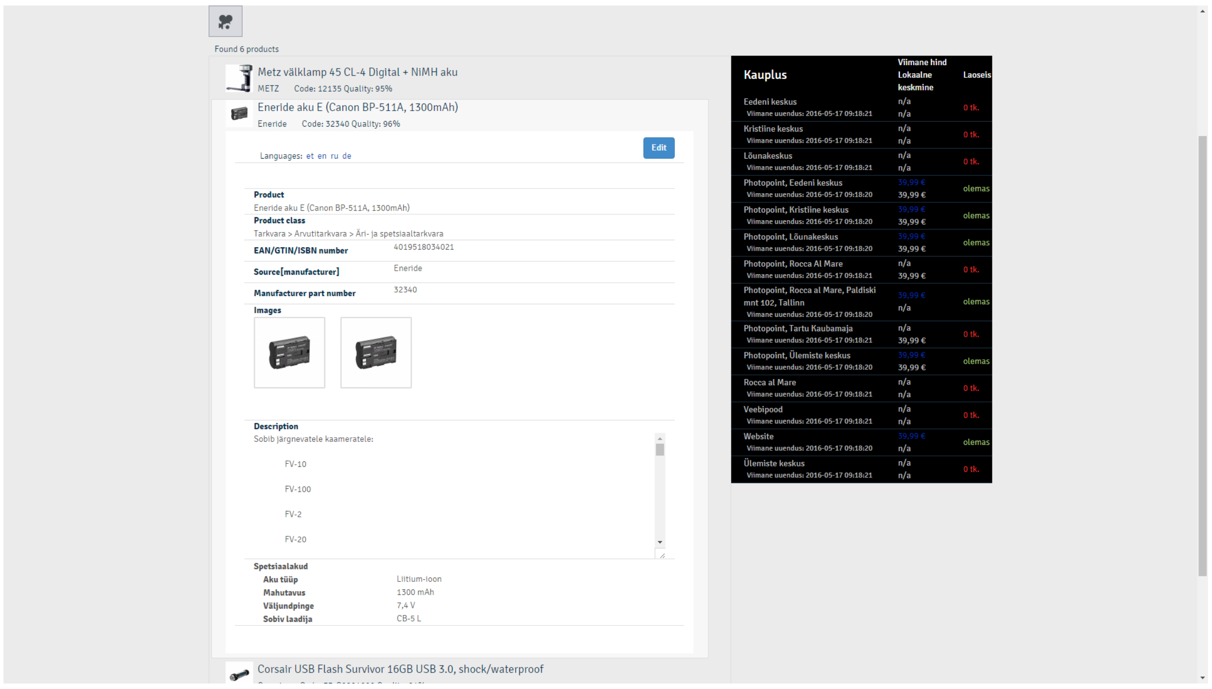
14 päeva tagastusõigus internetist ostmisel

Tooteinfo | Tehnilised andmed | Tarneviisid | Saadavus

Spetsiaalakud

Sobivus	Canon
Aku tüüp	Liitium-ioon
Mahutavus	1300 mAh
Väljundpinge	7,4 V
Sobiv laadija	CB-5 L

Vajad abi või lisainfot? Kliki siia!



Found 6 products

Metz välklamp 45 CL-4 Digital + NiMH aku
METZ Code: 12135 Quality: 95%

Eneride aku E (Canon BP-511A, 1300mAh)
Eneride Code: 32340 Quality: 96%

Languages: et en ru de

Edit

Product
Eneride aku E (Canon BP-511A, 1300mAh)

Product class
Tarkvara > Arvutitarkvara > Äri- ja spetsiaaltarkvara

EAN/GTIN/ISBN number	4019518034021
Source[manufacturer]	Eneride
Manufacturer part number	32340

Images

Description
Sobib järgnevatele kaameratele:

