ISBN 978-981-11-0008-6
Proceedings of 2016 6th International Workshop on Computer Science and Engineering (WCSE 2016)
Tokyo, Japan, 17-19 June, 2016, pp. 167-179
doi: 10.18178/wcse.2016.06.028

A Web Application for the Collection and Comparison of E-Shop Product Data

Andrea Horch [+], Holger Kett and Anette Weisbecker

Fraunhofer Institute for Industrial Engineering IAO, 70569 Stuttgart, Germany

Abstract. E-commerce is a highly competitive and constantly growing market. On-line retailers need to compare their product offers with those of their competitors in order to remain competitive. Due to the large range of product offers on the Web and the fact that internet prices are adjusted on a daily basis or even more frequently the manual comparison of on-line product offers is a very time consuming task. Therefore, on-line retailers prefer the use of automated tools for performing that work. There are many existing ready-to-use software tools and on-line services from price comparison websites to market intelligence tools, but those tools still require a greater effort for the configuration of the tool or the comparison of the offers. The contribution of this paper is a Web application to automatically identify, extract and compare the product offers of the own e-shop with those of defined competitors. The specialties of the presented Web application are its degree of automation and the little effort for configuration.

Keywords: web mining, web data extraction, e-commerce.

1. Introduction

Europe's e-commerce is a competitive and constantly growing market. It's turnover in 2014 raised by 16.3% to 363.1 billion Euros. There are 264 million e-shoppers and an estimated amount of 645,000 on-line retailers in Europe [1]. As determined in [2] it is very important for e-shoppers to find the lowest price on-line. Thus, knowing the prices of the competitors and adjusting the own prices is crucial for on-line retailers in order to remain successful.

Since on-line prices are updated daily or even more often as depicted in [3] and in consideration of the large number of European e-shops, which offer a vast array of different products, to monitor the competitors' prices manually is a hard and time consuming task. Hence, on-line retailers need the support of software tools to be able to manage this challenge. There are several existing software tools and on-line services as price comparison websites or market intelligence suites for the monitoring and analysis of competitors' prices as well as for the comparison of prices, but these tools still need much effort for their configuration or to perform the product and price comparison. In this paper we introduce a novel Web application for the collection, monitoring and comparison of the own products and prices as well as those of defined competitors. The presented application uses a fully-automated approach to perform all necessary tasks from product identification within the on-line shops to product attribute extraction and the comparison of product offers.

2. System Requirements

In the following subsections the non-functional as well as the functional requirements for the Web application for the collection and comparison of on-line product offers are derived.

[+] Corresponding author. Tel.: + 49 (0) 711 970 2349; fax: + 49 (0) 711 970 5111.
E-mail address: andrea.horch@iao.fraunhofer.de.

2.1. Non-Functional Requirements

In consideration of the current market situation of the European e-commerce with regard to the large number of e-shoppers and on-line retailer in Europe, the existing cross-border shopping and the diversity of the most popular product categories the following non-functional requirements can be derived:

1. Since there is an estimated amount of 645,000 on-line retailers on the European market as depicted in [1] and on-line prices are updated daily or even more frequent [3] the system needs to be completely automated.
2. As shown in [2] the top-selling product categories are very diverse, therefore the application should be product domain independent.
3. Considering the huge amount of e-shops in Europe [1] the system needs to be source independent in order to be able to collect and analyse product offers of arbitrary e-shop websites.

2.2. Functional Requirements

To be able to automatically identify, extract and compare the product offers of arbitrary e-shop websites the system needs to offer the following functions:

1. Product page detection: Detection of all Web pages within an e-shop website containing product records.
2. Product record detection: Identification and extraction of all product records within a Web page.
3. Product attribute extraction and assignment: Identification and extraction of product attributes as e.g. product name, product image or product price within a product record and the assignment of the extracted attribute value to a defined product property for further processing.
4. Product resolution: Matching of identical and similar product offers referring the same real-world product.

3. Related Work

There are various existing software tools and services for the comparison of prices and products of e-shops on the market. Additionally, there exist on-line market places to master the whole task or to handle specific subtasks. Furthermore, there are several research approaches dealing with that challenge. These tools, services and scientific approaches will be described in the following paragraphs.

3.1. Comparison Shopping Services

Comparison shopping services offer easy-to-use interfaces to search for products and prices. Some of the most popular comparison shopping engines are Google Shopping [1], Shopzilla [2] and PriceGrabber [3] [4]. Comparison shopping services are Web-based applications which collect and aggregate on-line offers for a specific product from different e-shop websites. Thus, users can search for the on-line offers for a specific product quoted by various e-shops by using a comparison shopping service instead of searching for the offers on each single e-shop website [5]. Additionally, those services support users through ranking the result list of product offers according to specific attributes the users can select as e.g. price or popularity. There are two different technologies comparison shopping services use for the collection of product offer data [5]:

1. Data wrapping: The data is collected through data wrappers which are software programs to collect, to structure and to store Web data in an automated manner.
2. Data feeding: Vendors directly feed their product and price data into the database of the comparison shopping engine, e.g. through a Web form or an Application Programming Interface (API).

