Faceted Product Search Powered by the Semantic Web

Damir Vandic, Jan-Willem van Dam, Flavius Frasincar

Erasmus University Rotterdam
PO Box 1738, NL-3000 DR
Rotterdam, the Netherlands

Abstract

This paper presents a platform for multifaceted product search using Semantic Web technology. Online shops can use a ping service to submit their RDFa annotated Web pages for processing. The platform is able to process these RDFa annotated (X)HTML pages and aggregate product information coming from different Web stores. We propose solutions for the identification of products and the mapping of the categories in this process. Furthermore, when a loose vocabulary such as the Google RDFa vocabulary is used, the platform deals with the issue of heterogeneous information (e.g., currencies, rating scales, etc.).

Keywords: Semantic Web, SPARQL and RDF, product identification, category mapping, product search

1. Introduction

Online product search, as a tool to help customers find their products of interest, has become more important than ever as consumers nowadays purchase more often on the Web [1]. This is due to the fact that there is an increase in product specificity and consumer preference variation. The most important reason for this is technical advancement, as this has led to a large increase of different product types. A second reason is that general wealth increase causes consumers to strengthen their preferences. The search space on the Web for products has also grown, which makes product search even more important.

There are several problems with the current state of product search on the Web. First, the search engines cannot deal properly with synonyms and homonyms. Second, there is no good support for multiple languages, and more importantly, the aggregation of Web-wide information is seldom done. This is clear when we analyze the way we search for products on the Web. We keep switching back and forth from search results to find, for example, the cheapest price of a certain product. It would be useful if the product information is aggregated and shown to the user in one unified view. Third, there is no parametric Web-wide search available. Users cannot use queries like 'all solar panels which give 12A output and cost less than $2000'.

There are some localized, as opposed to Web-wide, product search Web sites where the user can perform this kind of parametric search. Usually these search engines only support basic product properties. Examples of these properties are the brand, the price, and the review rating of a product. Shopping.com, Google Products, and Shopzilla.com are three well-known parametric product search engines of such kind.

A user can search, for example, for a washing machine with a maximum price of £750 of the brand 'Bosch'. Figure 1 shows an example of this search. The user specifies the query constraints and the search engine queries the database, which con-

Email addresses: `vandic@ese.eur.nl` (Damir Vandic), `jwvdam@gmail.com` (Jan-Willem van Dam), `frasincar@ese.eur.nl` (Flavius Frasincar)

tains all products, in order to display the washing machines that fulfill the requirements of the user. As a result of this, only stores which are indexed in the database of the search engine are shown.

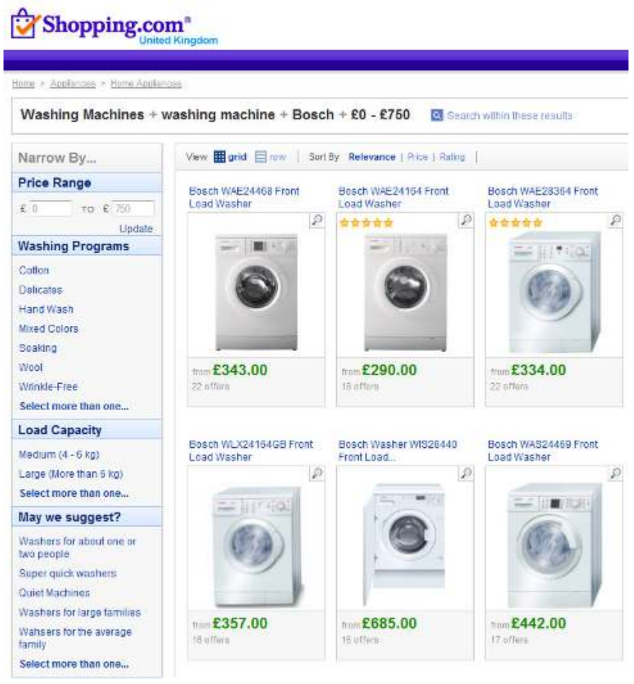

Shopping.com United Kingdom
Home > Appliances > Home Appliances
Washing Machines + washing machine + Bosch + £0 - £750
Search within these results
Narrow By...
Price Range
£ 0 TO £ 750
Update
Washing Programs
Cotton
Delicates
Hand Wash
Mixed Colors
Soaking
Wool
Wrinkle-Free
Select more than one...
Load Capacity
Medium (4 - 6 kg)
Large (More than 6 kg)
Select more than one...
May we suggest?
Washers for about one or two people
Super quick washers
Quiet Machines
Washers for large families
Wahsers for the average family
Select more than one...
View grid row Sort By Relevance | Price | Rating
Bosch WAE24468 Front Load Washer
from £343.00
22 offers
Bosch WAE24164 Front Load Washer
from £290.00
18 offers
Bosch WAE28364 Front Load Washer
from £334.00
22 offers
Bosch WLX24164GB Front Load Washer
from £357.00
18 offers
Bosch Washer WIS28440 Front Load...
from £685.00
18 offers
Bosch WAS24469 Front Load Washer
from £442.00
17 offers

Figure 1: Shopping for a 'Bosch' washing machine on Shopping.com

The databases of these kind of search engines are updated through application programming interfaces (APIs) of Web shops that sell products. Of course, not every Web shop has an API and/or data feed possibilities. Furthermore, every search engine has its own standards which have to be obeyed by the Web shops. For instance, the API of Shopping.com is different than the API of Shopzilla. This means that not every Web shop will have their data prepared for both Shopping.com and Shopzilla. As it is costly to adjust data to a standard, it is not likely that a single search engine will receive data from all Web shops. By annotating Web pages with information on the Semantic Web, the APIs of nowadays can be made obsolete. The annotated information is also publicly available, which enables a search engine to gather product information directly from Web pages.

There is one severe consequence of the current situation of product search. Because a user is not going to view all presented search results, there is a chance that (s)he cannot precisely find a product that matches his or her criteria. What happens is that users more quickly start to focus on the price and give less weight to the product features. The result is that a fierce price competition arises. This can be considered negative for both consumers and companies, as a user can prefer a product which meets all requirements but has a slightly higher price. With semantic search, the companies can develop new business models, because consumers can find the very specific products they are searching for, easier than before.

We have seen how Semantic Web can enhance online product search. In order to make this work, automatic aggregation of product information has to be made possible. The main goal of this research is to show how we can effectively aggregate product information from different sources. In this paper, we focus particularly focus on aligning product names (product identification) and categories (category mapping) from different Web shops. Product identification is the first step when an annotated Web page is processed. The actual product that is described on that Web page has to be identified in order to be able to perform aggregation of product properties (e.g., prices, reviews, etc.). Category mapping is necessary in order to make it possible for users to use the category facet in the user interface. Each product category that occurs in the product description needs to be mapped to a category from the internal product category hierarchy. Part of this research, we provide an implementation of this platform, the XploreProducts.com Web application, which uses a multifaceted user interface and is available online at `http://www.xploreproducts.com`. We do not cover product search engine algorithms, as our focus lies on the aggregation of product information. Further, we propose a solution that solves several issues that current product search engines face. These issues are based on the fact that (product) information on pages is often not well-structured. Product information is usually annotated using a loose vocabulary because there is no widely accepted standard for product names, categories, currencies, scales, units, etc. [2].

Web shops can provide their product offerings on XploreProducts.com simply by supplying the URLs of their product Web pages. This is called the *ping service* of XploreProducts.com. The provided URLs should point to Web pages that have been annotated using the data-vocabulary.org RDFa vocabulary. We choose to use this vocabulary because it is easy to use and already supported by Google. This vocabulary is a good example of a simple and non-restrictive (loose) vocabulary. A loose vocabulary is characterized by the fact that the *range* of properties is not well-defined. For example, for the range of the *price* property in the data-vocabulary.org RDFa vocabulary, it is not specified if it should be a string, float data type, or some other data type. Some users can specify '$199!' as the price while others might use '199.99'. This makes it easy to use by users, but poses issues for automatic aggregation of information.

Although Google is supporting the data-vocabulary.org vocabulary, it is not using it to improve the search engine by aggregating the product information that is annotated on Web pages. The search queries are still based on keywords, like in every traditional search engine. Currently, Google is only using these annotations for the presentation of their search results. Figure 2 shows a screen shot of one of the results of a search query on Google.com. As a result of annotating the Web page of the 'Art of Pizza' restaurant on yelp.com, Google can show information about the price, reviews, etc. We expect that Google's support for these, so-called, rich snippets [3] will lead to a fast adoption of RDFa and/or microformats, to improve search, presentation, and ranking of information. With our re-

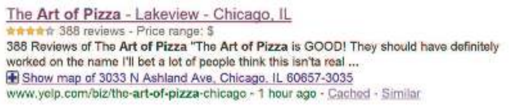

The **Art of Pizza** - Lakeview - Chicago, IL
★★★★☆ 388 reviews - Price range: $
388 Reviews of The **Art of Pizza** "The **Art of Pizza** is GOOD! They should have definitely worked on the name I'll bet a lot of people think this isn'ta real ...
Show map of 3033 N Ashland Ave, Chicago, IL 60657-3035
www.yelp.com/biz/the-**art-of-pizza**-chicago - 1 hour ago - Cached - Similar

Figure 2: A result of searching for 'Art of Pizza restaurant' on Google.com

search, we want to show how the Semantic Web can be used to improve the search and exploration of products on the Web by aggregating information from various sources.