FV-10

FV-100

FV-2

FV-20

Spetsiaalakud

Aku tüüp	Liitium-ioon
Mahutavus	1300 mAh
Väljundpinge	7,4 V
Sobiv laadija	CB-5 L

Corsair USB Flash Survivor 16GB USB 3.0, shock/waterproof

Kauplus	Viimane hind Lokaalne keskmine	Laoseis
Eedeni keskus Viimane uuendus: 2016-05-17 09:18:21	n/a n/a	0 tk.
Kristiine keskus Viimane uuendus: 2016-05-17 09:18:21	n/a n/a	0 tk.
Lõunakeskus Viimane uuendus: 2016-05-17 09:18:21	n/a n/a	0 tk.
Photopoint, Eedeni keskus Viimane uuendus: 2016-05-17 09:18:20	39,99 € 39,99 €	olemas
Photopoint, Kristiine keskus Viimane uuendus: 2016-05-17 09:18:20	39,99 € 39,99 €	olemas
Photopoint, Lõunakeskus Viimane uuendus: 2016-05-17 09:18:20	39,99 € 39,99 €	olemas
Photopoint, Rocca Al Mare Viimane uuendus: 2016-05-17 09:18:21	n/a 39,99 €	0 tk.
Photopoint, Rocca al Mare, Paldiski mnt 102, Tallinn Viimane uuendus: 2016-05-17 09:18:20	39,99 € n/a	olemas
Photopoint, Tartu Kaubamaja Viimane uuendus: 2016-05-17 09:18:21	n/a 39,99 €	0 tk.
Photopoint, Ülemiste keskus Viimane uuendus: 2016-05-17 09:18:20	39,99 € 39,99 €	olemas
Rocca al Mare Viimane uuendus: 2016-05-17 09:18:21	n/a n/a	0 tk.
Veebipood Viimane uuendus: 2016-05-17 09:18:21	n/a n/a	0 tk.
Website Viimane uuendus: 2016-05-17 09:18:20	39,99 € n/a	olemas
Ülemiste keskus Viimane uuendus: 2016-05-17 09:18:21	n/a n/a	0 tk.

Figure 18: Data in original webpage (top) and in extracted form (bottom).

Algorithm 4: Improved data extraction function.

```
Input: siteName, datasetName
1  $links \leftarrow getUnvisitedLinks(siteName)$
2  foreach $link$ in $links$ do
3      load url in jsdom environment;
4      attach bookmarklet to loaded document;
5      $extractedData \leftarrow wrapper.extract()$ ;
6      if $extractedData.score > extractionThreshold$ then
7          saveToDataBase('pages', datasetName, extractedData) ;
8          $hastable[link.referer.xpath] \leftarrow extractedData.score$;
9      else
10         $hastable[link.referer.xpath] *= 0.8$;
11     end
12     $extractLinks(link, document, extractionThreshold)$ ;
13     link.processed = true ;
14     saveToDataBase('links', link) ;
15 end
```

4.0.1 Crawler Performance

Extraction was greatly speed up after implementing focused crawling algorithm. Figure 19. shows difference between random and focused crawler. It is clearly visible that with focused crawler more relevant pages are downloaded and less time is spent following other pages.

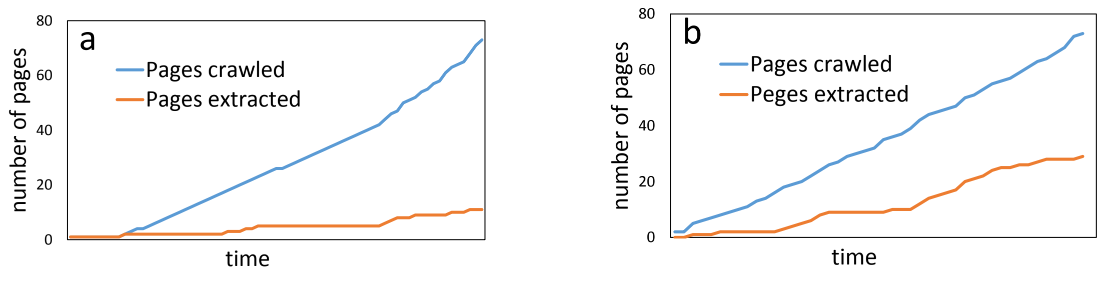


Figure 19: Data extraction with random crawler (a) and with focused crawler (b).

4.0.2 Extracted dataset

More than 50000 product data records from 34 Estonian e-commmerce web sites were extracted and same amount of wrappers were generated. Wrapper creation time depended on the amount of different data presented in web page and usage of semantic data. Fair amount of web pages required to use additional transformation rules and therefore made the wrapper creation process more time consuming. For some web pages it was decided to extract only limited set of data as extraction rules were to difficult to maintain for

Algorithm 5: Link extraction from crawled web page.

```
1  Function extractLinks (referer, document, extractionThreshold)
       Global: hastable, threshold
2      $clusters \leftarrow clusterLinks(document)$;
3      foreach xpath, links in clusters do
4          foreach link in links do
5              $link.referer \leftarrow referer$;
6              $link.xpath \leftarrow xpath$;
7              $link.discovered \leftarrow now()$;
8              $link.score = hastable[xpath].score$;
9              saveToDataBase('links', link) ;
10             if hastable[xpath] < extractionThreshold then
11                 break;
12             end
13         end
14     end
```