Detriments of the use of comparison shopping services for comparing a larger set of product offers for commercial purposes are (1) the limitation of sources of the price comparison services as the services only

[1] http://www.google.com/shopping
[2] http://www.shopzilla.com/
[3] http://www.pricegrabber.com/

include offers of a limited set of e-shops and (2) the effort to perform searches for each product as well as for the extraction and further processing of the result data.

3.2. Web data Extraction Services

Web data extraction services as import.io[4] or kimono[5] are Web-based software applications to extract data directly from websites. Those tools usually offer an easy-to-use Web interface like a Web browser plug-in in order to define the information to be extracted directly within the website e.g. through marking it in the browser. The resulting data is typically offered in the form of a comma-separated values (CSV) sheet or a JavaScript Object Notation (JSON) file which can be downloaded after the automated data extraction. The major advantage of such Web-based services is the ability to extract actual data directly from the Web without the need of programming skills. The main disadvantage is the huge effort to configure the service for each website.

3.3. Price and Market Intelligence Tools

Price and market intelligence tools and services offer a large set of features as e.g. price monitoring and comparison or graphical and tabular data reports. Those tools include a huge set of connected e-shops and e-shop data within their databases. Additionally, the providers of those services promise a fast integration of new required e-shops. Most price and market intelligence services are created in the form of a cloud solution, thus they are ready-to-use and do not need to be integrated into the own information technology (IT) infrastructure, but they need to be connected to the own e-commerce platform, pricing system or Enterprise resource planning (ERP) system. Examples for such tools are price2spy[6] and Prisync[7]. Price Intelligence is a cloud service which offers a completely automated matching of products, but it needs to be integrated into the own e-commerce platform or pricing system. Prisync does not require any integration but it needs the configuration of the product URL for each competitor and for each product. The service price2spy can automatically match products based on a unique identifier (Automatch), for products not having a unique identifier the matching needs to be done manually by the user.

3.4. Product and Price APIs

Product and price APIs like priceAPI.com[8] and Semantics3[9] offer real-time data of products, prices and competitors through an API. The data comes from comparison shopping websites, on-line market places and directly from merchants. Products and competitors can be searched by several attributes as category or branch. The users can integrate the data by connecting their systems to the API. Hence, the use of such APIs requires some programming skills as well as some effort for the programming.

3.5. Scientific Approaches

The Lixto research project which initially started research in developing a logic-based extraction language and a tool for visually generating software programs to extract data from websites (Wrappers). That work is described in [6]-[8]. Currently, the scope of the Lixto project has been extended to additional research in the field of fully automated and unsupervised Web data extraction [9]. The supervised work of the Lixto project focuses on data extraction from deep Web pages which includes the challenge of form filling and the navigation on Web pages. The unsupervised research focuses on the automated detection of websites including relevant information about a specific subject, automated navigation among the links on the identified websites and the extraction of data of the most important pages within these pages which are presented within tables. A major advantage of the supervised approach of the Lixto project is the possibility to collect data behind Web forms (deep Web data) which is necessary to access service offer data like e.g. rental car offers or flight offers. However, the major disadvantage of that approach is the huge effort for the configuration of the tool for each website which shall be tapped. Other detriments are the need to clean and

[4] https://import.io/
[5] https://www.kimonolabs.com/
[6] http://www.price2spy.com/en/home.html
[7] http://www.prisync.com/
[8] https://www.priceapi.com/
[9] https://www.semantics3.com/

integrate the collected data as well as the necessity of entity resolution to be able to e.g. compare product prices. The unsupervised approach of the Lixto project is only able to extract data from table structures and it is just adequate for use cases where a precise data extraction is not required. Additionally, that approach does not support the assignment of the extracted attribute to a specific meaning as e.g. a particular product attribute. Moreover, it does not provide an entity resolution, which is required for a product and price comparison.

There are also scientific approaches dealing with a subtask of e-commerce data mining and analyses as on-line product data extraction or product resolution. Research approaches for the extraction and structuring of product descriptions from e-shop websites can be found in [10]-[19].

4. Approach

The approach was derived from the requirements presented in Section 2. Figure 1 shows the process of data collection, data processing and data analysis and the intermediate results of each process step as well as the required input and the resulting output of the approach. The process is separated into two parts: (1) product data extraction and (2) product resolution.

