FLOPPIES: A Framework for Large-Scale Ontology Population of Product Information from Tabular Data in E-commerce Stores

Lennart J. Nederstigt, Steven S. Aanen, Damir Vandic, Flavius Frasincar

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

Abstract

With the vast amount of information available on the Web, there is an urgent need to structure Web data in order to make it available to both users and machines. E-commerce is one of the areas in which growing data congestion on the Web impedes data accessibility. This paper proposes FLOPPIES, a framework capable of semi-automatic ontology population of tabular product information from Web stores. By formalizing product information in an ontology, better product comparison or parametric search applications can be built, using the semantics of product attributes and their corresponding values. The framework employs both lexical and pattern matching for classifying products, mapping properties, and instantiating values. It is shown that the performance on instantiating TVs and MP3 players from Best Buy and Newegg.com looks promising, achieving an F_1-measure of approximately 77%.

Keywords: product, ontology, population, instantiation, e-commerce, Semantic Web

1. Introduction

A few decades ago, it was hard to imagine the enormous impact the Web would have on our daily lives these days. However, with the vast amount of information available, still doubling in size roughly every five years [1], there is a serious need to structure all the Web data in order to keep it findable. With this aim in mind, the Semantic Web [2] was conceived in 2001. In the past years, some developments based on the ideas of the Semantic Web have been adopted for large-scale use. One of these is the introduction of a semantic vocabulary called schema.org [3], proposed by the four major search engines Bing, Google, Yahoo!, and Yandex. Schema.org is a very broad vocabulary with which the search engines aim to have a high-level shared vocabulary that focuses on popular Web concepts. This means that it is by no means an effort to have an ontology of 'everything' or an ontology that is very specialized in one domain. Furthermore, Google introduced recently the Knowledge Graph [4], which is a project that augments search results with appropriate semantic metadata. Despite these recent movements, which are often attributed to the concept of 'Linked Data' [5], the initially envisioned Semantic Web is still at its infancy.

One of the areas in which growing data congestion on the Web has serious consequences, is the field of e-commerce [6]. Today's search engines are still primarily keyword-based, fail to work with syntactical differences, and are language-dependent. Web-wide parametric product search is unavailable, making it difficult for users to find the optimal purchase for their needs. According to existing research [7], a large fraction of online shoppers get confused or are overwhelmed by the information they

Email addresses: lennart@student.eur.nl (Lennart J. Nederstigt), steve@student.eur.nl (Steven S. Aanen), vandic@ese.eur.nl (Damir Vandic), frasincar@ese.eur.nl (Flavius Frasincar)

get presented while searching for products. The result can be that prices become the determining factor for purchases on the Web. This situation is not optimal for both buyers and sellers: the buyers could be better off with a more expensive product if that would fit better to their needs, whereas the sellers might want to be competitive on other characteristics than pricing alone [8].

This research focuses on the extraction of product information from tabular data sources on the Web, such as product information pages. Many major e-commerce shops use, in one way or another, a tabular format for the product specifications. This especially holds for complex (technical) products. For example, Amazon, BestBuy.com, Walmart, and Shopping.com, which are 4 well-known e-commerce sites, all represent product information in a tabular format.

In order to extract product information from tabular data, we propose *FLOPPIES: a Framework for Large-scale Ontology Population of Product Information in E-commerce Stores*. FLOPPIES is a semi-automatic approach, aided by user-defined ontology annotations in the form of lexical representations and regular expressions. Using the tabular data often available on Web store product pages, which conveys the factual information about a product, the ontology-driven framework creates a structured knowledge base of product information. In order to achieve this goal, FLOPPIES employs user-defined annotations for lexical and pattern matching, which facilitates product classification, property association, and value extraction. Our knowledge base, the proposed OWL [9] ontology *OntoProduct*, defines specific properties and characteristics for 24 consumer electronic product classes. Figure 1 provides an overview of the input and output of the framework, based on our product ontology.

The proposed *OntoProduct* ontology is mapped to the GoodRelations ontology for e-commerce [10], which is a more formal ontology than the schema.org vocabulary and is developed and maintained by Martin Hepp since 2002. It is a highly standardized vocabulary that not only can describe product data, but also company data and product offerings. This ontology aims to specify all aspects that come into play in the domain of e-commerce. For example, it supports statements to depict time frames for which an offering is valid. Fortunately, in 2012, the schema.org team announced that GoodRelations has been integrated in their vocabulary, which means that schema.org can now be used to describe more granular product information [3]. Although GoodRelations defines concepts that can be used to describe product classes, i.e., their hierarchy and the associated product properties, the actual product classes, such as 'Phone' or 'Television', are not defined. This is one of the reasons why we propose the *OntoProduct* ontology and a system that can semi-automatically extract instances from unstructured product information.

When product information is formalized in an ontology, better product comparison or recommendation applications can be built, employing more intelligent parametric search by exploiting the semantics of product attributes and their corresponding values. Furthermore, there will be no need for existing Web stores to provide their data in a specific format (which is currently the case), as search engines will be able to effectively 'pull the information' from the Web stores themselves by consuming the annotated product information on the Web pages. Information could be more easily aggregated in order to have a very extensive source of product information. A prototype that utilizes Semantic Web technology to aggregate product information from multiple sources, as a means to improve product comparison, has been implemented in earlier research [11].