The contribution of this paper stems from several aspects. First, we propose an algorithm that is able to identify products with a high precision. Second, we propose an algorithm for the mapping of a category from an external product category taxonomy to an internal product category taxonomy. This algorithm is compared with two earlier approaches and the results show that our approach gives better performance on average. The comparison also gives insight into the relationships between the different approaches. Third, our evaluation is done on significantly larger data sets than previously used in similar work. For the category mapping, we use 6 data sets that contain each 500 manually mapped categories. These manually mapped categories are obtained by three individuals which are not related to this project. Fourth, we implemented our solution and made it available online, to give a snapshot of future product search engines.

The rest of the paper is organized as follows. Section 2 gives background information on the topics addressed in this research. Section 3 discusses related work about product search, Semantic Web, and faceted search. Then, in Section 4, we present the XploreProducts.com platform. In Section 5, XploreProducts.com is evaluated. Last, in Section 6, we draw conclusions and discuss future work.

2. Background

This section provides some background information on the topics that play an important role in this paper. First, we discuss the Web page annotation topic and then we give an introduction to online product search related topics. Further, we provide justifications for the choices we have made in this paper.

2.1. The Semantic Web

The Semantic Web is an extension to the current Web that adds semantics to the Web data. Nowadays Web pages are human-readable and human-understandable, but the information on these pages is not understandable by machines. For instance, when someone states on his personal home page that he works for Philips then this statement is understandable for a human, but for a machine it is just another sequence of characters. Furthermore, it is not clear for the machine that the page actually describes a person and that Philips is a company.

The Semantic Web makes it possible to describe real world objects (concepts) on Web pages. These concepts can be persons, companies, products, etc. Furthermore, it is possible to describe relationships between these concepts. Allowing machines to understand these concepts, facilitates automatic aggregation of information over resources. Thus, if two different Web sites describe the same concept then it is possible to aggregate this information. For instance, the Wikipedia page of 'Barack Obama' does not contain his public agenda, but his own official Web page does. With a semantic search engine it would be possible to combine the information of these two different sources describing the very same concept.

2.2. Semantic Searching versus Traditional Searching

Traditional search engines, like Google or Yahoo!, crawl Web pages and perform keyword-based search. If a user searches for a certain search phrase, like 'Nokia 3 inch screen', all the Web pages which match this phrase are returned. The search engine does not know that 'Nokia' denotes a mobile phone brand and that '3 inch screen' is a feature of that phone. For the search engine these words are just sequences of characters.

With semantic search it would be possible to search for a mobile phone which has a screen size of at least 3 inch. Other mobile phones could be shown based on a different set of criteria. This enables a user to explicitly state its query constraints. In [4] some other semantic search examples are given. One of these examples is the search for all the papers published by 'Eric Miller'. When one would perform a regular keyword-based search 'Eric Miller' on Google, one would find all documents which contain the words 'Eric' and 'Miller', thus also the papers which cite the 'Eric Miller'. With a semantic based search engine it would be possible to state the constraint that you are only interested in papers which are *written* by this author.

2.3. Faceted Navigation versus Keyword Driven Search

Traditional Web search is performed by entering a search phrase with some keywords. After submitting the search phrase all the relevant Web pages are shown, usually ordered by their relevance to the search phrase. For instance, the three most important Web search engines, Google, Yahoo!, and Bing work in such a manner. It is not possible to perform parametric search, i.e., to specify certain attributes in a search query. Thus, in such Web search engines you cannot specify: "I am looking for a product in the category 'Notebooks' with the brand 'Apple' and a price lower than $1,000".

You might have noticed that parametric search is possible in certain Web shops or product comparison Web sites, e.g., Shopping.com. It is possible to narrow your search by specifying preference values for these attributes (facets). This is called 'multifaceted category navigation integrated with keyword search' [5]. This approach has become a very successful way of searching or navigation in information collections [6]. The main idea is to build a set of category hierarchies each corresponding to a different facet (dimension or feature type) that is relevant to the collection to be navigated. Each facet has a hierarchy of values associated with it, e.g., for the facet 'product category' these values could be 'electronics', 'laptops', and 'televisions'. After designing the facet hierarchies, each item in the collection can be assigned any number of labels from the facet hierarchies. The resulting interface is known as faceted navigation, or is sometimes called guided navigation [7].

2.4. *Annotating Pages*

In order to enable semantic search engines like the one described above, pages need to be annotated. As a consequence, some extra mark-up needs to be added to existing, static or dynamic, (X)HTML pages. For annotating Web pages, two possible languages exist, namely microformats and RDFa. Microformats are small XHTML tags which represent things like people, products, events, reviews, and tags in Web pages. However, these formats are less flexible than RDFa, but, as a result of this, the language is more easily understood by Webmasters who usually do not have a background knowledge in the Semantic Web.

The World Wide Web Consortium (W3C) supports its own initiative, namely the 'Resource Description Framework in attributes' (RDFa) [8]. RDFa is a W3C Recommendation that adds a set of attribute-level extensions to XHTML for embedding rich metadata within Web documents. The RDF data model mapping enables its use for embedding RDF [9] triples within XHTML documents, and it also enables the extraction of RDF model triples by compliant user agents.

2.4.1. *Resource Description Framework (RDF)*

In order to better understand the annotation of Web pages, we briefly highlight some features of the Resource Description Framework (RDF). RDF, as the name already indicates, describes resources. Each resource is denoted by a URI, this is in most cases just a URL. In RDF everything is a resource, including properties and classes. An RDF document is a collection of statements. A statement is a triple which contains a *subject*, a *predicate*, and an *object*. Each of these is a resource, except for the object, which can also be a literal, either a string or data type value (integer, float, etc.).

Let us consider the statement: '*Product X* is called *Nokia N97 Smartphone 32 GB WCDMA (UMTS) / GSM*'. Assume that the Web page fragment `http://example.com/#X` describes this product. If we want to create an RDF statement about this product we need to first decide which vocabulary to use. Because the data-vocabulary.org vocabulary is supported by Google and our platform makes use of it, let us consider this vocabulary. This vocabulary has a predicate called *name*. So in order to make the above statement, we create the triple:

subject: `http://example.com/#X`
predicate: `http://rdf.data-vocabulary.org/#name`
object: "Nokia N97 Smartphone 32 GB WCDMA (UMTS) / GSM"

Graphically, this statement can be visualized as shown in Figure 3.

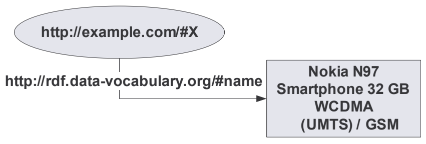

http://example.com/#X
http://rdf.data-vocabulary.org/#name
Nokia N97 Smartphone 32 GB WCDMA (UMTS) / GSM

Figure 3: An example of an RDF statement

2.4.2. *RDF in Attributes (RDFa)*

As already mentioned, today's Web is built predominantly for human consumption. Despite the fact that machine-understandable data starts to appear on the Web, it is typically distributed in a separate file, with a specific format, and limited correspondence between the human and machine versions [8].

The essence of RDFa is to provide a set of attributes that can be used to carry annotations (metadata) in an XML language (hence the 'a' in RDFa). RDFa is to be used with XHTML 1.x or higher. Thus, only one XHTML file exists, which contains the information and this information is readable by both humans and machines. An example of XHTML content, annotated with RDFa, is shown below:

```
<div xmlns:v="http://rdf.data-vocabulary.org/#"
     typeof="v:Product">
  <span property="v:brand">Nokia</span>
  <span property="v:category">Mobile Phones</span>
  <span property="v:name">Nokia N97 Smartphone 32 GB WCDMA
  (UMTS) / GSM</span>
  <span property="v:description">The Nokia N97 introduces
  the concept of 'social location'. With integrated A-GPS
  sensors and an electronic compass, the Nokia N97 mobile
  computer intuitively understands where it is. The Nokia
  N97 makes it easy to update social networks automatical-
  ly with real-time information.</span>
</div>
<a href="http://europe.nokia.com/find-products/devices/nok-
ia-n97" rel="v:url">Nokia Europe - Nokia N97 - Products</a>
```

The `typeof="v:Product"` declaration indicates that the marked-up content represents a product. Each product property (like the name and the brand) is labeled using the attribute `property`. The predicate names are prefixed with `v:`, this shows that the attribute is from the `v` namespace, in this case the `http://rdf.data-vocabulary.org/` namespace. Each attribute has a certain meaning that is defined in the vocabulary.

The namespace declaration using `xmlns`, shown in the XHTML above, indicates which vocabulary is used. The `xmlns:v="http://rdf.data-vocabulary.org/#"` namespace declaration is the vocabulary which is supported by Google. There are several other (more sophisticated) vocabularies available online. Another well-known ontology for describing products is the GoodRelations [10] ontology The GoodRelations ontology is more advanced than Google's vocabulary. For instance, with GoodRelations it is possible to specify the warranty, the delivery options, the payment methods, the currency, etc. GoodRelations is not to be compared with data-vocabulary.org, as GoodRelations is a fully-fledged OWL ontology where the user is able to specify mainly product specific properties. Examples of these kind of properties are 'the number of rooms when one is selling a house' or 'the battery life time for a mobile phone'. With the Google supported RDFa, it is only possible to specify relatively simple and general information about a product, like the product name, the brand, the category, the price, the review rating, etc.

As already mentioned, in this paper we will use the less advanced vocabulary from data-vocabulary.org, which is supported by Google. We chose for this vocabluary as we want to show that even with vocabularies without many constraints, it is possible to perform advanced product search on Web level. Furthermore, as Google, the largest Web search engine, is supporting this vocabulary, we expect this vocabulary to be commonly used worldwide in the near future.

2.4.3. SPARQL

Just like with relational databases, a query language to query the data from a set of RDF statements is needed. For relational databases the most commonly used language is SQL. For RDF, there is a language which is similar to SQL, namely SPARQL [11].