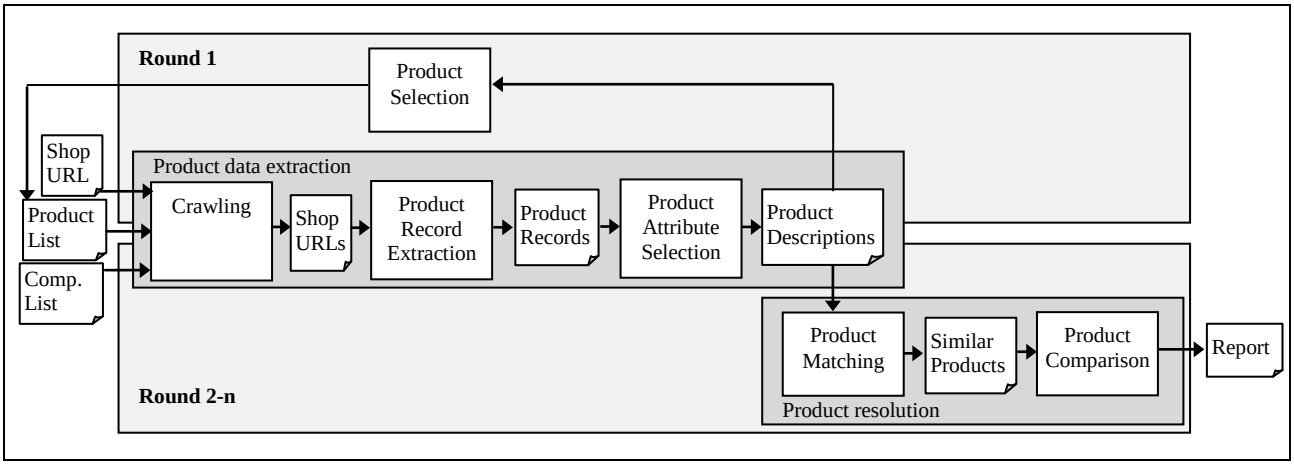

Fig. 1: Process of product and price data extraction and product resolution.

The first round of the process differs from the following rounds as in the first round required input data for the subsequent rounds are collected. The necessary input for round 1 is an arbitrary URI of the e-shop, which products shall be compared with the competitors' products. In that round the approach takes the URI as input and detects the page domain within the URI in order to identify the start page of the website. After the detection of the start page the e-shop website is crawled until its 3rd level in order to collect all links which are available until level 3 of the website. Level 0 is the homepage of the website, level 1 pages are pages which can be directly accessed from level 0, level 2 pages are pages which can be accessed by following two links from homepage and level 3 pages are pages three links away from homepage. The e-shop websites are crawled until level 3 to follow the results of an analysis of 50 different e-shop websites. These results have shown that 76% of the analysed websites had product lists on level 0, 96% presented product lists on level 1, 74% included lists of products on level 2 and still 12% had product lists on level 3. A more detailed description of our analysis is presented in [20]. After the crawling each Web page referenced by the collected URIs of the e-shop website is visited and checked for the occurrence of product records. Each identified product record is extracted and further processed by searching for the following product attributes within the product records: (1) product name, (2) product image, (3) link to the product image, (4) link to the detail page of the product, (5) actual product price, (6) regular product price, (7) currency of prices and (8) product units as weight or volume information (e.g. 100 g or 50 ml). The results of that step are presented to the users. The users can select the products from the list of the collected products for which they want to monitor the own prices and the competitors' prices. Round 1 is completed after the product selection.

The further rounds of the approach require the following input: (1) the e-shop URI (same as for round 1), (2) the list of selected products from round 1 and (3) the URIs of the competitors to compare the products. The crawling step will not be performed each time as it is sufficient to check an e-shop website several times a month for the occurrence of new links (but not every day). Thus, in the most cases the process takes the already collected URIs and checks each page for the occurrence of product records. Each of the identified product records is collected and the product attributes are extracted. These steps are performed for the original e-shop website as well as for the e-shop websites of the defined competitors. After the extraction and structuring of the product information of these websites the products of the original e-shop are matched with products of the competitors' e-shop websites for all product descriptions having a high degree of similarity as these product descriptions are expected to represent identical or similar products. In the last step the identical and similar products are compared e.g. with regard to their price. The results of the comparison are presented to the users.

5. Application

The approach described in Section 4 was implemented in a Web application. The design of the application is presented in Fig. 2. The system consists of four main components which partially contain further modules, two databases and a RESTful application programming interface (API). The application was implemented in Python. The database containing the data defined by the users and the clean product data as well as the API are located on a Virtual Machine separated from the data collection modules. Hence, the first Virtual Machine holds the front-end, the second one the back-end of the system. Both machines are Windows 8 systems.