The formalization of product information has several advantages in practice for both business and consumers. For example, solving the information heterogeneity problem in e-commerce can lead to serious improvements in the business information exchange [12]. Furthermore, the consumers' product retrieval capabilities will increase because of the more intelligent product search engines. For exam-

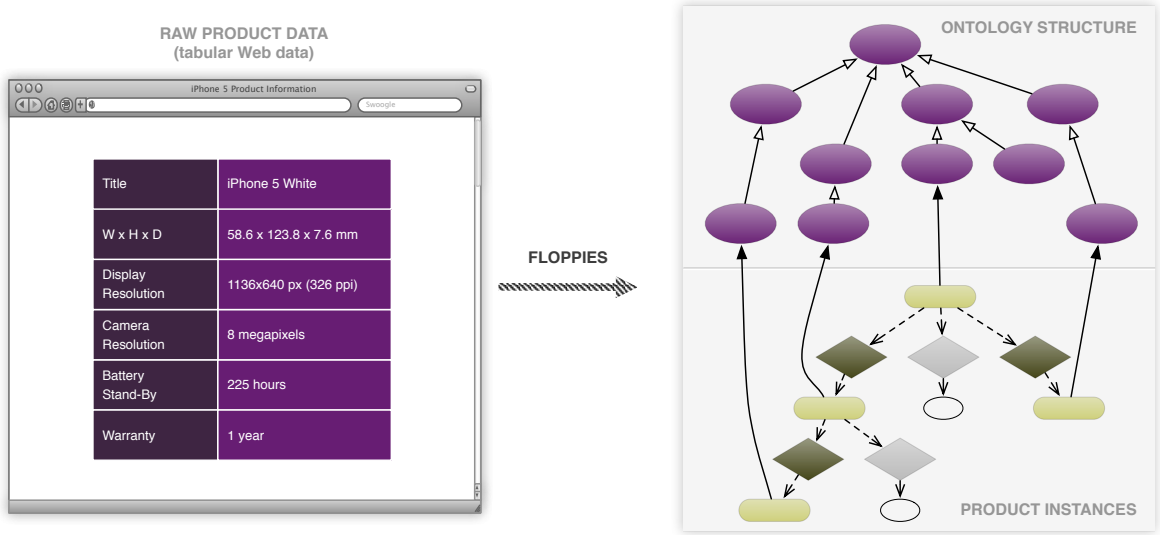

Figure 1: Overview of the input and output of the proposed framework. The tabular Web product data on the left (input) is transformed into the product instances part on the right (output), using the given ontology structure.

ple, search engines will be able to better rank products because they can reason about how values of a product attribute relate to one another. This is best illustrated with an example. Consider the facts that 'HSPDA' is faster than '3G' and that '3G' is faster than 'GPRS'. From these facts, a semantic search engine can deduce that 'HSPDA' is faster than 'GPRS' if the property 'faster than' is declared to be transitive. This reasoning can help in cases where fuzzy search is needed, i.e., when a user is searching for a phone with 'HSPDA' but none actually exist with the current selection of properties and the next best phone has to be displayed. FLOPPIES supports these developments by providing a semi-automatic method to store actual facts about a product (i.e., the values of its attributes) in a product ontology. As a result, one has access to a knowledge base that is understandable for both humans and machines.

This paper is organized as following. First, related research approaches are discussed in Section 2. Then, Section 3 explains the proposed framework in detail. Section 4 evaluates the performance of FLOPPIES in a component-wise analysis and compares it with a baseline approach. Last, conclusions and future work directions are given in Section 5.

2. Related Work

In this section, we discuss some similar research approaches for ontology population that are applicable in the e-commerce field. Furthermore, some existing product ontologies are reviewed, as such an ontology is required for instantiation in our problem context. The scope of this research is the ontology population itself, and not HTML table extraction. Therefore, approaches focusing on this topic are not discussed in this paper.

2.1. Ontology Population Approaches

Due to the wealth of information that is now available, both on the Web and within organizations, it would be impractical to manually instantiate all that information in an ontology. Therefore, several semi-automatic ontology population approaches have been conceived in recent years, which are also applicable to the e-commerce domain.

Holzinger et al. [13] propose a fully autonomous process, which only needs some initial seed knowledge about

a specific product domain. Using this knowledge, it performs a knowledge extraction process on Web pages, which retrieves tabular data that is relevant to the product domain from the Web page. However, instead of populating an ontology with the extracted information using an external framework that contains custom logic, they propose a more integrated approach in which parts of the required logic are replaced by OWL reasoning in the ontology. Once the tabular data has been extracted from the Web page, content spotters are employed, which detect specific values through regular expressions and are able to annotate this information with OWL statements. Afterwards, additional facts can be derived from the annotated tabular data using the domain-specific ontology that was given as the seed knowledge for the process. The authors argue that this provides a more modular and transparent system in which logical tables and domain models can be easily substituted.

A different approach, using the semantic lexicon WordNet [14], is proposed by Patel et al. [15]. OntoGenie is a semi-automatic tool that takes domain ontologies and unstructured data, often in the form of natural language, as input. It first maps the concepts in a domain ontology to a WordNet equivalent. Then it captures the terms occurring in Web pages and tries to map each word to a WordNet concept. Finally, the relationships between the domain ontology concepts and the words on the Web pages can be determined by examining their mappings to WordNet. It employs the principle of locality to compute the distance between concepts, using information discovered from other pages, for increasing the recall.

Ontosophie [16] is a strictly semi-automatic system for ontology population, requiring user input for each information extraction cycle. It consists of a natural language processor, which uses shallow syntactical parsing for annotating terms in sentences. The next process is deriving a set of extraction rules from the annotated documents. A conceptual dictionary induction system is employed for this phase, which uses a training corpus to derive a dictionary of concept nodes. The extraction rules are generated using the different combinations of concept nodes occurring in the sentences of the training corpus. However, as not every extraction rule might be correct or specific enough, Ontosophie also computes a rule confidence factor for each extraction rule, using K-fold cross-validation. During this process, it merges rules giving identical results and assigns each rule a confidence factor. After reviewing all the generated extraction rules, the extraction rules with a sufficient rule confidence factor are used to populate an ontology. The authors argue that it is important for the user to maintain control of the process, while only presenting suggestions that the process considers to be correct. Therefore, Ontosophie comes up with instantiation suggestions and ultimately lets the user decide on whether it should instantiate the information or not. Furthermore, it employs configurable thresholds for setting the desired minimum confidence factor for making the suggestions.