Algorithm 6: In page link clustering.

```
1  Function clusterLinks (url, document)
2      $nodes \leftarrow document.querySelectorAll("a")$ $clusters \leftarrow \{\}$ $templateId \leftarrow ""$
        foreach nodeinnodes do
3          xpath = getXPath(node);;
4          $clusters[path].push(node.href)$;
5      end
6      return clusters;
```

all pages. This was usually the case when very little HTML formatting was used to decorate structured data and rather natural language processing tools should have used. Table 1. gives summary of limited set of crawled websites and brings out following key features that affected the creation of wrapper to extract product information. From this small subset we see that one third of websites have adopted semantic web features. But semantic meaning has given only to subset of product data and therefore fully automatic extraction was not possible on any of these sites.

- **Embedded microdata** indicates if semantic web attributes were present allowed automatic extraction of structured data.
  - yes - microdata was used
  - no - microdata was not used
- **Full product specification** indicates if detailed product parameters were present.
  - yes - full specification was presented
  - limited - only basic product features were presented (color, dimensions)
  - no - only product name, minimal description and image were presented

Table 1: Features available on crawled web pages.

site	microdata	product spec	product identification	store stock	wrapper-complexity
euronics.ee	yes	yes	yes*	limited	low
kodumasinad.ee	no	yes	yes*	limited	medium
miterassa.ee	no	yes	yes*	limited	medium
rigonda.ee	no	no	yes*	yes	high
prismamarket.ee	yes	limited	yes	no	low
bauhof.ee	no	limited	no	yes	medium
charlot.ee	no	yes	yes	yes	medium
vunder.ee	yes	limited	yes	yes	high
elion.ee	yes	yes	limited	limited	low
onoff.ee	no	limited	limited	yes	medium
kodumasin.ee	no	yes	limited	no	medium
elektrikaup.ee	no	no	no	no	medium
cdmarket.ee	yes	yes	yes	no	low
photopoint.ee	yes	yes	no	yes	low
koduekstra.ee	no	no	no	yes	high
itshop.ee	no	yes	limited	yes	medium

- **Product identification** indicates if product could be uniquely identified, either by GTIN/EAN code or brand name and model code.
  - yes - GTIN/EAN code was displayed
  - yes* - both product model and brand name was displayed
  - limited - only product model or brand name was displayed but not both
  - no - only product name was displayed
- **Item availability listing** indicates if physical store listing was presented with item availabilty.
  - yes - exact amount of items aviable was shown
  - limited - in stock and out of stock information was displayed
  - no - no item aviablity was displayed
- **Wrapper complexity** indicates how much manual work was needed to create wrapper.
  - none - extraction was fully automatic
  - low - automatic extraction of microdata was performed and rest of rules were specified manually
  - medium - all rules were created by manual annotation
  - high - all rules were created by manual annotation also also addition refinement rules had to be used

4.0.3 Integration with product search engine

ZedBot system was designed to be integrated with local product aggregation and search system Napstock[16] to provide highly accurate product information for it. Two systems were integrated using RESTful API (see Figure 9). During development ZedBot was also adapted as a source for gathering product sales information, both from online stores and from physical stores when such information was available. And also for gathering details of physical stores (location, openin hours, steetview image, contact information). Extracted data is imported to product search engine using its import API. Using single schema for extraction and by making it compatible with this import API, allowed data from crawler to be directly loaded into Napstock system which aggregates information from different web pages together.