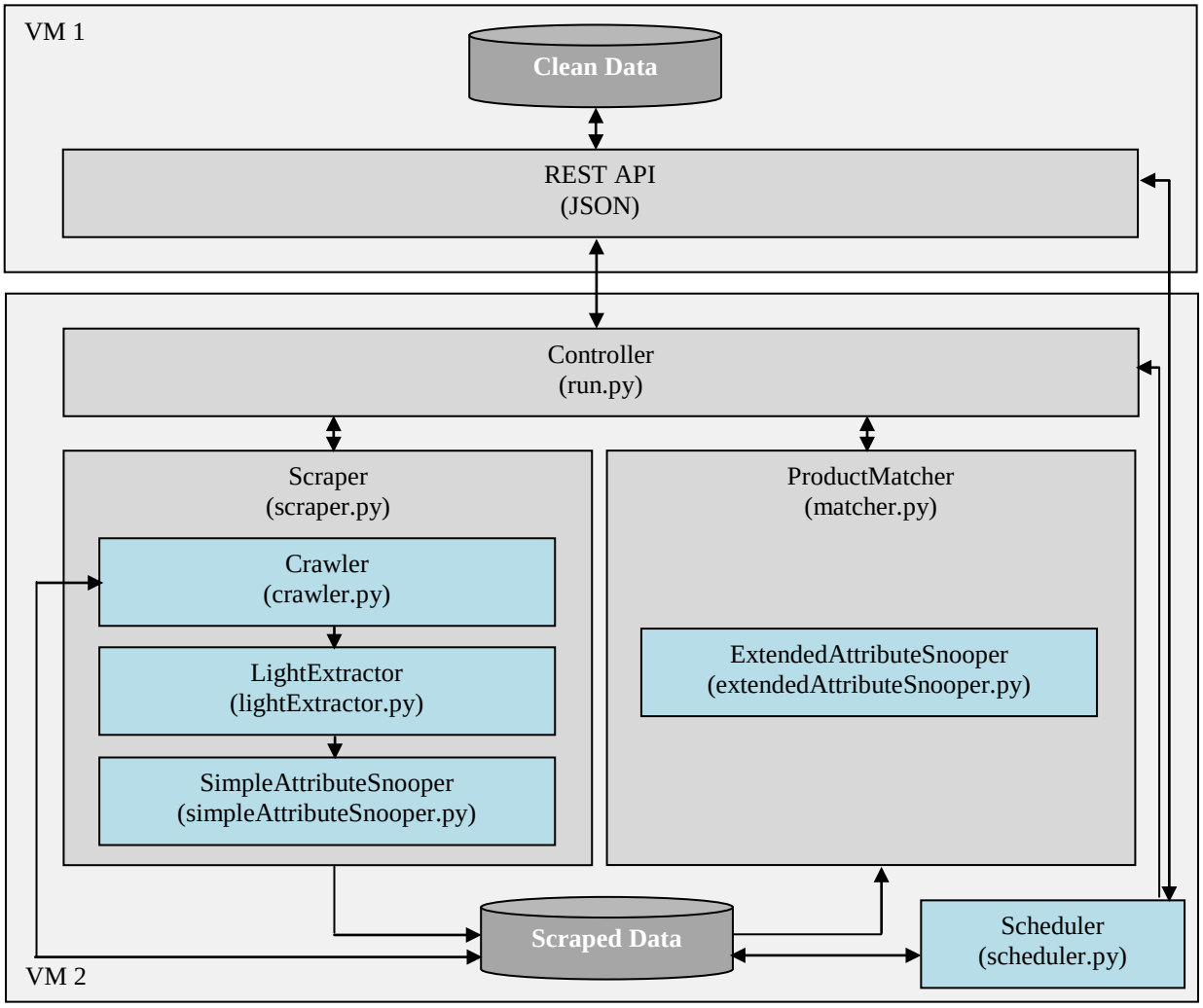

Fig. 2: Design of the web application.

5.1. Application Programming Interface (API)

The Application Programming Interface (API) consists of a RESTful service which is connected to a MySQL database containing the user data and clean product data. The interface is running on an Apache HTTP Server to enable the communication with the user or an external system. The RESTful service was built on the Flask[10] micro framework for Python. The RESTful service provides different operations to perform the following tasks: (1) to register an e-shop (create account), (2) to create, update and delete competitors, (3) to create, update and delete data collection jobs, (4) to retrieve shop, competitor and product information and (5) to retrieve collected product data. The data is provided in JavaScript Object Notation (JSON).

5.2. Scheduler

The frequency for the data collection is defined by the user for each data collection job. A job is executed for the first time directly after its creation. The *Scheduler* checks each job for its last run and its frequency for the data collection and executes it on the day of its calculated next run. The *Scheduler* ensures that only a defined number of jobs are executed at the same time. The initial information of a job is stored in the clean database and is sent to the back-end where the information to schedule each job is calculated and stored by the *Scheduler*. The *Scheduler* itself is controlled and started by a Windows task.

5.3. Controller

The *Controller* operates the data collection jobs started by the *Scheduler*. For this purpose it takes the input for a job from the *Scheduler*, collects all further information from the *API* and controls the execution sequence of the *Scraper* and the *ProductResolver* as well as the data transfer between the modules.

5.4. Scraper

The *Scraper* is one of the core modules of the application. It is responsible for the data collection and the pre-processing of the product information. To perform those tasks the *Scraper* controls and executes the following sub-modules in a sequential order: (1) *Crawler*, (2) *LightExtractor* and (3) *SimpleAttributeSnooper*.

5.4.1. Crawler

The *Crawler* takes the URI of the e-shop and searches the protocol type, scheme and domain by using the python library urlparse in order to find the start page of the e-shop website. Subsequently, it goes to the identified start page and extracts all links including the page domain. Then it follows the identified links and extracts all links on each visited Web page until level 3 as described in Section 4. The system uses the Python module urllib2 to retrieve the Web pages and the Python library BeautifulSoup[11] to detect and extract the links from each page.

5.4.2. LightExtractor

The *LightExtractor* module identifies and extracts the product records within the Web pages of the e-shop website. It uses a clustering technique based on the element paths of the HTML elements of the page as well as a special filter to find the product records within the Web pages. The algorithm is shown in Fig. 3.

The Algorithm renders the Web page and adds all styles from external style files. The external styles are necessary for the next step to identify the product attributes (see Section 5.4.3). The algorithm runs through the elements of the HTML tree of the page whereat it ignores style and script elements (line 1 and 2). All other elements are checked by a filter (see line 6) to contain the following elements: (1) at least three child nodes, (2) some text, (3) an image element and (4) an anchor tag. Elements including all those contents are potential product records. For all elements identified as potential product records the element path is created as presented in Fig. 4 (see line 7). The elements are stored to a dictionary which indices are the tag path of the corresponding elements in order to cluster elements having the same tag path together (see line 8). The system assumes that the cluster which contains the largest number of elements includes the product records (see line 10 and 12). A more detailed description of *LightExtractor* is given in [20]-[21].