OntoSyphon [17] is an unsupervised ontology population system, which takes any ontology as input and uses it to specify Web searches. Using both the ontology and the additional information obtained from the Web, it is able to automatically identify entities and their relations. The advantage of this approach is that the entire Web can be used as a corpus for instantiating entities in the ontology.

Our approach differs from these approaches on several aspects. First, with the exception of [13], the aforementioned methods are not specifically targeted at populating an ontology with (tabular) product information gathered from the Web. Second, the other methods generally rely on natural language processing, using the syntactical context of a text to derive facts, while our approach focuses on tabular data. Last, even though the framework we propose shows some resemblance with the approach in [13], as both use regular expressions and lexical representations for entity annotation, there is on important difference. Unlike other approaches, including the approach in [13], our ap-

proach employs a GoodRelations-based ontology for annotating instances, making it compatible with major search engines (GoodRelations is already supported by some of the major search engines). The approaches that are discussed in this section do not share this advantage.

Even though most other methods are not directly applicable to the discussed problem, we can, nevertheless, reuse some of their elements. For instance, the classification of products can be achieved by mapping the category hierarchy of a Web store, if it is available, to product classes in the ontology. It could use a similar approach as [15], to create the mappings by employing WordNet.

In addition, the proposed value instantiation process, as used by the framework, employs a set of different value extraction rules capable of converting key-value pairs to the proper format for instantiating the ontology. Unfortunately, as there is no freely available implementation of a relevant ontology population framework, and not enough information to precisely recreate an existing framework, we cannot compare the performance of our proposed framework to that of the aforementioned frameworks.

2.2. Ontologies for E-commerce

Ontologies have been successfully applied in various domains, ranging from mechanical engineering [18, 19, 20] to biology [21, 22, 23]. Also, various ontologies and categorization standards have been proposed for usage in the e-commerce domain. These can help businesses in a variety of ways, for example by improving communication possibilities between companies, and by automating various processes such as stock management.

A commonly used classification system for products is the *United Nations Standard Products and Services Code* (UNSPSC) [24]. Though UNSPSC is not freely available, it is applied broadly as it covers a very wide range of products. The UNSPSC dataset has also been converted into an OWL [9] ontology for research use, though it is questionable whether the purely hierarchical data structure of UNSPSC benefits from such a conversion [25]. Similar to UNSPSC, *eCl@ss* [26] provides a wide base of product classes and descriptions. It is also a commercial standard, competing with UNSPSC, though more successful in Germany and containing properties per product class as well. For eCl@ss, an OWL conversion project is also maintained for research purposes [27, 28]. A third categorization standard worth mentioning is the *Open Directory Project (ODP)* [29], which is a project aiming to categorize the Web. Its shopping division consists of roughly 44,000 categories, but the classes have no further information attached to them.

In the e-commerce domain, another project, *RosettaNet* [30], is a non-profit standard for sharing business to business information. It is based on XML [31], and is mostly used in the supply chain area. These and other general e-commerce categorization standards are evaluated and discussed in a survey by Hepp et al. [32].

Moving on to projects more related to Semantic Web, *GoodRelations* [10] is a high-potential ontology describing products and service offerings for e-commerce. It has been adopted by various large Web stores in the form of RDFa [33] annotations. Furthermore, by mapping it to the schema.org vocabulary, the project is increasingly gaining attention from major search engines, which offer support for a growing set of annotations from GoodRelations. However, GoodRelations only specifies the way in which products and services need to be described, and does not contain product-specific properties or product classes.

In an attempt to augment GoodRelations with product classes and properties for consumer electronics, the *Consumer Electronics Ontology* (CEO) [34] has been developed. Although this ontology includes a subclass-of relationship between the product entities, product attribute information is not available.

There are some other approaches that are related to the product ontology that we propose. One of them is the productontology.org project [35]. This project pub-

lishes an ontology that extends the GoodRelations ontology with approximately 300,000 product and service definitions, based on the automatic extraction of these from Wikipedia. It contains some basic properties that are mapped to GoodRelations. However, these properties are very general and apply to many products. There are not many properties that are product specific, such as 'maximum load' for washing machines and 'cpu speed' for laptops. Furthermore, existing ontologies miss the metadata needed for appropriate extraction of properties and their values from text.

There are also other efforts that do not directly rely on Wikipedia for the schema creation [36, 37]. Although the ontologies that are proposed in these projects contain more detailed schemas, they fail to address the issue of formal semantics with respect to the unit of measurements. Our proposed OntoProduct ontology does address this aspect by integrating an existing ontology-driven on units of measurement.

3. FLOPPIES Framework

In this section, we provide a detailed explanation of the FLOPPIES framework. First, the general processes for the framework are discussed in an overview. Then we elaborate on the ontology that is used for instantiation. Last, each step of the framework is explained in more detail.

3.1. Framework Overview

The goal of the FLOPPIES framework is to structure consumer product information from Web sites in order to improve product search or recommendation systems. To achieve this goal, several steps are required: extraction of key-value pairs from (tabular) Web data; instantiation of the product data into an ontology; product entity resolution to detect and aggregate duplicate products from different Web sources and; an application that uses the instantiated ontology data for product search or recommendation. A lot of research effort has already been invested in extraction of (tabular) Web site data [38], and in (product) duplicate detection [39, 40]. Therefore, these steps are left outside the scope of this research.

The FLOPPIES framework starts with the assumption that product information in the form of key-value pairs is present. Collecting this data is often trivial, as many Web stores already offer product information in tabular form, ordered as key-value pairs. FLOPPIES uses this *raw product data*, as we refer to it, for instantiating the individual products and their features into a predefined product ontology. This domain ontology has to be able to describe individual products with specific properties for each type of product. For instance, a TV shares some properties with digital audio players (i.e., 'screen size'), but it also has properties that digital audio players do not possess (i.e., 'remote control'). Although significant effort has been put into establishing ontologies on the Web, a domain-specific ontology for products with the required amount of specificity does not yet exist. Therefore, we introduce the *OntoProduct* ontology, which will be explained in more detail in the next subsection.