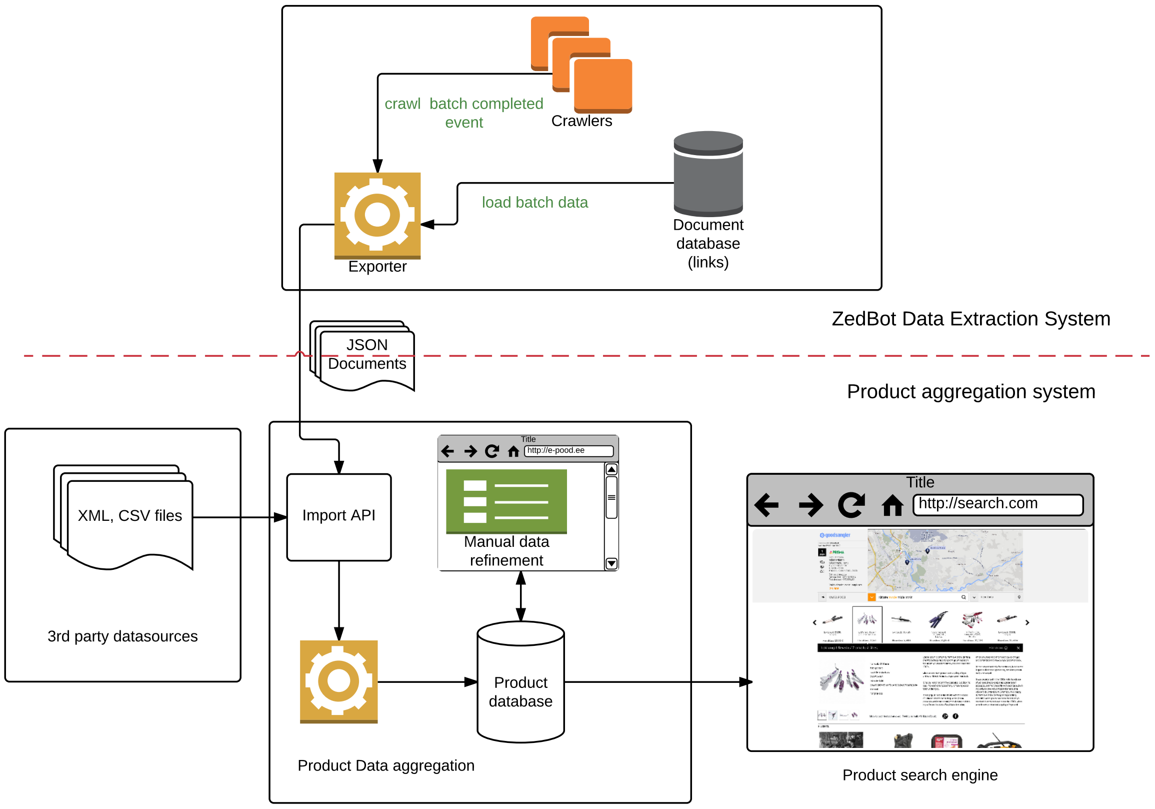


Figure 20: Integration of ZedBot and Product search engine Napstock.

[16] http://napstock.com

5 Conclusions and future work

In this thesis we studied web data extraction problem. There are huge amount of research done in this area from tools that allow human expert to effectively write data extraction program called Wrapper up to fully automatic systems that do not need human supervision. We presented semi automatic information extraction system ZedBot which consist of web annotation combined with rule generation tool and custom web crawler. It complies with majority of requirements set for modern data extraction system. It is platform independent, it has powerful and semi automatic wrapper generation system. Easy to use user interface for annotating structured data. Specially designed web crawler allows to extraction to be performed on whole web site level without human interaction. We showed that presented tool is suitable for extraction highly accurate data from large number of websites and can be used as a data source for product aggregation system with search engine.

Annotating large amount websites and crawling huge amount of pages created valuable dataset, that can be used to develop supervised learning algorithms that make annotation more automatic by classifying elements on web pages. Or it can be used as validation dataset for unsupervised data extraction algorithms.

Although presented system fulfils its task very well, there are many improvements, that can be made to improve its performance. This includes better user interface for annotation and manual wrapper editing. By adding heuristics to rule generation we could avoid manual refinement of complex rules and therefore speeding up wrapper creation process.

Until Semantic Web standards are not widely taken into use by web developers and software designers, need for such information extraction systems remains actual.

Presented software is made publicly available on GitHub[17] under GNU Affero General Public License.

[17]https://github.com/zedbot

References

[BEG$^+$05] Robert Baumgartner, Thomas Eiter, Georg Gottlob, Marcus Herzog, and Christoph Koch. Information Extraction for the Semantic Web. In Norbert Eisinger and Jan Małuszyński, editors, *Reasoning Web*, number 3564 in Lecture Notes in Computer Science, pages 275–289. Springer Berlin Heidelberg, 2005. DOI: 10.1007/11526988_8.

[BFG01a] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Declarative Information Extraction, Web Crawling, and Recursive Wrapping with Lixto. In Thomas Eiter, Wolfgang Faber, and Miros law Truszczyński, editors, *Logic Programming and Nonmotonic Reasoning*, number 2173 in Lecture Notes in Computer Science, pages 21–41. Springer Berlin Heidelberg, September 2001. DOI: 10.1007/3-540-45402-0_2.