[10] http://flask.pocoo.org/
[11] http://www.crummy.com/software/BeautifulSoup/

```
render Web page
(add external styles)

for each element in HTML page tree:
  ignore style & script elements
  if product record filter matches:
    generate tag path
    add element to tag path cluster

get tag path of cluster with max. elements

results = elements with identified tag path

return results
```

Fig. 3: LightExtractor algorithm.

```
if element is not a StyleSheetElement:
   for parent in element.findParentNodes:
      if parent is not a StyleElement:
         element_path =  "/" + parent.tagName + element_path
      try:
         element_path += "/" + element.tagName + "["
                 + element.classAttribute + "]"
      except:
         element_path += "/" + element.tagName
```

Fig. 4: Tag path creation for element clustering.

The implementation of *LightExtractor* for the described system uses the headless browser mechanize[12] to retrieve Web pages which do not use Asynchronous JavaScript and XML (AJAX), for Web pages using AJAX it utilizes Selenium WebDriver[13]. In order to be able to use a consistent method for the element analysis the module uses BeautifulSoup to parse the HTML tree of the pages instead of the mechanisms implemented in mechanize and Selenium WebDriver.

5.4.3. SimpleAttributeSnooper

The *SimpleAttributeSnooper* identifies and extracts the following product attributes from the product records which are extracted by *LightExtractor*: (1) product name, (2) product image, (3) product prices, (4) currency of prices and (5) link to the detail page of the product. The approach of the *SimpleAttributeSnooper* is presented in detail in [20], but we will give a short description of its function here, too. The approach assumes that the product image is included in the image element within the product record with the largest file size. Therefore, the image with the largest file size is extracted as product image. Prices are identified by a regular expression. Our analysis of fifty e-shops of 6 different countries and various product domains has shown that a product record can contain up to 5 different price types. *SimpleAttributeSnooper* identifies the actual product record by assuming that the price with the largest font size is the actual product price whereas the regular product price is expected to be crossed-out. The style information to find the largest font size and crossed-out elements is taken from the Cascading Style Sheets (CSS) description of the HTML elements. The currency is also extracted by using a regular expression. Currently, *SimpleAttributeSnooper* is able to detect the following currencies: Great Britain Pounds (GBP), Euros (EUR) and United States Dollars (USD). The product name is assumed to be included in the link which is the closest to the product image. The product name is expected to be found within the *alt* attribute or *title* attribute of the product image or the *title* attribute of the anchor element including the text with the largest font size (identified by the CSS

[12] https://pypi.python.org/pypi/mechanize/
[13] http://selenium-python.readthedocs.org/

information). If this anchor element does not contain a *title* attribute the text of the element is assumed to be the product name. *SimpleAttributeSnooper* uses BeautifulSoup for parsing the HTML tree of the product record elements and CSS21Parser[14] is utilised to embed the external CSS of a Web page. The CSS need to be embedded in the *LightExtractor* module before the product records are extracted in order to be able to properly allocate the CSS to the corresponding HTML elements.

5.5. ProductMatcher and ExtendedAttributeSnooper

Another main module of the application is called *ProductMatcher*. The main function of this module is the matching of the products of the original e-shop with those of its competitors, which is also called product resolution. Product resolution is a special variation of entity resolution and stands for the identification of product offers which represent the same real-world product [16]. For this step the product are compared based on their names (product titles) and units (e.g. weight or volume information).

The comparison and eventual matching of the product names is based on a pairwise comparison of the names of the products to compare. For this puspose the ProductMatcher calculates the string similarity of all pairs of product names of the original e-shop and its competitor. The similarity of string pairs is determined by the following similarity values.

1. Intersection of words: This similarity value is the ratio between the amount of common words of the strings to compare and the number of words of the longest string.
2. Maximum 3-gram similarity: The maximum 3-gram similarity is the maximum value of all string similarities calculated for all 3-grams of the product names to compare. The similarity of the 3-grams is calculated by using the ratio method of difflib.SequenceMatcher[15].
3. Average similarity: The average similarity is the average value of the the intersection of words and the maximum 3-gram similarity.

The described similarity values were created, selected and validated through an experiment on real-world data presented in [22]. The matching algorithm pre-processes the name strings of all products by removing all charactes which are no letters (also Greek or Spanish letter), numbers or underscores and converts the resulting strings to lower case. Following it calculates the introduced similarity values for each pair of product names. The product term is the product name of the original e-shop as defined by the user through selecting the product name during the first run of the application. As the product name of the original product may be changed since the first run the application needs to find it within the collected dataset of product descriptions of the original e-shop just like within the extracted dataset of the competitors. Therefore, the matching algorithm first compares the product term to all product names of the collected product records of the original shop in order to find the actual product description of the original e-shop, and then it compares the product name of the identified actual product record of the original e-shop to the product names of all collected product records of the competitors. The comparison of product names is performed by calculating the similarity values for all pairs of names. The product names with the highest average similarity are assumed to be matches. As there are also products which may not have a match within the dataset of product records to compare there is a need to define a threshold which has to be exceeded by the similarity values of the product name pairs to be a match. The threshold has to be exceeded either by the intersection of words or by the maximum 3-gram similarity value. The threshold can be determined by calculating the similarity values for a small subset of the real data and by identifying the smallest theshold finding all or a huge amount of matches. In [22] we identified the thresholds for the real-world data of two e-shops of the product domains *clothing* and *books*, the smallest threshold of these domains was 0.68 for the domain *books*. If for a product name there is not another product name with a similarity value for the intersection of words or for the maximum 3-gram similarity greater than the threshold value the original product name does not have a match within the product names to compare. Otherwise, if there are product names within the product records to compare which have a similarity value (intersection of word or maximum 3-gram similarity)

[14] http://pythonhosted.org/tinycss/extending.html
[15] https://docs.python.org/2/library/difflib.html

greater than the threshold, the product record with the maximum average similarity is assumed to be a match for the original product.