Consider the set of statements shown in Figure 4. An example query which retrieves all resources which have the name property is shown below.

```
PREFIX v: <http://rdf.data-vocabulary.org/#>

SELECT ?product ?name
WHERE {
   ?product v:name ?name .
}
```

If we want only the products with a price larger than $400.00,

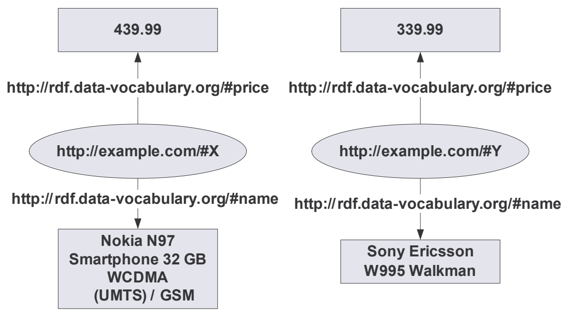

439.99
http://rdf.data-vocabulary.org/#price
http://example.com/#X
http://rdf.data-vocabulary.org/#name
Nokia N97 Smartphone 32 GB WCDMA (UMTS) / GSM
339.99
http://rdf.data-vocabulary.org/#price
http://example.com/#Y
http://rdf.data-vocabulary.org/#name
Sony Ericsson W995 Walkman

Figure 4: An example of a SPARQL query on RDF statements

then the following query can be used:

```
PREFIX v: <http://rdf.data-vocabulary.org/#>


SELECT ?product ?name
WHERE {
   ?product v:name ?name .
   ?product v:price ?price .
   FILTER(?price > 400) .
}
```

3. Related Work

In this section, we first discuss literature that addresses Semantic Web approaches for product search. After that, we give an overview of existing product ontologies. Then we discuss related product search approaches. We conclude this section with the description of the product search inherent categorization problem.

3.1. Products and the Semantic Web

A lot of research has been performed to show the potential of using the Semantic Web for improving the search for products on the Web, e.g., [12], [13], and [14]. There exist several general categorization standards for products and services, like UNSPSC, eOTD, eClass, and the Rosettanet Technical Dictionary (RNTD). In [15] a comparison is made between all these standards and the authors reveal weaknesses and shortcomings in all of the four standards. It depends on the purpose and segment of interest which standard performs the best.

3.1.1. Ontologies

Based on the just mentioned standards eClassOWL [16] and unspscOWL [17] are developed. These are general product and service ontologies. However, according to [10], these general product and service ontologies do not provide the means required for e-commerce on the Semantic Web. Annotating products require more advanced statements than just ‘a Sony Vaio is an instance of the product class Notebooks’. Therefore, several ontologies for Semantic Web-based e-commerce on Web scale have been proposed, e.g., GoodRelations [10].

With these e-commerce ontologies it is possible to specify and annotate product specific properties and business relations between the product and the merchant. For instance, this leads to annotations which answer questions like ‘Is this product in stock?’, ‘Is this an offer for rent or to sell?’, ‘Is this a special price?’, ‘What is the size of this mobile phone?’, or ‘For which period does this offer hold?’.

Unfortunately, these ontologies are not yet commonly used. However, well-known Web tools as Joomla! [18], osCommerce [19], and Drupal [20] have plug-ins or even support the GoodRelations ontology. So, we believe it is just a matter of time before this ontology will be used on a large scale. Nevertheless, as already mentioned, for this paper we use a simpler vocabulary: the data-vocabulary.org vocabulary. We chose for this vocabulary as Google currently already supports it, and thus it allows for the immediate applicability of our results.

3.2. Product Search

Although the potential of using the Semantic Web for improving the search for products on the Web is emphasized in [12], [13], and [14], surprisingly, there is not much research performed where the focus is on semantic product search.

In [21] a search and comparison system for products is proposed. Their approach is based on the assumption that each shopping mall offers a product ontology and a product search service based on Web Services. The authors state that more shopping malls are expected to offer Web Services, and their search engine will be able to use Web Services to collect structured product information. We believe that this assumption is a weak point, because the Semantic Web offers a lot more and makes the data semantics directly available on the Web. Further, there are still many small shopping sites that do not plan to support Web services at all. Therefore, we propose a product search engine which makes only use of Semantic Web technologies.

The authors of [22] propose a completely semantic-driven product search engine is proposed. Unfortunately, this engine is not a general search engine, but it is an engine for a specific product category. In this work 'mobile phones' is chosen as a practical example. Furthermore, the authors use a self-made ontology, instead of one of the well-known earlier mentioned ontologies. In this paper, we propose a general product search engine which makes use of standardized Semantic Web technologies and vocabularies.

In the literature we can find other related work that covers product search in an e-commerce environment. In [23] and [24] a federated product search application using heterogeneous sources is proposed. Unfortunately, the authors do not use advanced faceted search, as only one facet, the product category, is available. This approach is similar to keyword-based search, with the possibility to specify the product category, separately from the keywords. Furthermore, Semantic Web technologies are ignored as the authors use Web services and semi-structured HTML to extract product data.

The authors of [25] propose a ranking algorithm that can be applied to product descriptions in an e-commerce environment for product search. The algorithm is based on concept contradiction and concept abduction. The difference between this approach and our approach is that we present a solution that also includes product identification and category mapping. The algorithm presented in [25], as opposed to our approach, assumes that the e-commerce data is already stored in some formal language, like OWL.

In [26] a framework similar to XploreProducts.com is proposed. The authors use a three-layered architecture as a basis for their product information retrieval framework. First, the framework extracts product information from HTML pages using predefined information extraction rules. Next, this data is stored and exposed using two search methods, i.e., a search method based on the vector space model, and a search method that is based on SPARQL. Last, the authors provide an easy to use graphical implementation of the search interfaces. Compared to our approach, the solution presented in [26] is different in several ways. For example, the authors do not focus on the fact that multiple ontologies can be used. As a consequence, there is no product name identification and no solution for the alignment of product category taxonomies.

3.3. Category Mapping

As already discussed in subsection 3.1, several categorization standards exist for products and services. Furthermore, many Web shops use their own vocabulary for categorizing their products and/or services. For instance, we searched for a 'Nokia N97' cell phone on Amazon.com and Circuitcity.com. Amazon associates this phone with the category 'Cell phones & services' while Circuitcity.com associates this phone with the category 'Unlocked Cell Phones'. Also, the hierarchical product category relationships differ over Web shops.

As there is no general product classification standard, the IEEE Intelligent Systems Magazine even launched an international research challenge to come up with a generic model and working solution that is able to (semi-)automatically map a given product description in two different e-commerce product classification standards [27]. In this paper we focus on mapping product categories to an existing product category hierarchy. The eClassOWL product classification standard is a comprehensive classification scheme and is freely available for non-commercial use. We have chosen not to use it as it does not fit our domain so well as this ontology contains many Business-to-Business product categories. We have therefore chosen to use an informal, yet comprehensive, category hierarchy, namely the one of Shopping.com.

As already mentioned in the introduction, we compare our category mapping approach with two other approaches from literature. These approaches showed good results with respect to precision and recall. The goal of category mapping is to map a category from the source taxonomy to a category from the target taxonomy. First, we compare our approach against the approach presented by Park & Kim [28]. Their approach consists of three main steps. In the first step, they apply word sense disambiguation to find the correct synonym set from WordNet for a given source category. This is done by analyzing and comparing the hypernym tree of the source and target paths. The second step consists of syntactic matching of terms in the synonym set in order to find candidate paths from the target taxonomy. In the last and final step, a similarity between the source path and all previously identified candidate paths is computed. If the highest similarity is above some threshold, the corresponding candidate target path is selected as the mapping, otherwise the algorithm does not provide a mapping for the category. The similarity between two paths is the average of the proposed co-occurrence similarity and order-consistency measure. The co-occurrence similarity represents the number of common categories between the source path and a candidate path. This similarity takes into account the synonyms of a category. The order-consistency measure takes into account the ratio of correctly ordered matching categories, which emphasizes that the order in which categories occur in a path is important.

Second, we compare our approach with PROMPT, presented in [29]. PROMPT is a very popular ontology mapping toolkit which has been used in many domains and evaluated thoroughly, for example in [30]. PROMPT takes as input a set of *anchor point* pairs. An anchor point is a pair of categories, where one category comes from the source taxonomy and the other from the target taxonomy. These initial mappings can be either obtained through a manual, semi-automatic, or fully automatic process. Next, all possible paths are generated where the start and end point of a path is an anchor point pair. This is done for both the source and target taxonomy. With these two sets of paths, pairs of paths of equal length are considered, where one path is from the source taxonomy and the other from the target taxonomy. For each pair, the categories in between the anchor points at the same position are registered as being similar and their similarity is increased with a constant. When this procedure finishes, all pairs of categories that are similar can be considered as a mapping. In order to improve the mappings and make the algorithm less sensitive to noise, PROMPT proposes to filter the scores. This filtering can be done by computing the median of all mapping scores and removing all mapping scores that are smaller than the median. PROMPT has two parameters, the first determines the maximum path length for paths that are generated in the first step. The second parameter determines whether or not the scores should be filtered.

In [31] a method for ontology mapping is proposed, which can be used in the context of product category mapping. The basic idea of this algorithm is that it uses instance data to semantically enrich the ontologies that need to be mapped. This is done in order to be able to compute the semantic similarity (cosine similarity) between concepts in the ontology. The concepts are represented as vectors, where each element is representing a word from a concept instance description. To obtain the concept vector, an average is taken over all vectors of documents that belong to a concept. Our approach is different from this one as it does not assume that we have any instance information, i.e., it maps product categories from two sources without knowing the products that are in these categories. As our framework requires product category mapping without any knowledge of products, we have chosen not compare our approach with this one.