[BFG01b] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. The Elog Web Extraction Language. In Robert Nieuwenhuis and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, number 2250 in Lecture Notes in Computer Science, pages 548–560. Springer Berlin Heidelberg, December 2001. DOI: 10.1007/3-540-45653-8_38.

[CKGS06] Chia-Hui Chang, Mohammed Kayed, Moheb Ramzy Girgis, and Khaled F. Shaalan. A Survey of Web Information Extraction Systems. *IEEE Trans. on Knowl. and Data Eng.*, 18(10):1411–1428, October 2006.

[CMM05] Valter Crescenzi, Paolo Merialdo, and Paolo Missier. Clustering Web Pages Based on Their Structure. *Data Knowl. Eng.*, 54(3):279–299, September 2005.

[CYL$^+$08] Rui Cai, Jiang-Ming Yang, Wei Lai, Yida Wang, and Lei Zhang. iRobot: An Intelligent Crawler for Web Forums. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 447–456, New York, NY, USA, 2008. ACM.

[DH99] Jeffrey Dean and Monika R. Henzinger. Finding Related Pages in the World Wide Web. In *Proceedings of the Eighth International Conference on World Wide Web*, WWW '99, pages 1467–1479, New York, NY, USA, 1999. Elsevier North-Holland, Inc.

[FB11] Emilio Ferrara and Robert Baumgartner. Intelligent Self-repairable Web Wrappers. In Roberto Pirrone and Filippo Sorbello, editors, *AI*IA 2011: Artificial Intelligence Around Man and Beyond*, number 6934 in Lecture Notes in Computer Science, pages 274–285. Springer Berlin Heidelberg, September 2011. DOI: 10.1007/978-3-642-23954-0_26.

[FDMFB14] Emilio Ferrara, Pasquale De Meo, Giacomo Fiumara, and Robert Baumgartner. Web data extraction, applications and techniques: A survey. *Knowledge-Based Systems*, 70:301–323, November 2014.

[FGG$^+$12] Tim Furche, Georg Gottlob, Giovanni Grasso, Giorgio Orsi, Christian Schallhart, and Cheng Wang. AMBER: Automatic Supervision for

Multi-Attribute Extraction. *arXiv:1210.5984 [cs]*, October 2012. arXiv: 1210.5984.

[GC14] Tomas Grigalis and Antanas Cenys. Using XPaths of inbound links to cluster template-generated web pages. *Computer Science and Information Systems*, 11(1):111–131, 2014.

[GLSRN00] Paulo B. Golgher, Alberto H. F. Laender, Altigran S. da Silva, and Berthier Ribeiro-Neto. An Example-Based Environment for Wrapper Generation. In Stephen W. Liddle, Heinrich C. Mayr, and Bernhard Thalheim, editors, *Conceptual Modeling for E-Business and the Web*, number 1921 in Lecture Notes in Computer Science, pages 152–164. Springer Berlin Heidelberg, October 2000. DOI: 10.1007/3-540-45394-6_14.

[GRI14] Tomas GRIGALIS. *STRUCTURED DATA EXTRACTION FROM TEMPLATE-GENERATED WEB PAGES*. DOCTORAL DISSERTATION, VILNIUS GEDIMINAS TECHNICAL UNIVERSITY, Vilnius, 2014.

[KB00] Raymond Kosala and Hendrik Blockeel. Web Mining Research: A Survey. *SIGKDD Explor. Newsl.*, 2(1):1–15, June 2000.

[KFT14] K. Kanaoka, Y. Fujii, and M. Toyama. Ducky: A data extraction system for various structured web documents. In *ACM International Conference Proceeding Series*, pages 342–347, 2014.

[KT02] Stefan Kuhlins and Ross Tredwell. Toolkits for Generating Wrappers. In Mehmet Aksit, Mira Mezini, and Rainer Unland, editors, *Objects, Components, Architectures, Services, and Applications for a Networked World*, number 2591 in Lecture Notes in Computer Science, pages 184–198. Springer Berlin Heidelberg, October 2002. DOI: 10.1007/3-540-36557-5_15.

[KT14] Kei Kanaoka and Motomichi Toyama. Effective Web Data Extraction with Ducky. In *Proceedings of the 19th International Database Engineering &#38; Applications Symposium*, IDEAS '15, pages 212–213, New York, NY, USA, 2014. ACM.

[Kus97] Nicholas Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, University of Washington, 1997. AAI9819266.

[Kus00] Nicholas Kushmerick. Wrapper verification. *World Wide Web*, 3(2):79–94, October 2000.