As there can be several products having the same average similarity value for the original product to be able to find the best match there is a need to check and compare further product attributes. For this purpose the proposed matching algorithm additionally compares the units such as size and volume information of the product descriptions which are assumed to be matches by the comparison of product names in order to find the best match. For the comparison of product units we use the approach we have described in [23]. The unit extraction is perfomed by the module *ExtendedAttributeSnooper*. The approach is based on a unit ontology we have manually created for the idenfication, extraction and unification of product units as size, volume or weith units. The design of the ontology is shown in Fig. 5.

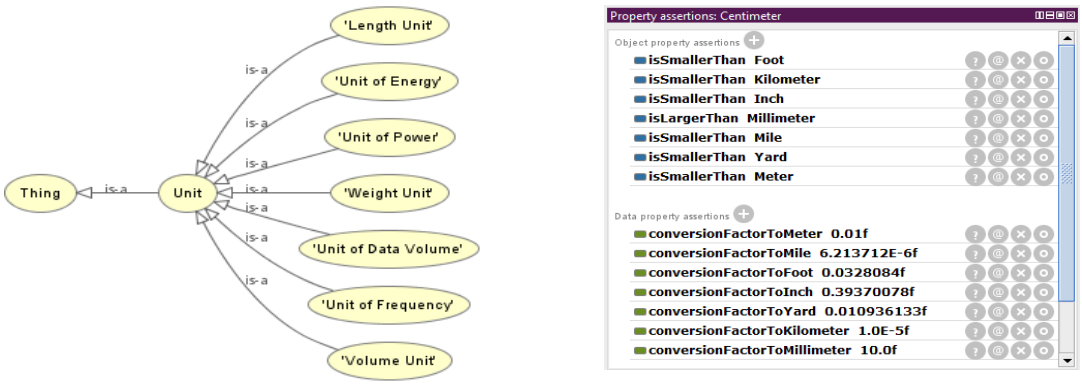

Fig. 5: Unit ontology for unit data extraction.

The unit ontology provides the following information to identify and extract the units and the corresponding values:

1. Hierachical information of units, e.g. mililitres are smaller than litres.
2. Common unit abbreviations, e.g. *mL* for mililires or *km* of kilimetres.
3. Regular expressions in order to find and extract units, e.g. `'\d+(,|\.)?\d*\s?(in|"|inch|Inch|Zoll)(\s|$|\))'` to find and extract inch units and values.
4. Conversion factors of units to convert the value of a unit into another unit.

The regular expressions stored for each unit in the unit ontology are selected by a SPARQL query[16] and used to identify and extract units and the corresponding values within the product names and product short descriptions of the product records which probaly are matches and which were identified by the product name comparison as well as within those of the original product (identified product of the original e-shop) and the product term (selected product name at the first run). All extracted unit values and unit names are converted into the smallest unit defined within the unit class of the onology by using the conversion factor and unit abbreviations provided by the ontology in order to make them comparable (e.g. *1 litres* is converted to the key-value pair *{'1000':'mL'}*). The matching algorithm counts the common units of all pairs of product records to compare and the product record with the maximum number of intersecting units is considered as the best match. For original product which do not have any product having matching units within the data records of products which are matches according to the product name comparison the product record including the most similar units is considered to be the best match. For this purpose all values of the same units of the product pair are compared and the similarity in percent for each unit value pair is calculated. Afterwards the average unit similarity is determined by calculating the average of the similarities of all unit value pairs. The product having the maximum average similarity is considered to be the best match.

6. Evaluation

[16] https://www.w3.org/TR/rdf-sparql-query/

The approach was evaluated by performing an experiment on real-word data. The set-up of the experiment is described in Section 6.1 and the results of the experiment are shown in Section 6.2.

6.1. Experimental Set-up

The approach was evaluated by running the system on real-world data of several e-shops. For this purpose there were selected ten e-shops of five of the ten top-selling product categories identified by [2]. For each product category two e-shops and the corresponding competitor e-shops had been selected by performing the Google search queries shown in Fig. 6 and Fig. 7.

```
1 <<product category>> shop .<<country code>>
2 <<product name>> shop .<<country code>>
```

Fig. 6: Common Google Search Query to identify shops and competitors.

The first line of Fig. 6 shows the structure to find an e-shop of a specific product category. The query comprises the name of the product category the keyword shop to define that an online shop is searched and a dot followed by the code of the country of the searched e-shop. A specific example for a search query to find a Greek e-shop of the product category cosmetics is shown in line 1 of Fig. 7. The first Greek e-shop within the result list of the Google search is taken for the experiment.

```
1 cosmetics haircare shop .gr
2 Tol Velvet Relief Massage Oil 125ml shop .gr
```

Fig. 7: Example Search query for the identification of shops and competitors.