The authors of [32] take a different approach and use Bayesian decision making for the purpose of ontology mapping. The basic idea is that the ontology mapping task is transformed into an decision problem by computing a probability for each possible mapping. The authors present several strategies for computing these probabilities, including relying on the names of the concepts and concept instance data. Then the Bayesian risk (also sometimes called the loss) is minimized in order to obtain the best mapping. We have not considered this approach as the results from the paper showed that the algorithm performs best with the use of instance data and in our case we cannot assume to have this information.

Another related algorithm is proposed in [33]. In this paper, an algorithm based on similarity flooding [34] is proposed that is able to match substructures of a taxonomy, such as concept $\rightarrow$ property $\rightarrow$ concept. In their paper, they evaluate their technique on data sets that contain educational concept maps drawn by students. We do not consider this technique as it might be inappropriate because it does not provide solutions for issues that occur specifically for product category taxonomies, such as path ambiguity, and synonyms.

4. The XploreProducts.com Platform

In this section we discuss the XploreProducts.com platform. We first provide a problem description and then move on to a detailed discussion of the platform.

4.1. Problem Description

As already explained, in e-commerce there exist many heterogeneous data sources. Every Web shop uses its own standards, languages, vocabularies, and ontologies. With XploreProducts.com we aim to search over different data sources. We focus on the following heterogeneous aspects which exist when querying over several Web shops:

- the use of different product names for identical products;
- the use of different category standards;
- the use of different category hierarchies;
- the use of different currencies;
- the use of different review rating scales;
- aggregating over review rating information.

4.2. Identical Product Recognition

In order to be able to aggregate information across different sources, we need to identify the product names that represent the same product. We now describe an algorithm which determines if two products are equal given their names. The products to be compared are assumed to be annotated with the same brand.

Before we can step into the details of the product name identification algorithm, we need to explain an existing text similarity measure. The Levenshtein distance [35] is a metric for measuring the amount of difference between two strings (i.e., the so-called edit distance). The Levenshtein distance between two strings is given by the minimum number of operations needed to transform one string into the other, where an operation is an insertion, deletion, or substitution of a single character. It can be considered a generalization of the Hamming distance, which is used for strings of the same length and only considers substitution edits. It is often used in applications that need to determine how similar, or different, two strings are.

For example, the Levenshtein distance between ‘hair’ and ‘stairs’ is 3, since the following three edits change one into the other, and there is no way to do it with fewer than three edits:

1. hair → shair (insert of ‘s’ at the beginning)
2. shair → stair (substitution of ‘h’ for ‘t’)
3. stair → stairs (insert ‘s’ at the end).

This distance is called the *absolute* Levenshtein distance. We denote it by alv_{ij}, which is the absolute Levenshtein distance between strings i and j.

In our framework we use the *normalized* Levenshtein distance, which is a function of the absolute Levenshtein distance. We use the notation lv_{ij}, which is the normalized Levenshtein distance between strings i and j. The normalized Levenshtein distance is defined as

$$\mathrm{lv}(x, y) = \frac{\mathrm{alv}(x,y)}{\max\big(\mathrm{length}(x), \mathrm{length}(y)\big)} \tag{1}$$

The normalized Levenshtein distance addresses the issue of short string lengths. If you have two strings, of both length 24, then an absolute Levenshtein distance of 3 is not large. However, with two strings of length 6, this distance is quite large as it is 50% of the tag length. According to the absolute Levenshtein distance these two distances are the same. But the normalized Levenshtein distances are in this case 0.125 and 0.5. This indicates that, according to the normalized Levenshtein distance, the two pairs of strings do not have the same distance, i.e., the first pair is more similar.

The product name identification process is summarized in Algorithm 1. The algorithm has two input parameters, the two product names, and four control parameters (thresholds and weights). We replace common words like ‘and’, ‘or’ and characters like ‘&’, ‘/’, ‘-’, with a space. This eliminates any ‘noise’ in the product names and makes them more easy to compare.

Several functions are defined, which need some explanation. The function calcCosineSim(a, b) calculates the cosine similarity between two product names a and b. It is defined by the following equation:

$$\mathrm{calcCosineSim}(a, b) = \frac{|a \cap b|}{\sqrt{|a|}\,\sqrt{|b|}} \tag{2}$$

where a and b are the set of words of the first and second product name, respectively. So for the product names ‘sony ericsson X1’ and ‘xperia X1’ the cosine similarity is $\frac{1}{\sqrt{3}\sqrt{2}} = 0.408$.

The function extractModelWords(a) is used to extract the words from a product name which contain alphabetic/punctuation, and numeric characters, i.e., the possible ‘model words’. Table 1 shows ‘model words’ for some example product names.

With avgLvSim(X, Y), the average Levenshtein similarity between two sets of words can be computed. In some cases, the set of words X and Y are preprocessed before using this function. We then consider only the first maximal n elements from each set, so that both sets are of an equal size. Using the normalized Levenshtein distance function lv(x, y), we can give the definition of the function avgLvSim(X, Y), where X and Y are sets, as following:

$$\mathrm{avgLvSim}(X, Y) = \sum_{x \in X} \sum_{y \in Y} (1 - \mathrm{lv}(x, y)) \frac{\mathrm{length}(x) + \mathrm{length}(y)}{\sum_{x \in X} \sum_{y \in Y} \mathrm{length}(x) + \mathrm{length}(y)} \tag{3}$$

where length$(\cdot)$ returns the length of a string.

The function avgLvSimMW is very similar to avgLvSim, the only difference is that the average is taken over only the model words which have the non-numeric part *approximately the same*

Table 1: Product names with corresponding ‘model words’

Product Name	Model Words
Nikon D700 Digital SLR Camera - 3” Active Matrix TFT Color LCD	D700, 3”
Nikon D700 12.1MP Digital SLR Camera	D700, 12.1MP
Apple iPod touch 32GB Flash Portable Media Player	32GB
Apple iPod touch 32 GB (3rd Generation) NEWEST MODEL by Apple	3rd
Samsung BD-P1600 Blu-Ray Player - 1080p, HDMI, USB, Ethernet, Remote Control	BD-P1600, 1080p
Samsung BD-P1600 - Blu-ray DVD Player	BD-P1600
Sony DSC-W290 Cyber-shot 12 Megapixel Digital Camera with 5x Optical Zoom, 3” LCD, HD, Black	DSC-W290, 5x, 3”
Sony Cyber-Shot DSC-W290 - 12.1 Megapixels, 5x Optical Zoom, 3.0” LCD, Black	DSC-W290, 5x, 3.0”

and the numeric characters are the same:

$$\text{avgLvSimMW}(X, Y) = \sum_{x \in X} \sum_{y \in Y \wedge x \approx y} (1 - \text{lv}(x, y)) \frac{\text{length}(x) + \text{length}(y)}{\sum_{x \in X} \sum_{y \in Y \wedge x \approx y} \text{length}(x) + \text{length}(y)} \quad (4)$$

The process of recognizing identical products is outlined in Algorithm 1. The algorithm starts by checking for a general high cosine similarity and returns ‘true’ if it finds one, indicating that the product names represent the same product, as shown in lines 1 through 4. On lines 5 and 6, the ‘model words’ from both product names are extracted and stored in sets. Then a check is performed to see if the two products can immediately be classified as different. This is done by examining if there is a pair of model words where the non-numeric part is *approximately the same* but the numeric part is not. We define *approximately the same* by using a normalized Levenshtein distance with a certain threshold. The argument for this is that when one finds two ‘model words’ which are *approximately the same* with respect to the non-numeric characters, but not with respect to the numeric part, then the two ‘model words’ belong to two different products, i.e., the ‘model words’ for the Sony VGN laptop series: ‘VGN-NW350FS’ versus ‘VGN-NW320FS’.

If the algorithm did not return false, the similarity between the two product names is computed, as shown on line 10. Next, a check is performed to see if the computed similarity on line 10 needs to be updated. The similarity is updated when there is at least one pair (a ‘model word’ from set X and a ‘model word’ from set Y), where the two ‘model words’ have *approximately the same* non-numeric characters and equal numeric characters. The condition is necessary as we do not want to compute the similarity between ‘model words’ pairs like ‘3Ghz’ and ‘24inch’. If this condition is satisfied, the similarity is updated by giving more weight to the similarity between the ‘model words’ (only the pairs which pass the previous condition test), as shown on line 13. The reason for the update of the initially computed product name similarity is that ‘model words’ play an important role in product identification, as these are natural identifiers of products. We therefore give a relatively large weight, i.e., δ, to this similarity when updating the final similarity. As last, if the final similarity, *finalNameSim*, is higher than the threshold ϵ, the product names represent the same product. Consider the product names ‘Panasonic DMPBD605 Blu-ray Player - 1080p, BD-Live’ and ‘Panasonic DMP-BD60K Blu-ray DVD Player’. These two product names appear to be the same but they do not represent the same product, as indicated by the words ‘DMPBD605’ and ‘DMP-BD60K’, these are two different model numbers.