Figure 2 provides an overview of the FLOPPIES framework. It starts with the raw product data, the key-value pairs describing a product, as input. The final output of the framework is an OntoProduct ontology file, instantiated with the product and its features.

Between input and output, we identify three main processes, as shown in the diagram. First, it is necessary to obtain the type of product that is being instantiated: the *Classification* process. The classes are predefined in the ontology and determine the possible properties of the product. Most Web stores nowadays have some kind of product class or category data of each product available. Therefore, the Classification process in the FLOPPIES framework is seen as optional, in case the class data is not available.

The second framework process is called *Property Matching*. This step is used to create a link between the key-value pair keys from the raw product data, and the prop-

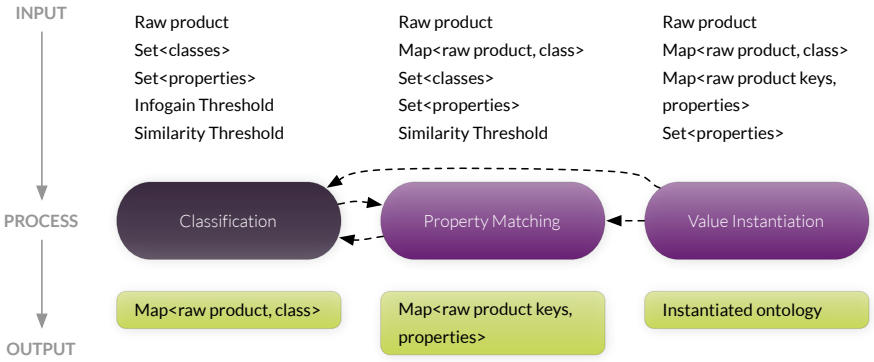

Figure 2: Overview of the processes in the proposed framework. Dashed lines indicate a usage relationship. Classification is only used when no class data is available.

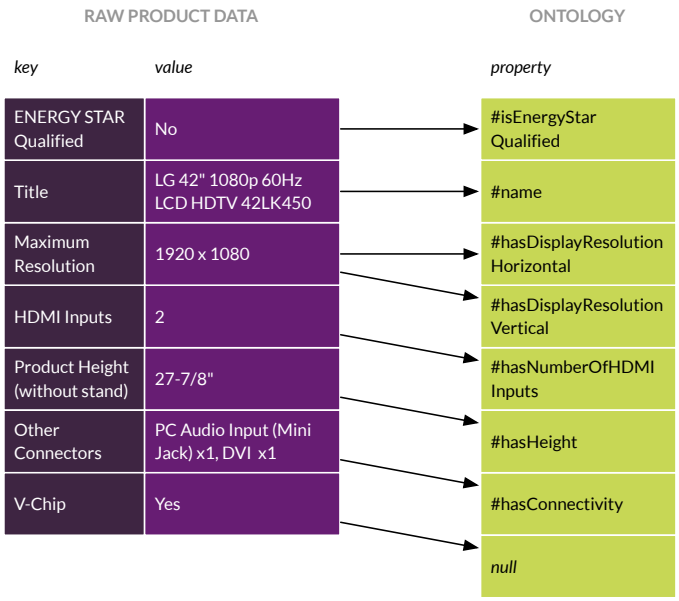

Figure 3: Example of property matching between (real-world) raw product data and ontology properties from OntoProduct.

erties from the ontology. This is dependent on the product class, as the class determines the possible ontology properties that can be linked. Figure 3 indicates more clearly what Property Matching is about. Note that, as the figure indicates with the mapping of 'Maximum Resolution', one raw product key can be mapped to multiple ontology properties. This is required as some raw product key-value pairs combine multiple characteristics of a product, which are separately stored in the ontology.

The third and last process in the FLOPPIES framework is that of *Value Instantiation*. This part uses the class obtained from Classification, or directly from input data if the class is available, together with the result of Property Matching to instantiate the values in the ontology. Value Instantiation is very much about content spotting, parsing, and creating property assertions in the ontology. After the Value Instantiation, the raw product information from the Web has been structured and semantically annotated using an ontology. From that point on, applications can use the data to improve for example product search or facilitate product recommendation.

3.2. The OntoProduct Ontology

As Section 2.2 discussed, there have been various attempts to create product ontologies. However, unfortunately none of them are freely available and are both broad and specific enough to describe products in the domain this research uses: consumer electronics. Therefore, we introduce a new OWL [9] product ontology for consumer electronics, which builds on the foundations of existing work: *OntoProduct*. It was conceived by four domain experts, who used training data originating from existing Web stores to create the ontology structure.

3.2.1. Dependencies of OntoProduct

OntoProduct is fully compatible with GoodRelations, known as 'The Web Vocabulary for E-commerce' [10]. However, GoodRelations is a high level ontology, which misses the specificity that is required to describe product features in detail. Another project led by GoodRelations' creator Martin Hepp, the Consumer Electronics Ontology (CEO) [34], attempts to extend GoodRelations with specific properties and product classes for better description possibilities of products. Although CEO provides a fruitful extension to GoodRelations, it only defines product properties and some product classes, but not the links between these. OntoProduct, nevertheless, uses CEO as a base, and extends it with new properties, product classes, and relations between these. In total, OntoProduct contains 24 product classes and 270 distinct product properties from the consumer electronics domain, which allows for the instantiation of product information with sufficient detail.

In e-commerce, many product features are quantitative and use a unit of measurement. For example, the weight of a product can be given in pound or in kilogram, resulting in a very different meaning. To cope with this problem, OntoProduct requires a unit of measurement to be linked to quantitative values. Although GoodRelations does not include a standard list of units of measurement, nor a way to define for example the used notations, we were able to extend it with another ontology that does enable to do this: the Units of Measurement Ontology (MUO) [41]. MUO provides the ontology structure for working with units of measurement, but does not yet contain the instances. For the instances, OntoProduct uses the Unified Code for Units of Measure code system (UCUM) [42]. The authors of MUO have made the dataset available to use UCUM in conjunction with the MUO ontology.