The competitors of the e-shops determined by the Google search of line 1 are searched by another Google search structured as presented in line 2 of Fig. 6. The product name in the second Google search is the name of a product found within the e-shop identified by the first Google search. This product is randomly chosen by browsing the e-shop website. The example in line 2 of Fig. 7 shows a search query to find a competitor for the e-shop identified by running the search query in line 1. The search competitor must include a product with the name “Tol Velvet Relief Massage Oil” and shall be a Greek e-shop. The first e-shop which includes the searched product and has the same language as the e-shop from the associated first Google search is selected from the result list. Table 1 shows the resulting e-shops and corresponding competitors for the experiment.

Table 1: Table of Resulting e-Shops and Competitors (Experimental Dataset)

No.	Shop	Competitor	Product category
1	http://www.e-bookshop.gr/	http://books.gr/	Books
2	http://www.whsmith.co.uk/	http://www.easons.com/	Books
3	http://www.hair-care24.de/	http://www.hair-express.de/	Cosmetics & haircare
4	http://bestpharmacy.gr/	http://efarmakeio.gr/	Cosmetics & haircare
5	http://www.comtech.de/	http://www.notebooksbilliger.de/	Home electronics
6	http://www.dabs.com/	http://www.acerdirect.co.uk/	Home electronics
7	http://www.blockshop.es/	http://www.baddaclothes.com/	Clothing & footwear
8	https://www.uebervart-shop.de/	http://asphaltgold.de/	Clothing & footwear
9	http://www.bargainmax.co.uk/	http://www.kidsstufftoys.co.uk/	Toys
10	http://www.toyplanet.es/	http://www.juguetilandia.com/	Toys

For the experiment there were selected 10 products per product category which can be found within the original e-shop and the e-shop of the associated competitor. The system need to find all matching products in

the competitor's e-shop. As a result the system shall return all identified products and related matches in the form of structured product descriptions including product name, link to the detail page of the product, product price and currency information.

6.2. Experimental Results

The results were evaluated by calculating the precision and recall values for the achieved results. The precision shows the fraction of correct matches in proportion to incorrect matched products. The recall indicates the fraction of correct matches in proportion to possible matches which could not be identified by the system. The equations to calculate precision and recall are given in Fig. 8.

Precision = True Positives / (True Positives + False Positives)
Recall = True Positives / (True Positives + False Negatives)

Fig. 8: Equations to calculate precision and recall.

The results the system could achieve for the experimental dataset descried above are shown in Fig. 9.

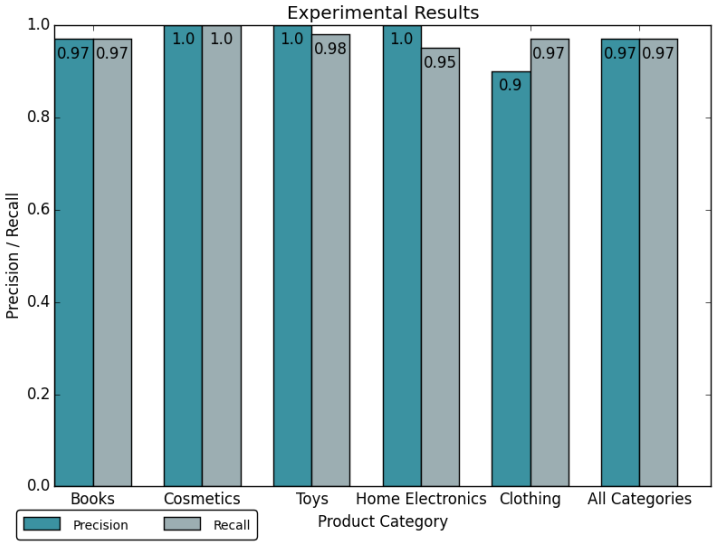

Fig. 9: Experimental results.

Fig. 9 shows the precision and recall per product category as well as the overall value of all categories. The smallest value for precision could be achieved for the category *Clothing* whereas the lowest recall was obtained for the category *Home Electronics*. The lower recall for the category Home Electronics was caused by an inaccuracy of the matching algorithm as the pre-set threshold for the similarity values is too high since for the category *Home Electronics* the string similarity of the product names are not as close as those of the other categories. The smaller value for precision the category Clothing was caused by an inadequate extraction of some product names from the e-shop website. So, some product names of searched matches were not extracted and could not be matched what result in False Positives as other similar product names were found and matched. Apart from some identified False Positives and not recognized False Negatives the system works fine and the results are very satisfactory. Another smaller issue which can be easily improved is the wrong extraction of product prices in the case of prices which are distributed over two tags. Currently, the system can only correctly extract the values of prices which are distributed over sibling tags, but some prices of the experimental dataset were distributed over a tag and its child tag and such prices could not be correctly extracted, but that issue can easily be solved.

7. Conclusion and Future Work

This paper introduces a Web application for an automated identification, extraction and comparison of e-shop websites. First, the system requirements were defined and then an approach for identifying, extracting and matching product descriptions was built based on the requirements. The approach was implemented in a

Web application which was evaluated by an experiment on real-world data. The experimental results were very satisfactory, but some smaller improvements on the algorithm for the attribute extraction have to be made in the next version of the system.