4.3. Category Mapping

As discussed in the previous subsection, Web shops use different names for describing product categories on the Web. Furthermore, different hierarchies are used. For instance, ‘Nintendo DS Games’ can be a child of ‘Games’, where on another Web site it is a child of a more specific category ‘Console Games’. To cope with this problem we use an existing product

Algorithm 1 Product name identification

```
Require: The input: product names $a$, $b$
Require: The out: *true* if the product names $a$ and $b$ represent the same product, *false* otherwise
Require: The parameters:
    • $\alpha$, the threshold for the cosine similarity of the product names (*nameCosineSim*)
    • $\beta$, the weight for the cosine similarity of the product names (*nameCosineSim*)
    • $\delta$, the weight for the cosine similarity of the identified 'model words' in the product names (*modelWordSimVal*)
    • $\epsilon$, the threshold for the final similarity between two product names (*finalNameSim*)
Require: calcCosineSim $(a, b)$ gives the cosine similarity, as defined by Equation 2, between product names $a$ and $b$
Require: extractModelWords $(a)$ returns a set of 'model words' (words which contain both numeric and alphabetic characters), extracted from product name $a$
Require: avgLvSim $(X, Y)$ returns a similarity $\in [0, 1]$ between two sets of 'model words' $X$ and $Y$
Require: avgLvSimMW $(X, Y)$ returns a similarity $\in [0, 1]$ between two sets of 'model words' $X$ and $Y$, considering only elements $x \in X$ and $y \in Y$ such that the non-numeric parts are approximately equal ($x \approx y$).
 1: nameCosineSim = calcCosineSim $(a, b)$
 2: if nameCosineSim $> \alpha$ then
 3:     return true {the product names represent the same product}
 4: end if
 5: modelWordsA = extractModelWords $(a)$
 6: modelWordsB = extractModelWords $(b)$
    {analyze the two sets of 'model words', comparing each 'model word' from one set to each 'model word' from the other set (called a pair)}
 7: if found a pair where non-numeric characters are approximately the same AND numeric characters are not the same then
 8:     return false {the product names do not represent the same product}
 9: end if
    {compute initial product name similarity}
10: finalNameSim = $\beta \times$ nameCosineSim $+ (1 - \beta) \times$ avgLvSim $(a, b)$
    {check if we have a pair of 'model words' which are likely to represent the same}
11: if there is at least one pair where the non-numeric characters are approximately the same AND the numeric characters are the same then
12:     modelWordSimVal = avgLvSimMW (modelWordsA, modelWordsB)
13:     finalNameSim $= \delta \times$ modelWordSimVal $+ (1 - \delta) \times$ finalNameSim
        {updated the calculated product name similarity}
14: end if
15: return finalNameSim $> \epsilon$ {the product names represent the same product if true, false otherwise}
```

category hierarchy, i.e., the internal taxonomy, to map the new product categories to. We chose for the Shopping.com product hierarchy as it is freely accessible and available through their API.

As with the identification of the products names which describe identical products, we have to identify to which existing product category our system should map new unknown product categories. There are two difficulties with this process, as one has to deal with syntactic variations and with semantic variations. The syntactic variations are for example singular / plural forms, abbreviations, and typographical mistakes. The semantic variations are synonyms and homonyms. In order to deal with these issues we developed an algorithm which is able to take into account both of these variations.

Algorithm 2 shows the steps taken to find a matching product category in the internal taxonomy, given a new category. We assume that the category taxonomy of the new category is available. The algorithm starts with a loop that runs for each category in the target taxonomy. The idea is that we compute a similarity between the input category and all target categories and choose the target category that has the highest similarity. If the highest similarity is below a certain threshold, the algorithm does not map the input category. We denote an input category by c and a target category by $target$. The way we compute the similarity between two categories is that we first generate the path that goes to the root category in the taxonomy for both categories. Then, we compute a weighted average of similarities between pairs of categories that are on the same level from the two paths. For example, consider a path $\mathbf{p}_1 = (c_1, c_2, c_3, c_4)$, obtained from the input category c_1, and a target path $\mathbf{p}_2 = (t_1, t_2, t_3)$. In this example, c_4 is the root category of the source taxonomy and t_3 is the root category of the target taxonomy. The similarity between $\mathbf{p}_1$ and $\mathbf{p}_2$ would be an weighted average of the similarity between (c_1, t_1), (c_2, t_2), and (c_3, t_3). The category c_4 is not used as the target path ($\mathbf{p}_2$) is only 3 categories long. The weights that are used to compute the weighted average give the combination (c_1, t_1) more weight than (c_3, t_3). The reasoning behind this is that a pair of categories close to the to be mapped category is more important than one far away. We have chosen for the following weight function:

$$w_i = \frac{1}{i+1} / \sum_{j=1}^{n} \frac{1}{j} \tag{5}$$

where i represents the zero-based index for the location on the path and n is the total number of elements in the weighted average. So for the weights for the above mentioned example are $w_0 = (1/1)/(1/1 + 1/2 + 1/3)$, $w_1 = (1/2)/(1/1 + 1/2 + 1/3)$, and $w_2 = (1/3)/(1/1 + 1/2 + 1/3)$. These values are respectively 0.55, 0.27, and 0.18.

Before we compute a similarity between two categories, the category to be mapped c and the target category *target* are cleaned. This is happening on lines 3 and 4 for the input category and a target category. The cleaning of category names is necessary in order to remove possible 'noise'. For example, some users write 'Camera and Photography' while others might write the abbreviated form 'Camera & Photography'. We solve this issue by replacing occurrences of both 'and' and '&' by a space character. After the category names are cleaned, the similarity can be computed, as done on line 5. The function that is used to calculated the similarity between two cleaned category names is given by:

$$\text{getCatSim}(a, b) = \lambda \cdot \text{calcCosineSim}(a, b) + (1 - \lambda) \cdot \text{avgLvSim}(a, b) \quad (6)$$

where a and b are sets of words, calcCosineSim (a, b) is defined by Equation 2 and avgLvSim (a, b) is defined by Equation 3. The sets a and b are obtained by splitting a category name on the space character, this is achieved by using the function cleanAndSplit $(\cdot)$, which is also used to clean the product names. The multiplication with w_0 indicates that the first similarity is weighted by w_0, as already explained.

The algorithm continues by computing the same similarity for all pairs of parent categories of the input category and target category, this is done in 7 to 16. The index for the weight is increased on line 15. On line 17, when all ancestor similarities are computed, the final similarity is added to the set of all similarities S. After the similarities with all target categories are computed, a category needs to be chosen. If the highest similarity in the set S is above some threshold, the corresponding target category is selected as the matching product category.

Algorithm 2 Category mapping algorithm

```
Require: The input: a new category $c$ to be matched to an existing category from the framework
Require: The taxonomy $T$ (existing categories provided by the framework, for example, the categories from Shopping.com)
Require: The function cleanAndSplit$(a)$ 'cleans' a category name, i.e., replaces 'and', 'or', '&', '(', ')', ',', and other special characters with a space and returns set of words obtained by splitting $a$ on the space character
Require: The vector $\mathbf{w}$ where each elements represent the weight for the similarity, i.e., $w_0$ is the weight for the leaf nodes similarity, $w_1$ is for their parents, etc.
Require: The function getCatSim$(a, b)$ gives a similarity $\in [0, 1]$ between category names $a$ and $b$
Require: The minimum similarity threshold $\phi$
 1: $S = \{\}$
 2: for $target \in T$ do
 3:     $a$ = cleanAndSplit$(c)$
 4:     $b$ = cleanAndSplit$(target)$
 5:     $sim$ = getCatSim$(a, b) \times w_0$
 6:     $i = 1$
 7:     for $targetAncestor \in$ {ancestors of $target$} do
 8:         $cAncestor$ = nextAncestorOf$(c)$
 9:         if not exists $cAncestor$ then
10:             return
11:         end if
12:         $a$ = cleanAndSplit$(cAncestor)$
13:         $b$ = cleanAndSplit$(targetAncestor)$
14:         $sim = sim$ + getCatSim$(cAncestor, targetAncestor) \times w_i$
15:         $i = i + 1$
16:     end for
17:     $S = S \cup \{(target, sim)\}$
18: end for
    {The category that is the best match for $c$ is the one with the highest similarity in the set $S$}
19: return $\{x \mid (x, m) \in S, m \geq \phi, \forall (y, n) \in S : n \leq m\}$
```

4.3.1. *Different Currencies*

The XploreProducts.com platform also offers support for various currencies. As already mentioned, the data-vocabulary.org vocabulary does not have many restrictions on the values of properties. This means that a price can be expressed as a numeric value (e.g., 149.99) or as a string (e.g., FREE, \$ 149.99, € 149.99). Note also that the decimal separator is not given as this is not known upfront. We use simple heuristics to deal with these issues. Some examples of these heuristics are:

- When the currency is not stated, the domain name of the Web page is used to retrieve the currency of that country (e.g., '.nl' is the Netherlands, which uses the Euro). With non-country domain names, like '.com' or '.org', we use an API [36] to map the IP address of the server to the country where the server is located.

- Statements like 'free', 'free of charge', and 'for nothing' are treated as 0.00. For this purpose we have developed a comprehensive list of lexical representations that denote a null price.

4.4. The Platform

The XploreProducts.com platform consists of two environments, namely an environment where the end-users can search and explore products on the Web and an environment where Web shops are able to ping our platform with their product information containing Web pages. Figure 5 gives an overview of the platform with its two environments. The arrows in the Figure denote a 'uses' relationship, i.e., the 'Faceted Search User Interface' uses the 'SPARQL Query Generator'.