3.2.2. OntoProduct Structure

Figure 4 gives an example of an instantiation in the OntoProduct ontology. Any instantiated product individual, such as `op:LG-47LV` in this example, is member of a product class, in this case `ceo:TV`. This product class determines which properties are valid for the type of product that is being instantiated. In general, we identify three important property types: *quantitative object properties* (i.e. `ceo:hasWidth`), *qualitative object properties* (i.e. `ceo:hasDataFormat`), and *data properties* (i.e. `ceo:hasTouchscreen`). OntoProduct contains 57 qualitative object properties (with 783 qualitative individuals), 151 quantitative object properties, and 62 data properties. The domain of these properties entails one or more product classes, to define which characteristics a product can have. The range of the properties depends on the type: object properties have a range of respectively quantitative and qualitative values, whilst data properties point to data types. In the case of qualitative values, the range also determines the possible units of measurement that can be attached to some property value.

3.2.3. OntoProduct Metadata

As Section 3.1 mentioned before, FLOPPIES is a semi-automatic framework for product ontology instantiation. The reason we do not present it as being automatic, is because the algorithms largely depend on ontology annotations for linking product properties to raw product keys, and for parsing values from the raw product data. In practice, this means that for new data sources (i.e., a new Web store), the ontology needs to be annotated with appropriate metadata. For example, one Web store might specify the property of diagonal display size as 'Display size' while another uses 'LCD screen size'. Moreover, the lexical representations in the ontology can be used to enable processing for data with differing denominations or even from different languages. In OntoProduct Metadata, which is an extension to OntoProduct used purely for the purpose of assisting the ontology instantiation with human input, the lexical representations can be applied to all properties and qualitative value individuals.

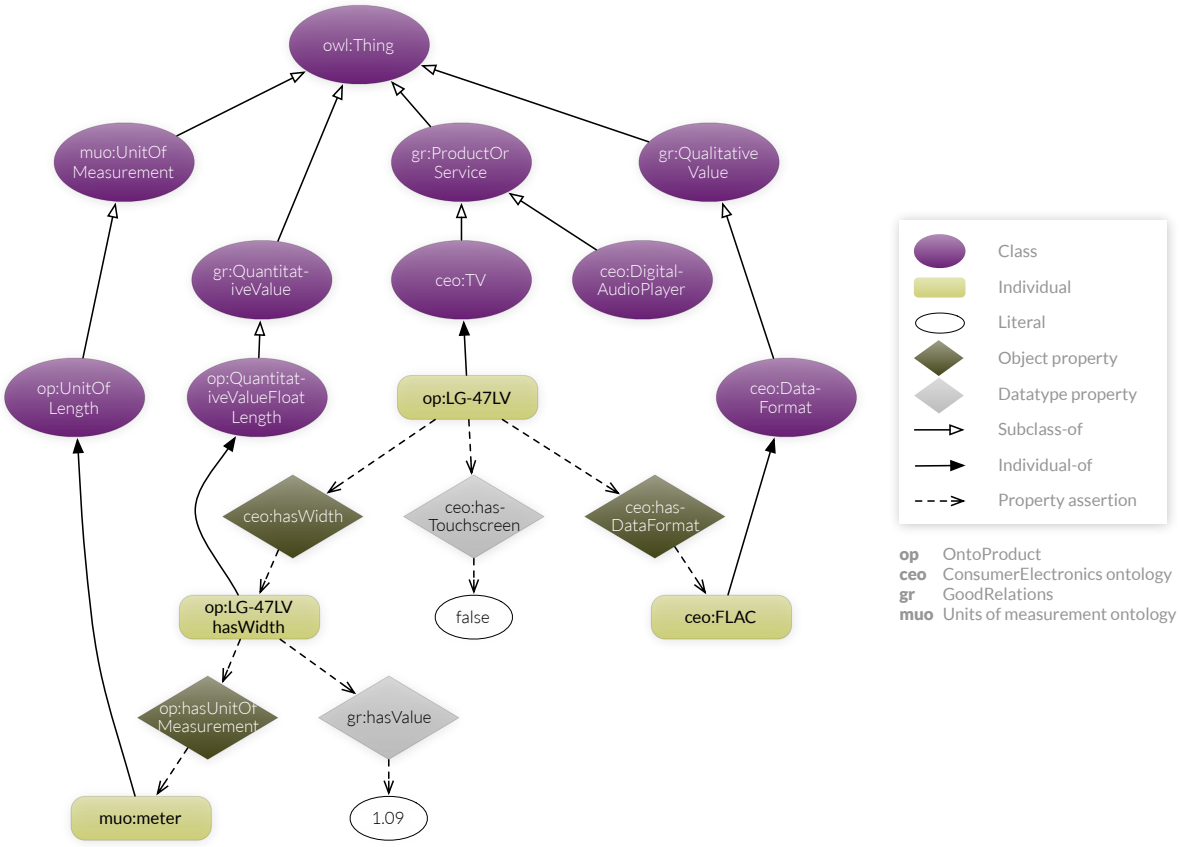

Figure 4: Example of an instantiated TV in the OntoProduct ontology.