[Liu11] Bing Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2011.

[LMJ04] Hongyu Liu, Evangelos Milios, and Jeannette Janssen. Probabilistic Models for Focused Web Crawling. In *Proceedings of the 6th Annual ACM International Workshop on Web Information and Data Management*, WIDM '04, pages 16–22, New York, NY, USA, 2004. ACM.

[LPH00] L. Liu, C. Pu, and W. Han. XWRAP: an XML-enabled wrapper construction system for Web information sources. In *16th International Conference on Data Engineering, 2000. Proceedings*, pages 611–621, 2000.

[LRNdST02] Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva, and Juliana S. Teixeira. A Brief Survey of Web Data Extraction Tools. *SIGMOD Rec.*, 31(2):84–93, June 2002.

[LYHL10] Cindy Xide Lin, Yintao Yu, Jiawei Han, and Bing Liu. Hierarchical Web-Page Clustering via In-Page and Cross-Page Link Structures. In M. J. Zaki, J. X. Yu, B. Ravindran, and V. Pudi, editors, *Advances in Knowledge Discovery and Data Mining, Pt Ii, Proceedings*, volume 6119, pages 222–229. Springer-Verlag Berlin, Berlin, 2010. WOS:000281629400022.

[O'R06] Sean O'Reilly. *Nominative Fair Use and Internet Aggregators: Copyright and Trademark Challenges Posed by Bots, Web Crawlers and Screen-Scraping Technologies [article]*. Number 3. 2006. TY: GEN; ID: Accession Number: hein.journals.lyclr19.20; Item Citaton: Loyola Consumer Law Review, Vol. 19, Issue 3 (2007), pp. 273-288, O'Reilly, Sean, 19 Loy. Consumer L. Rev. 273 (2006-2007); Accession Number: hein.journals.lyclr19.20; Publication Type: periodical; Source: Loyola Consumer Law Review; Language: English; Publication Date: 20060101.

[PPPD08] K. Pol, N. Patil, S. Patankar, and C. Das. A Survey on Web Content Mining and Extraction of Structured and Semistructured Data. In *First International Conference on Emerging Trends in Engineering and Technology, 2008. ICETET '08*, pages 543–546, July 2008.

[SC13] H. A. Sleiman and R. Corchuelo. A Survey on Region Extractors from Web Documents. *IEEE Transactions on Knowledge and Data Engineering*, 25(9):1960–1981, September 2013.

[SFG+11a] Andrew Sellers, Tim Furche, Georg Gottlob, Giovanni Grasso, and Christian Schallhart. Taking the OXPath Down the Deep Web. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 542–545, New York, NY, USA, 2011. ACM.

[SFG+11b] Andrew Jon Sellers, Tim Furche, Georg Gottlob, Giovanni Grasso, and Christian Schallhart. OXPath: Little Language, Little Memory, Great Value. In *Proceedings of the 20th International Conference Companion on World Wide Web*, WWW '11, pages 261–264, New York, NY, USA, 2011. ACM.

[SWL+13] Shengsheng Shi, Wu Wei, Yulong Liu, Haitao Wang, Lei Luo, Chunfeng Yuan, and Yihua Huang. NEXIR: A Novel Web Extraction Rule Language toward a Three-Stage Web Data Extraction Model. In Xuemin Lin, Yannis Manolopoulos, Divesh Srivastava, and Guangyan Huang, editors, *Web Information Systems Engineering – WISE 2013*, volume 8180 of *Lecture Notes in Computer Science*, pages 29–42. Springer Berlin Heidelberg, 2013.

[TVE14] Maarten Truyens and Patrick Van Eecke. Legal aspects of text mining. *Computer Law & Security Review: The International Journal of Technology Law and Practice*, 2014.

[Vel13] Juan D. Velásquez. Web mining and privacy concerns: Some important legal issues to be consider before applying any data and information extraction technique in web-based environments. *Expert Systems with Applications*, 40(13):5228–5239, October 2013.

[WPC$^+$15] Tim Weninger, Rodrigo Palacios, Valter Crescenzi, Thomas Gottron, and Paolo Merialdo. Web Content Extraction - a Meta-Analysis of its Past and Thoughts on its Future. *arXiv:1508.04066 [cs]*, August 2015. arXiv: 1508.04066.

[YG14] T Yang and A Gerasoulis. Web Search Engines: Practice and Experience. In *Computer Science Handbook*. Chapman & Hall/CRC Press, 3rd edition, 2014.