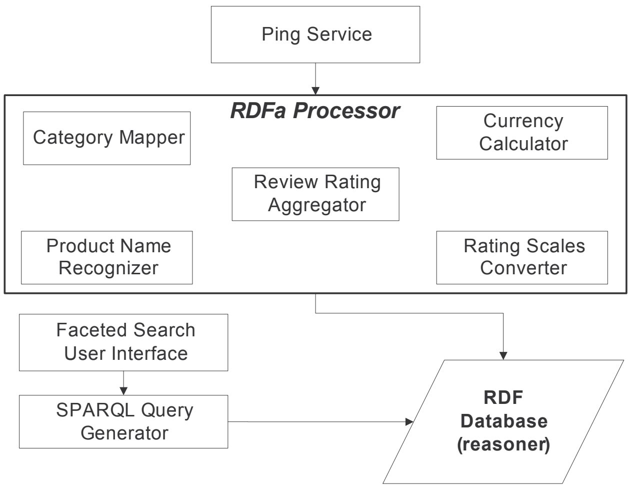


Figure 5: The XploreProducts.com Platform

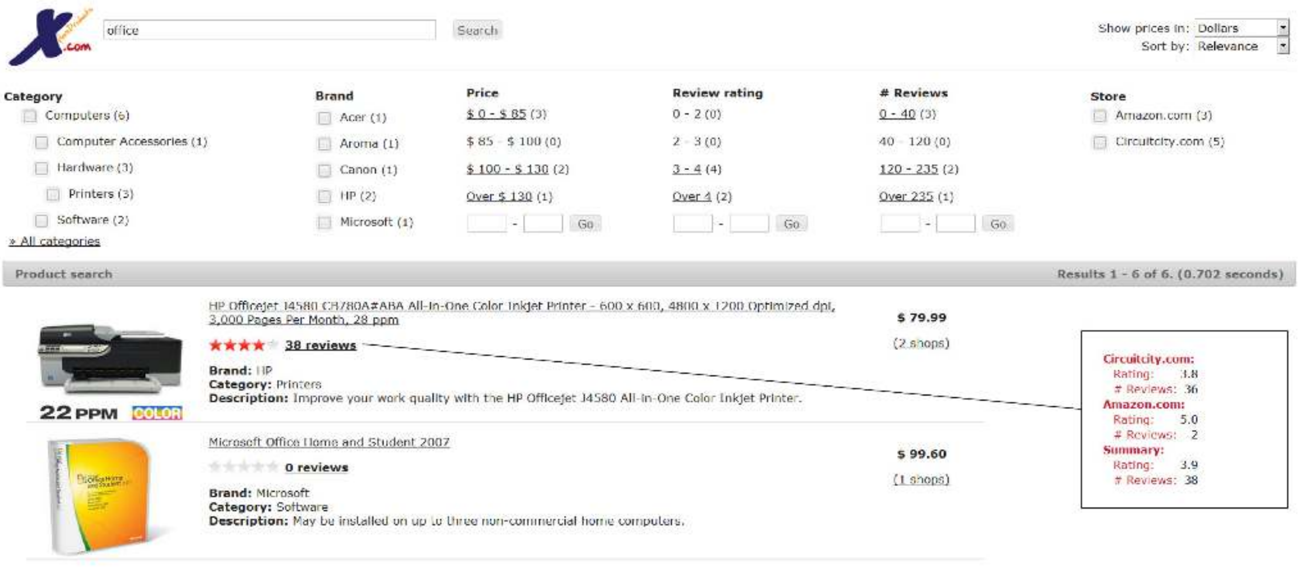
4.4.1. Ping Service

The ping service in XploreProducts.com allows companies, such as online Web shops, to submit their annotated Web pages so that the platform can process and interpret the new information and consequently update the database. This service can be used by consumers in a scenario where one wants to add some product Web pages to the XploreProducts.com database. We think that this service is mostly used by companies that want to have their products listed in as many product search engines as possible. For these companies the XploreProducts.com ping service is an easy and fast way to promote their offerings. In the ping service, the user submits a list of product URLs that need to be added to the database. The user also has the possibility to upload a site map file or specify an online site map URL, if many URLs need to be provided. After a user submits one or more product URLs, the processing starts. The ping service extracts the RDF statements from the annotated HTML pages and uses the services from the 'RDFa Processor' to process the extracted RDF data, as shown in Figure 5. The RDF data is prepared, cleaned, and adjusted to be stored in an RDF database which is compatible with the framework. The database is updated immediately after the processing of the RDF data is finished. When a user searches for his products in the search engine of XploreProducts.com, the results can be instantly viewed.

4.4.2. Searching on XploreProducts.com

For the implementation of the user interface, we use multifaceted category navigation, integrated with keyword search. The facets which we use are: the product category, the brand, the price, the review rating, the number of reviews, and the store name, as shown in Figure 5. When the user enters a query, the SPARQL query generator translates the user's input to a SPARQL query in order to get relevant results from the RDF database. Figure 6 shows an example of searching for 'office'.

Furthermore, one can notice in Figure 6 that aggregation over shop results is performed. The HP printer can be found on both Amazon.com and Circuitcity.com, but the average rating (and



Show prices in: Dollars
Sort by: Relevance

Results 1 - 6 of 6. (0.702 seconds)

HP Officejet J4580 CB780A#ABA All-In-One Color Inkjet Printer - 600 x 600, 4800 x 1200 Optimized dpi, 3,000 Pages Per Month, 28 ppm
38 reviews
Brand: HP
Category: Printers
Description: Improve your work quality with the HP Officejet J4580 All-In-One Color Inkjet Printer.
$ 79.99
(2 shops)

Circuitcity.com:
Rating: 3.8
Reviews: 36
Amazon.com:
Rating: 5.0
Reviews: 2
Summary:
Rating: 3.9
Reviews: 38

Microsoft Office Home and Student 2007
0 reviews
Brand: Microsoft
Category: Software
Description: May be installed on up to three non-commercial home computers.
$ 99.60
(1 shops)

Figure 6: Searching for 'Office' on XploreProducts.com

price) differs across these shops. This is a result of the product name identification, presented in Section 4.2.

The ranges for the prices and the review counts are computed automatically to redistribute the values optimally such that each range contains an equal number of products. This allows the user to browse the results with the most flexibility. Additionally, one is able to change the currency, so that users can compare prices more easily with the currency in their own country. The default value for this filter is U.S. Dollars. When a user changes the currency, not only the prices shown in the results will update, but also the computed ranges will change.

5. Evaluation

In this section we evaluate the algorithms used in XploreProducts.com. First, we analyze the results of the recognition of identical products, an important task of XploreProducts.com as the aggregation of information is dependent on the success of this task. Second, we evaluate the category mapping task of XploreProducts.com, and last, the user interface is evaluated.

5.1. Product Name Identification

For the evaluation of the product name identification algorithm we collected 603 product name combinations from Amazon.com, Circuitcity.com, and other online shops. We manually identified which product names are describing identical products. These mappings are used to facilitate the evaluation of our algorithm, where the automatic mapping is compared to the manual mapping. Some examples of product names which are identified by the algorithm as being identical products, are given in Table 2.

Our final results are summarized in Table 3 and Table 4. Table 3 shows the accuracy, the precision, the recall, and the specificity. These measures are commonly used with *independent binary* classifiers, i.e., which give a yes/no answer for a certain decision. The evaluation is done using a two-way contingency table, in our case given in Table 4. These tables always have four cells: *true positives* (TP), *false positives* (FP), *false negatives* (FN), and *true negatives* (TN). True positives is the number of cases where 'YES' was correct and false positives is the number of cases where 'YES' was incorrect. False negatives is the number of cases where 'NO' was incorrect and true

Table 2: Product name examples which describe identical products

Cluster number	Product names
1	Nikon D700 Digital SLR Camera - 3” Active Matrix TFT Color LCD Nikon D700 12.1MP Digital SLR Camera
2	Dell Vostro 1520 Laptop Computer (Intel Core 2 Duo T6670 250GB/2GB) Dell Vostro 1520 15.4-Inch Widescreen Laptop (Black)
3	Samsung BD-P1600 Blu-Ray Player - 1080p, HDMI, USB, Ethernet, Netflix Ready, DVD/CD, Remote Control Samsung BD-P1600 - Blu-ray DVD Player
4	VIZIO VA370M 37-Inch Full HD 1080p LCD HDTV VIZIO VA370M - 37” Widescreen 1080p LCD HDTV - 15,000:1 Dynamic Contrast Ratio - 6.5ms Response Time

negatives is the number of cases where ‘NO’ was correct. In our case, ‘YES’ represents the decision taken by the algorithm that two product names represent the same product and ‘NO’ that the two products are different. Performance measures from information retrieval are defined and computed from these contingency tables. These measures are recall (r), precision (p), specificity (s), accuracy (ac) and error (e):

$$r = TP/(TP + FN)$$

$$p = TP/(TP + FP)$$

$$s = TN/(FP + TN)$$

$$ac = (TP + TN)/(TP + FP + FN + TN)$$

$$e = (FP + FN)/(TP + FP + FN + TN)$$

The results of the product name identification algorithm look promising with accuracy, precision, recall, and specificity, all above 91%. These results are obtained with $\alpha = 0.80, \beta = 0.10$, $\delta = 0.40$, and $\epsilon = 0.50$. To select these parameters we experimented with every possible parameter combination when using a step size of 0.05 and for all the parameters a range between 0 and 1. The processing time for all the $20 \times 20 \times 20 \times 20 = 160,000$ parameter combinations was less than 80 minutes on an Intel Quad Core Q9550 2.83Ghz CPU, the software ran on 8 threads. For every 160, 000 parameter combinations the 603 product name combinations are evaluated.

The algorithm can be configured to minimize a single error measure, for example the FP’s. This would rather lead to a relatively high number of FN’s. There is a typical trade-off that arises with the use of binary classifiers and this phenomenon is shown in Figure 7. This figure shows a scatter plot where the points represent a parameter combination, with the false positives on the x-axis and the false negatives on the y-axis. We can see that there is a trade-off between the FP’s and the FN’s. Our decision is to minimize equally both the FP’s and FN’s (both with the same weight). This is equivalent to choosing the combination which has the smallest distance to the origin. From the figure we can see that the algorithm performs well, as there are many points that are close to the origin.