Next to lexical representations, OntoProduct Metadata can be used to annotate quantitative object properties and data properties with regular expressions [43]. Regular expressions provide a pattern to which the raw product values should match for a certain property. This is used for Property Matching to filter out possible faulty mappings. In addition, regular expressions are used in the Value Instantiation process to parse numeric data from the raw product values, by means of grouping. Grouping is commonly used in regular expressions to select certain parts of a matching region in the input value. For instance, consider the key-value pair `['Refresh Rate','60Hz']`, which can be mapped to the ontology property `op:hasScreenRefreshRate`. A screen refresh rate needs to have a unit of measurement for the frequency, commonly measured in Hertz (Hz), therefore we annotate the property with the following regular expression: `(\d+)\s?(?:Hz|Hertz)`. A regular expression searches the raw product value for a region which corresponds to the specified pattern, in this case a numerical value followed by either 'Hz' or 'Hertz'. If the search succeeds, it stores the numerical value in a separate group, which can be retrieved by the Value Instantiation process to instantiate the numerical value with the property `gr:hasValue`.

As another example of the flexibility offered by regular expressions, take key-value pair `['Dimensions','55.3" x 33.2" x 1.3"']`. Since there is no property to specify 'dimensions' in the ontology, it is required to break up the raw product value into multiple instantiations. Using lexical representations, the user could annotate ontology property `ceo:hasWidth` with 'Dimensions' for improved property matching. Adding a regular expression would enable the Value Instantiator to detect a match with value '`55.3" x 33.2" x 1.3"`', and select the first number, 55.3, from

it through grouping. Similarly, the height and depth can be annotated for improved matching and parsing.

Annotation of properties is one of the key reasons why FLOPPIES is successful in instantiation, as we shall see. The user can help the computer by specifying recognition patterns in the form of regular expressions, and lexical representations, after which the computer can automatically instantiate most of the products with their various characteristics. For practical use, one could consider building a (Web) application to make the annotation easier for the end-user, for example by pre-collecting lexical representations from raw product data which the user can select for addition to the OntoProduct Metadata database. For this research however, the ontology editor Protégé [44] was used to create the annotations.

3.3. Classification

As mentioned in the previously given overview, the first core framework process of Classification is optional. Class data is often already available in Web data sources, for example through means of a category hierarchy. When a category hierarchy is available, a category mapping algorithm, such as SCHEMA[45], can be used to obtain mappings between the category hierarchy and the product classes in the ontology. However, this subsection explains the process we propose to use when class data is not available. It uses the Property Matching process (explained in the next subsection), to measure the best fit between a raw product and the ontology product classes.

Figure 2 shows that the input of the Classification process consists of the raw product to classify, the sets of total classes and properties in the ontology, and two threshold parameters. The output of the algorithm is an association (type-of) between the raw product and an ontology class, such as 'TV' or 'Camcorder'. Algorithm 1 explains how the proper class is determined.

Classification computes the *highest information gain* per key-value pair to create a fit score per product class

Algorithm 1 Classification of a Raw Product

Require: set of product classes C from the ontology
Require: set of key-value pairs K from the raw product
Require: Average Information Gain Threshold t
Require: function getHighestInformationGain(k, c), returns the highest normalized information gain between a key-value pair $k \in K$ and a product class $c \in C$

```
1: i_c ← 0 {Keep track of average information gain i_c for each c ∈ C}
2: for all k ∈ K do
3:    {Add the highest information gain for this key-value pair and each product class, divided by the total number of key-value pairs for normalization, to the total information gain of each product class.}
4:    for all c ∈ C do
5:       i_c = i_c + getHighestInformationGain(k, c)/|K|
6:    end for
7: end for
8: {Determine the best matching product class and return it if its score exceeds the threshold}
9: bestMatch ← null
10: bestScore ← 0
11: for all c ∈ C do
12:    if i_c ≥ t and i_c > bestScore then
13:       bestMatch ← c
14:       bestScore ← i_c
15:    end if
16: end for
17: return bestMatch
```

(by taking into account all key-value pairs): the average information gain. The information gain measures the specificity of a property for a certain product class. The information gain used here differs from the "classical" information gain measure used for instance-based classifications with decision trees.

Algorithm 2 explains how the highest information gain between one key-value pair and a product class is computed. As visible from the pseudo-code, the algorithm searches for the best-fitting property to a key-value pair.

Algorithm 2 Computing the Highest Information Gain

Require: key-value pair k from the raw product
Require: set of product classes C from the ontology
Require: product class $c \in C$ from the ontology, which is to be matched with k
Require: set of properties P_c from the ontology, for which c is in the domain of each $p \in P_c$
Require: Similarity Threshold s
Require: function propMatchScore(k, p), which computes the similarity score between k and $p \in P_c$
Require: function propDomainSize(p), which returns the number of classes in C that are entailed by the domain of property $p \in P_c$

```
1: maxInformationGain ← 0
2: for all p ∈ P_c do
3:    matchScore ← propMatchScore(k, p)
4:    if matchScore ≥ s then
5:       informationGain ← 1 − propDomainSize(p)/|C|
6:       if informationGain > maxInformationGain then
7:          maxInformationGain ← informationGain
8:       end if
9:    end if
10: end for
11: return maxInformationGain
```

For this property, it returns the information gain, which is thus the *highest* information gain. It is the added value of the fact that the raw product has a certain property, in relation to finding the correct product class. A matching property that is used for many product classes, such as 'width', adds little value, whereas a specific one, such as 'TV tuner', yields a higher information gain. For every product class, the highest information gains per key-value pair of the raw product are aggregated, and their average is computed in order to obtain the average information gain. Based on this measure, the best class is chosen, as Algorithm 1 illustrates.

The information gain is dependent on the *Property Match Score*, as Algorithm 2 depicted. This is actually the score that is computed by the Property Matching process, and explains the dependency of Classification on the Property Matching process. Algorithm 3 explains how the score is computed. The details will however be explained in the subsection on Property Matching.

The Classification process is dependent on two parameters, as stated in the requirements of the algorithms and Fig. 2. The first, the *Average Information Gain Threshold*, is used to strike a desirable balance between the recall and precision of the algorithm. When no threshold is used, products with a very low average information gain will still be classified, but with a high probability of failure. When the Average Information Gain Threshold is set, high-risk classifications will be skipped, that is, the classifier will return `null`. This moment could be used in an application to ask for user input, to prevent the product ontology from getting polluted. The higher the Average Information Gain Threshold, the higher the precision and the lower the recall of the Classification process. The second parameter is the *Similarity Threshold*, which is actually a parameter from the Property Match process. It will therefore be explained in the next subsection.