8. References

[1] B. Nagelvoort, R. van Welie, P. van den Brink, A. Weening und J. Abraham, „Europe b2c e-commerce light report 2014," Ecommerce Europe, 2014. [Online]. Available: https://www.adigital.org/media/2014/06/european-b2c-ecommerce-report-2014.pdf. [Accessed on 25/01/2016].

[2] PostNord, „E-commerce in Europe 2014," 2014. [Online]. Available: http://www.postnord.com/globalassets/global/english/document/publications/2014/e-commerce-in-europe-2014.pdf. [Accessed on 25/01/2016].

[3] CIVIC Consulting, „CIVIC PROJECTS," 2013. [Online]. Available: http://www.civic-consulting.de/project_50.html. [Accessed on 25/01/2016].

[4] M. Weinstein, „Search Engine Watch," 14 06 2014. [Online]. Available: https://searchenginewatch.com/sew/study/2097413/shopping-engines. [Accessed on 25/01/2016].

[5] Y. Wan, Comparison-Shopping Services and Agent Designs, University of Houston, USA: Hershey, 2009.

[6] R. Baumgartner, S. Flesca und G. Gottlob, „Visual Web Information Extraction with Lixto," in *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001)*, Rome, Italy, 2001.

[7] R. Baumgartner, S. Flesca und G. Gottlob, „Supervised Wrapper Generation with Lixto," in *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001)*, Rome, Italy, 2001.

[8] R. Baumgartner, G. Gottlob und M. Herzog, „Scalable Web Data Extraction for Online Market Intelligence," in *Proceedings of the VLDB Endowment*, Lyon, France, 2009.

[9] J. Carme, M. Ceresna, O. Frölich, G. Gottlob, T. Hassan, M. Herzog, W. Holzinger und B. Krüpl, „The Lixto Project: Exploring New Frontiers of Web Data Extraction," in *Flexible and Efficient Information Handling*, Berlin Heidelberg, Springer, 2006, pp. 1-15.

[10] B. Liu, R. Grossman und Y. Zhai, „Mining Data Records in Web Pages," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, Washington, DC, USA, 2003.

[11] M. Walther, L. Hähne, D. Schuster und A. Schill, „Locating and Extracting Product Specifications from Producer Websites," in *Enterprise Information Systems*, Funchal, Madeira, Portugal, Springer, 2010, pp. 13-22.

[12] N. Anderson und J. Hong, „Visually Extracting Data Records from the Deep Web," in *Proceedings of the 22nd International World Wide Web Conference (WWW 2013)*, Rio de Janeiro, Brazil, 2013.

[13] T. Grigalis, „Towards Web-scale Structured Web Data Extraction," in *Proceedings of the sixth ACM international conference on Web search and data mining (WSDM 2013)*, Rome, Italy, 2013.

[14] T. Grigalis und A. Čenys, „Unsupervised Structured Data Extraction from Template-generated Web Pages," *Journal of Universal Computer Science (J.UCS)*, Bd. 2, Nr. 1, pp. 169-192, 2014.

[15] L. Yang, M. Ball, V. Bhavsar und H. Boley, „Weighted Partonomy-Taxonomy Trees with Local Similarity Measures for Semantic Buyer-Seller Match-Making," *Journal of Business and Technology*, pp. 42-52, 2005.

[16] A. Thor, „Toward an adaptive String Similarity Measure for Matching Product Offers," in *Informatik 2010: Service Science - Neue Perspektiven für die Informatik, Beiträge der 40. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, Leipzig, GI, 2010, pp. 207-210.

[17] K. Balog, „On the Investigation of Similarity Measures for Product Resolution," in *LHD 2011*, Barcelona, Spain, 2011.

[18] V. Gopalakrishnan, S. P. Iyengar, A. Madaan, R. Rastogi und S. Sengamedu, „Matching Product Titles Using Web-based Enrichment," in *Proceedings of the 21st ACM international conference on Information and knowledge management (CIKM 2012)*, Maui, HI, USA, 2012.

[19] Z. Yetgin und F. Gözükara, „New metrics for clustering of identical products over imperfect data," *Turkish Journal of Electrical Engineering & Computer Sciences*, pp. 1-14, 2014.

[20] A. Horch, H. J. Kett und A. Weisbecker, „Mining E-commerce Data from E-shop Websites,“ in *Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA*, Helsinki, Finland, 2015.

[21] A. Horch, H. J. Kett und A. Weisbecker, „A Lightweight Approach for Extracting Product Records from the Web,“ in *Proceedings of the 11th International Conference on Web Information Systems and Technologies (WEBIST 2015)*, Lisbon, Portugal, 2015.

[22] A. Horch, H. J. Kett und A. Weisbecker, „Matching Product Offers of E-Shops,“ in *Proceedings of the The 20th Pacific Asia Conference on Knowledge Discovery and Data Mining (PAKDD) 2016, LNCS (LNAI)*, Auckland, New Zeland, forthcoming 2016.

[23] A. Horch, H. J. Kett und A. Weisbecker, „Extracting Product Unit Attributes from Product Offers by Using an Ontology,“ in *Proceedings of the Second International Confernce on Computer Science, Computer Engineering, and Social Media (CSCESM)*, Lodz, Poland, 2015.