Table 3: Results for the product name identification algorithm

Measure	Value
Accuracy	93.60%
Precision	92.11%
Recall	91.30%
Specificity	95.05%

Table 4: Confusion table for the product name identification algorithm

Predicted \ Actual	*Identical*	*Not Identical*
Identical	210	18
Not Identical	20	346

5.2. *Category Mapping*

For the evaluation of the category mapping algorithm, we have collected the product taxonomies of Amazon [37], ODP [38], and Overstock [39]. The collected taxonomies contain in total 2575, 44181, and 1052 categories respectively. For each taxonomy a random sample of 500 categories has been

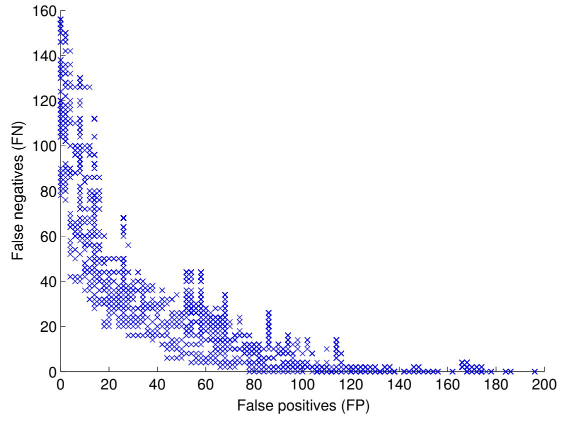


Figure 7: Product name identification: FP versus FN

drawn that is used for the evaluation of the mapping algorithms. The mapping algorithms are evaluated from this random source sample to a full target taxonomy, so for example $\text{Amazon}_{500} \rightarrow \text{ODP}_{\text{full}}$. This means that $6 \times 500 = 3000$ categories have been manually mapped for all 6 mappings. These manual mappings have been performed by three individuals that are not related to this project.

For each category in the random samples, each algorithm either finds a mapping or not. The way we compute the results is by comparing the mappings obtained by the algorithm with the manual mappings. True positives are the cases where the algorithm correctly provided a mapping and true negatives are the cases where the algorithm correctly identified that there is no suitable mapping. False positives are the cases where the algorithm mapped to a category while the correct action was either another category or no category at all (if no suitable mapping is found). False negatives are the cases where the algorithm incorrectly decided not to map the category. As with the evaluation of the product identification, we compute accuracy, precision, recall, and specificity for each algorithm and mapping combination. Further, we compute the F_1-measure, which is the harmonic mean of the precision and recall. The F_1-measure is widely used to represent the precision and recall using one value.

For each algorithm and each data set, we obtained an optimal set of thresholds by applying a hill-climbing procedure with an increment of 0.1. For Park & Kim, the optimal threshold was found to be quite low for each data set, being 0.1 for five mapping combinations and 0.2 for one mapping combination. This threshold defines the minimum value the similarity has to be in order for the algorithm actually to map the category. For PROMPT, we have found that the optimal parameters are a maximum path length of 2 and filtering the scores. For our approach, the optimal λ parameter is 0.8 and the mapping threshold is 0.3. The λ parameter is the cosine weight in the total similarity between two category names, as defined in Equation 6. The mapping threshold is again the minimum value the similarity has to be in order for the algorithm actually to map the category.

The results of the different category mapping algorithms are shown in Table 5 (XP stands for XploreProducts.com, our platform). We can see that our approach performs the best with respect to the F_1-measure and that the method of Park & Kim achieves the highest average precision. For recall, our approach also performs better than the approach of Park & Kim and PROMPT. In the evaluation of Park & Kim [28], the authors' results show that their method outperforms PROMPT with respect to recall. As one can see in the table, this is also confirmed for the data sets that we have used in this evaluation.

We have performed paired t-tests to test the significance of the differences in performance. Table 6 shows a subset of the alternative hypotheses and the corresponding p-values. The p-values that are below the 5% significance level are shown in bold. Alternative hypotheses that are not shown (e.g., PROMPT $>$ XP) are left out as all the p-values are above the significance level of 5%. From the table, we can conclude that Park & Kim have significantly better precision than our approach and PROMPT. More importantly however, for the F_1-

Table 5: An overview of the category mapping results for the Park & Kim (P&K) algorithm, PROMPT algorithm, and XploreProducts (XP) algorithm.

Data set	F_1-measure			Precision			Recall		
	P&K	PROMPT	XP	P&K	PROMPT	XP	P&K	PROMPT	XP
amazon → odp	0.191	0.053	0.250	0.241	0.077	0.160	0.159	0.040	0.580
amazon → overstock	0.238	0.191	0.389	0.602	0.356	0.292	0.148	0.131	0.580
odp → amazon	0.136	0.113	0.287	0.280	0.121	0.225	0.090	0.106	0.395
odp → overstock	0.269	0.129	0.270	0.253	0.155	0.225	0.288	0.110	0.335
overstock → amazon	0.180	0.288	0.471	0.525	0.420	0.331	0.109	0.219	0.814
overstock → odp	0.229	0.065	0.353	0.279	0.101	0.226	0.194	0.047	0.803
average	0.207	0.140	0.337	0.363	0.205	0.243	0.165	0.109	0.585

measure and the recall, we have significantly better results than Park & Kim and the authors of PROMPT.

Table 6: This table shows the result of the performed significance tests for the comparison of the category mapping algorithms. For this comparison, we have used one-tailed paired t-tests. In the first column the alternative hypothesis is shown and in columns two to four the p-values for the different measures are given.

Alternative hypothesis	F_1-measure	Precision	Recall
P&K > PROMPT	0.084	**0.000**	0.137
P&K > XP	0.988	**0.022**	0.997
XP > P&K	**0.012**	0.978	**0.003**
XP > PROMPT	**0.000**	0.175	**0.001**

One should note that although our results support the conclusions of Park & Kim, i.e., that their approach outperforms PROMPT, the performance with respect to the recall is found to be different. Park & Kim report better results with respect to recall on their data sets. This is probably due to the fact that we used more categories, in total 3000 categories, to perform the evaluation of the mapping algorithms. Also, the target product category taxonomies to which is mapped were also large (as already mentioned, ODP contains 44181 categories).

5.3. *User Interface*

In this section we discuss several use cases of our framework and we present how the user interface works. For this purpose, we created a tool which is used to automatically annotate Amazon.com and Circuitcity.com Web pages. We collected about 700 product Web pages from both Amazon.com and Circuitcity.com. After running the annotation tool and importing the annotated pages with the ping service, as described in Section 4, we have approximately 700 products in the demo database at our disposal.

During the design of the user interface we dealt with several issues. Our goal is to improve existing user interfaces by implementing several missing features of popular product search sites, such as Google Products, Shopping.com, and Shopzilla. This section will also highlight how we implemented these features in XploreProducts.com.

As already mentioned, the XploreProducts.com platform uses a faceted category navigation integrated with keyword search. For the evaluation, we have used the category taxonomy from Shopping.com, as it is concise and well-structured. The 327 categories were collected through the Shopping.com API, available at `http://developer.shopping.com`. The user starts navigating by entering one or more keywords. After the keywords are entered, the facets and initial results are displayed. A result page that is presented to the user, is shown in Figure 6. Note that prices shown on the page are minimum prices for each product. The numeric facets are based on numeric ranges of the retrieved results. The non-numeric facets can be ordered or non-ordered, the ordered facets can be split in hierarchical and non-hierarchical facets.

5.3.1. *Searching and browsing*

In order to explain the facets, and understand how they work, we give example queries and explain how our user interface is employed by the users. We start by considering the facet 'Category', which is a hierarchical facet, and that the user queries for 'office'. The search results show 5 categories with two root nodes, 'Computers' and 'Home and Garden'. When a user clicks on a category, the result screen immediately refreshes and only the category itself, its parent, and its ancestors are shown. A green marker indicates which categories are selected. Most product search Web sites do not support the selection of multiple product categories. On XploreProducts.com, this is allowed as one might want to select, for instance, multiple brands or product categories at the same time (which implies a disjunctive query).

The same functionality as described in the previous paragraph holds for the brand, multiple brands can be selected. This means that when a user selects a brand, he or she has the possibility to drill down (by filtering on other facets) or to broaden the search results (by selecting another brand). With most product search Web sites, for example with Google Products, this is not possible. Searching for 'gsm' on Google Products provides several brands. If we select one of them, e.g., 'Nokia', then the screen refreshes and the user can only choose 'Any brand' as an option. The other brand options are then made invisible.

Moving on to the next three facets, the numeric facets, we implemented a similar behavior for all of them. Each facet consists of 4 ranges which are recomputed each time the query changes. This is done in order to redistribute the values evenly across the different ranges. When there are no products for a certain range, for example a price range, then the price range is not selectable, as this would give no results. Last, we have the facet for the stores. This facet behaves in the same manner as the brands facet. The user can use this facet as an overview of how many products per store are returned in the results. If the user selects a store, only results from that store will be shown.

We also have evaluated the response time of XploreProducts.com. For a test sample with 100 queries, the average response time was 0.56 seconds. The performance is however very dependent on the machine the Web application is running. These statistics are for a dedicated server, with 16 GB of RAM and an Intel Xeon X3440 2.53GHz processor.

6. Conclusions and Future Work

In this paper we provided a solution for the issues that arise in Web-wide information aggregation and parametric product search. We have argued that the current situation is not beneficial for neither consumers nor producers, as the consumers focus primarily on the price and thus, a fierce price competition arises, which causes the margins for producers to diminish. Web-wide product search can help consumers to focus more on the properties of products instead of only the price, by providing a fast and efficient way to find the product a user is looking for. It also makes it more easy for small companies to survive as they can pursuit a specialized business model.