3.4. Property Matching

As Fig. 2 depicts, Property Matching is dependent on the result of Classification (a product class linked to the raw product), the raw product, the sets of ontology properties and classes, and the Similarity Threshold. The goal of Property Matching is to map each raw product key to an ontology property, as preparation for the Value Instantiation. To achieve this goal, the *Property Match Score* between each key-value pair from the raw product and each ontology property is computed using Algorithm 3.

The Property Match Score consists of two components: a lexical comparison between the raw product key and the ontology property, and a regular expression match. The regular expression match is optional, and depends on whether the ontology property is annotated with a regular expression in the OntoProduct Metadata or not. As

Algorithm 3 Property Match Score

Require: key-value pair k from the raw product

Require: set of product classes C from the ontology

Require: product class $c \in C$ for which to compute the Property Match Score using k

Require: set of properties P_c from the ontology, for which c is in the domain of each $p \in P_c$

Require: set of lexical representations L_p for each $p \in P_c$

Require: set of regular expressions R_p for each $p \in P_c$

Require: function levenshtein(k, L_p), which computes the maximum normalized Levenshtein similarity score between k and L_p

Require: function regexMatch(k, R_p), which matches value from k with regular expressions in R_p

$score_{lexical} \leftarrow$ levenshtein(k, L_p)
if $R_p \neq \emptyset$ **then**
  if regexMatch(k, R_p) = **true then**
    $score_{regex} \leftarrow 1$
  else
    $score_{regex} \leftarrow 0$
  end if
  $score_{total} \leftarrow (score_{lexical} + score_{regex})/2$
else
  $score_{total} \leftarrow score_{lexical}$
end if
return $score_{total}$

explained in Section 3.2.3, the regular expressions work as a filter for finding the right ontology properties to match, based on the raw product values. For instance, key-value pair `['Product Height (without stand)','27-7/8"']` from Fig. 3 would not be mapped to property 'hasHeight' if the regular expression of this property would not match to values with fractions such as 27-7/8.

The second component of the Property Match Score, the lexical comparison, uses the normalized Levenshtein similarity score to compare the raw product key to each lexical representation of the ontology property, which are part of the OntoProduct Metadata file. The Levenshtein distance [46] is a widely used edit distance measure for measuring the amount of difference between sequences of characters. Property Match Score uses the normalized Levenshtein similarity, which inverts the distance to transform it to a similarity, and then normalizes it by dividing with the maximum sequence length to become an index with range $[0, 1]$, where 1 would indicate that the sequences are equal. Of all lexical representations attached to the ontology property, the maximum similarity between a lexical representation and the raw product key is used.

For each key-value pair from the raw product, the ontology property with the highest Property Match Score is chosen under one condition: it must have a score that exceeds the Similarity Threshold (see Algorithm 2). This is a parameter of the framework that indicates how strict the Property Matching process should work regarding its mappings. When the threshold is very low, many raw product keys will be mapped, but with the chance of having a higher error rate. When the threshold is very high, less raw product keys will be associated with a property, but with higher accuracy. In the Evaluation section, we optimize the Similarity Threshold so that the algorithm works well under most conditions.

One special situation that can occur is when multiple properties match to a key-value pair with the same Property Match Score. In this case, the raw product key is mapped to all properties that have the same score, that is, if the Similarity Threshold has been exceeded. This characteristic enables for example the display resolution properties from Fig. 3 to be linked correctly with the key-value pair for resolution. In this case, both properties share the same lexical representation of 'Maximum Resolution', with which it has been annotated manually in OntoProduct Metadata. For this reason, the lexical score is equal. Moreover, the regular expressions of the display resolution properties both match to the value of the key-value pair, which results in both properties ending up with the same Property Match Score. Grouping in the regular expression enables the Value Instantiation process to ex-

tract the proper numeric data (for horizontal and vertical) from the complete raw product value.

3.5. Value Instantiation

Once the class of the raw product has been determined, and its key-value pairs have been mapped to ontology properties, the framework is ready for Value Instantiation. This step uses the output of the first two core process, in order to respectively create a product individual within the proper class, and to associate each value using the correct property. Value Instantiation consists of a collection of parsers, content spotters, and instantiation tools. This process is therefore explained using a flowchart, given in Fig. 5. For Value Instantiation, a clear distinction is made between qualitative and quantitative object properties, and data properties. These are therefore separately explained in the following subsections. The procedure from the flowchart is followed for every key-value pair from the raw product.

3.5.1. Instantiation of Qualitative Object Properties

When the Property Matching process has linked a key-value pair to a qualitative object property, all qualitative values from the ontology that are in the range of the property are gathered. The goal is to find one or multiple of these qualitative values in the raw product value. Often, Web stores combine multiple qualitative values in one key-value pair, as is the case with 'Other Connectors' in Fig. 3, for example. First, the lexical representations of all qualitative individuals are sorted on length, longest first. Then, the algorithm tries to find a matching lexical representation in the raw product value. If the search succeeds, the corresponding qualitative individual is attached to the product individual by means of the property found in the Property Matching process, and the matching part is removed from the raw product value string. This continues until no matches can be found anymore. The reason to order the procedure on lexical representation length, is that shorter labels might be contained in longer ones, leading to errors in parsing. This would for example be the case while parsing the raw product value `SDHC, MemoryStick, CompactFlash`; if the ontology contains qualitative value individuals for both SDHC and SD, the SD could match first without sorting, causing a faulty instantiation.