For our solution we have three main components: product name identification, category mapping, and multi-faceted search interface. For the first two components, product name identification and category mapping, we have devised our own algorithms.

For the product name identification algorithm the precision, accuracy, recall, and specificity are all above 91% for a data set with 603 product names. This algorithm is important for Web-wide product search as this is the key factor for the success of our information aggregation method. The algorithm deals with several issues, such as homonyms, synonyms, and product names of different lengths.

In the evaluation of the category mapping algorithm we have used a significantly larger data set than currently presented in related literature. In total we have obtained 3000 manual mappings spread across 6 mapping combinations. The results of this paper show that our algorithm, at a significance level of

5%, significantly outperforms two existing category mapping algorithms with respect to the F_1-measure and recall, i.e., the algorithm of Park & Kim [28] and PROMPT [29]. We confirm the results of Park & Kim, which show that their algorithm significantly performs better than PROMPT with respect to the F_1 measure, but we find that the difference in recall between their method and PROMPT is not significant at a significance level of 5%. Finally,we found that the method of Park & Kim significantly (5% significance level) outperforms our approach and that of PROMPT with respect to precision.

For the purpose of demonstration, we implemented a prototype of our framework, which uses a multifaceted category navigation integrated with keyword search interface. For merchants it is possible to ping their product Web pages. This has the advantage that the merchants do not need to make a data feed for each product search platform, they just have to annotate their current product Web pages. The users benefit from a flexible user interface, which has several features that other product search Web sites do not have, such as multiple category selection, multiple brand and store selection, automatic range calculation, and currency converting.

As future work, we propose to use more advanced ontologies to represent the product information, like the eClassOWL [16] or GoodRelations [10] ontologies. This will lead to more facets as users can provide more specific product information, like the battery life of a laptop, stock information, or the screen size of a mobile phone. With the currently used vocabulary, only general product properties as the 'product category' or the 'brand' can be used.

Furthermore, in our product search prototype, we use Shopping.com's existing product category hierarchy. Future work might be to deduce a product category hierarchy from the product information, e.g., the categories, the product descriptions, etc. Besides using product descriptions for deducing category hierarchies, these descriptions could also help to improve the results for the product name identification algorithm as descriptions also contain valuable product information.

While we focused on using a textual based faceted search shopping interface, in the future we would like to investigate how we can use graphical attribute-based shopping interfaces [40]. These interfaces would allow for a better overview of the results and product attributes.

References

[1] B. J. Corbitt, T. Thanasankit, H. Yi, Trust and e-commerce: a study of consumer perceptions, Electronic Commerce Research and Applications 2 (3) (2003) 203–215.

[2] E. Hovy, Combining and standardizing large-scale, practical ontologies for machine translation and other uses, in: Proceedings of the 1st Intelligent Conference on Language Resources and Evaluation (LREC 1998), 1998, pp. 535–542.

[3] K. Goel, R. Guha, O. Hansson, Introducing Rich Snippets, `http://googlewebmastercentral.blogspot.com/2009/05/introducing-rich-snippets.html` (2009).

[4] R. Guha, R. McCool, E. Miller, Semantic search, in: Proceedings of the 12th International Conference on World Wide Web (WWW 03), ACM Press, 2003, pp. 700–709.

[5] J. English, M. Hearst, R. Sinha, K. Swearingen, K.-P. Yee, Hierarchical faceted metadata in site search interfaces, in: Proceedings of the Conference on Human factors in Computing Systems (CHI 2002), ACM Press, 2002, pp. 628–639.

[6] M. A. Hearst, Search User Interfaces, Cambridge University Press, 2009.

[7] M. Hearst, A. Elliott, J. English, R. Sinha, K. Swearingen, K. Yee, Finding the flow in web site search, Communications of the ACM 45 (9) (2002) 42–49.

[8] S. Pemberton, B. Adida, S. McCarron, M. Birbeck, RDFa in XHTML: Syntax and processing, W3C recommendation, W3C, `http://www.w3.org/TR/2008/REC-rdfa-syntax-20081014` (Oct. 2008).

[9] D. Beckett, RDF syntax specification (revised), W3C recommendation, W3C, `http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/` (Feb. 2004).

[10] M. Hepp, Goodrelations: An ontology for describing products and services offers on the web, in: Proceedings of the 16th International Conference, Knowledge Engineering and Knowledge Management Conference (EKAW 2008), Vol. 5268 of LNCS, Springer, 2008, pp. 329–346.

[11] E. Prud'hommeaux, A. Seaborne, SPARQL query language for RDF, W3C recommendation, W3C, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/ (Jan. 2008).

[12] L. Obrst, R. E. Wray, H. Liu, Ontological engineering for B2B e-commerce, in: Proceedings of the international conference on Formal Ontology in Information Systems (FOIS 2001), ACM Press, 2001, pp. 117–126.

[13] D. Fensel, D. L. McGuinness, E. Schulten, W. K. Ng, E.-P. Lim, G. Yan, Ontologies and electronic commerce, IEEE Intelligent Systems 16 (1) (2001) 8–14.

[14] O. Corcho, A. Gómez-Pérez, Solving integration problems of e-commerce standards and initiatives through ontological mappings, International Journal of Intelligent Systems 16 (16) (2001) 2001.

[15] M. Hepp, J. Leukel, V. Schmitz, A quantitative analysis of eclass, unspsc, eotd, and rntd: Content, coverage, and maintenance, in: Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE 2005), IEEE Computer Society, 2005, pp. 572–581.

[16] M. Hepp, eClassOWL, http://www.heppnetz.de/projects/eclassowl (2009).

[17] M. Hepp, unspscOWL, http://www.heppnetz.de/projects/unspscowl (2010).

[18] J. C. Team, Joomla, http://www.joomla.org (2009).

[19] H. P. de Leon, osCommerce, http://www.oscommerce.com (2000).

[20] D. Buytaert, Drupal, http://drupal.org (2001).

[21] W. Kim, D. W. Choi, S. Park, Intelligent product information search framework based on the semantic web, in: Demo papers for the 3rd International Semantic Web Conference (ISWC 2004), 2004.

[22] A. F. Cuadrado, E. V. de la Torre, SIS: Semantic intelligent search engine from heterogeneous information sources applied to e-commerce, in: GI Jahrestagung (2), GI, 2008, pp. 700–705.

[23] D. Schuster, M. Walther, I. Braun, Towards federated consumer product search from heterogeneous sources, in: Proceedings of IADIS International Conference WWW/Internet, 2008, pp. 453–456.

[24] M. Walther, D. Schuster, A. Schill, Federated product search with information enrichment using heterogeneous sources, in: Proceedings of the 12th International Conference on Business Information Systems (BIS 2009), Vol. 21 of Lecture Notes in Business Information Processing, Springer, 2009, pp. 73–84.

[25] S. Colucci, T. Di Noia, A. Pinto, M. Ruta, A. Ragone, E. Tinelli, A non-monotonic approach to semantic matchmaking and request refinement in e-marketplaces, International Journal of Electronic Commerce 12 (2) (2007) 127–154.

[26] L. Zhang, M. Zhu, W. Huang, A framework for an ontology-based e-commerce product information retrieval system, Journal of Computers 4 (6) (2009) 436–443.

[27] E. Schulten, H. Akkermans, G. Botquin, M. Dorr, N. Guarino, N. Lopes, N. Sadeh, Call for participants: The e-commerce product classification challenge, IEEE Intelligent Systems 16 (4) (2001) 86–c3.

[28] S. Park, W. Kim, Ontology mapping between heterogeneous product taxonomies in an electronic commerce environment, International Journal of Electronic Commerce 12 (2) (2007) 69–87.

[29] N. Noy, M. Musen, The PROMPT suite: interactive tools for ontology merging and mapping, International Journal of Human-Computer Studies 59 (6) (2003) 983–1024.

[30] S. Kaza, H. Chen, Evaluating ontology mapping techniques: An experiment in public safety information sharing, Decision Support Systems 45 (4) (2008) 714–728.

[31] X. Su, J. A. Gulla, An information retrieval approach to ontology mapping, Data & Knowledge Engineering 58 (1) (2006) 47 – 69.

[32] J. Tang, J. Li, B. Liang, X. Huang, Y. Li, K. Wang, Using Bayesian decision for ontology mapping, Journal of Web Semantics 4 (4) (2006) 243–262.

[33] B. Marshall, H. Chen, T. Madhusudan, Matching knowledge elements in concept maps using a similarity flooding algorithm, Decision Support Systems 42 (3) (2006) 1290 – 1306.

[34] S. Melnik, H. Garcia-Molina, E. Rahm, Similarity flooding: A versatile graph matching algorithm and its application to schema matching, in: Proceedings of the 18th International Conference on Data Engineering (ICDE 2002), IEEE, 2002, pp. 117–128.

[35] V. I. Levenshtein, Binary Codes Capable of Correction Deletions, Insertions, and Reversals, Soviet Physics Doklady 10 (8) (1966) 707–710.

[36] J. Mehta, IP-to-Country Database and Tools, http://ip-to-country.webhosting.info (2003).

[37] Amazon.com, Inc, http://www.amazon.com (2011).

[38] Open Directory Project, http://www.dmoz.org (2005).

[39] Overstock shopping, http://http://www.overstock.com (2011).

[40] M. Kagie, M. van Wezel, P. Groenen, A graphical shopping interface based on product attributes, Decision Support Systems 46 (1) (2008) 265–276.