Extracting Qualitative Individuals from the Raw Product Key. The 'normal' way in which qualitative values are instantiated, is through the control path just described. Property Matching links the key-value pair to a qualitative object property, after which qualitative individuals are extracted from the raw product value. Two special situations arise however, in which qualitative values are parsed differently, as Fig 5 denotes: When a qualitative property is found, but the Value Instantiation process is incapable of extracting qualitative values, or, when the result of the Property Matching process for the key-value pair is `null`. In these cases, the Value Instantiation process does not examine the raw product value for qualitative individuals, but the raw product key. Although this might seem counterintuitive, it is actually an important aspect of the Value Instantiation process. For example, a common situation in which it is needed to examine the raw product key instead of the value, is for qualitative properties such as 'Features'. Many features, such as 'Sleep Timer', are often not structured as `['Feature','Sleep Timer']` in the key-value pairs, but more likely as `['Sleep Timer','Yes']`. In the last case, Property Matching will be unsuccessful, as Sleep Timer is a qualitative individual (from the features class), and not a property in the ontology. In this situation, the raw product key will be examined for matches with any qualitative individuals from the ontology, in a similar fashion as with 'normal' qualitative value instantiations, in which the Property Matching result is used. When a qualitative individual is found in the raw product key, the ontology is checked for properties that both have a range that includes the found individual, and a domain that entails the product class of the current

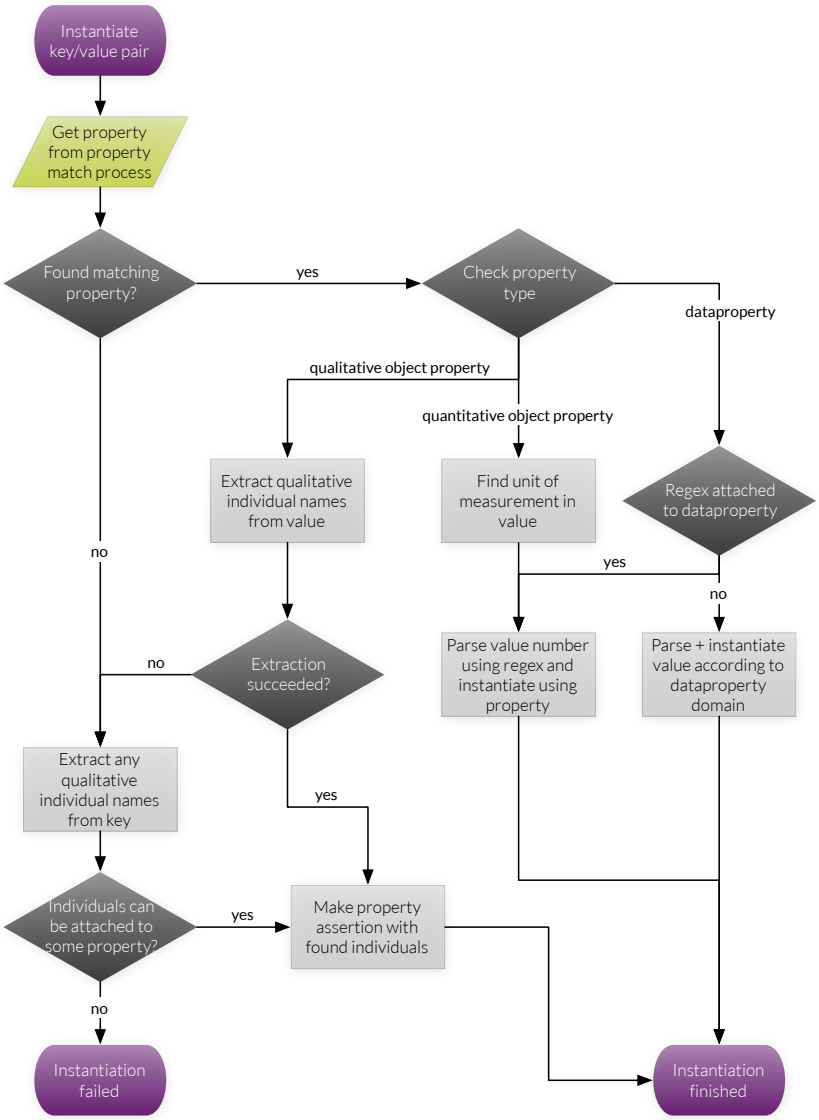

Figure 5: Overview of the instantiation process as flowchart.

product individual is entailed. Such a property is needed to be able to link the qualitative individual to the product individual in case that the property was not previously discovered with the Property Matching process.

Finding a qualitative individual in the raw product key does not provide sufficient information on itself to be able to assert ontology knowledge axioms. Whether the assertion can be made, also depends on the raw product value. Using what we call the *Boolean Value Convertor*, the raw product value is checked on terms such as 'false', 'no', 'none', '0', '-', 'optional', 'null', 'N/A', 'not available', and 'not applicable', and aborts the instantiation when such a term is encountered. If the raw product value passes this test, the ontology is instantiated with property assertions, each containing one found qualitative individual.

The extraction of qualitative individuals from the raw product key enables the Value Instantiation process to handle key-value pairs like `['Sleep Timer','Yes']`. As mentioned before, and as Fig. 5 makes clear, this procedure is also followed when 'normal' qualitative value instantiation is unsuccessful, that is, when there is a result from Property Matching, but no qualitative individuals

can be found in the raw product value. This problem arises for example with `'AM/FM Tuner','Yes'`, which does have a match with ontology property 'hasRadioTuner' based on one of its lexical representations, but does not contain qualitative individuals in the raw product value. In this case, looking at the raw product key solves the problem and successfully instantiates `hasRadioTuner` to `AM` and `hasRadioTuner` to `FM`.

3.5.2. Instantiation of Quantitative Object Properties

Parsing and instantiating quantitative values is very different from working with qualitative values. All quantitative values are parsed using regular expressions. By means of grouping, these enable to select the numeric data from the raw product value, disregarding additional content such as the unit of measurement. Note that some key-value pairs need multiple instantiations. Hence, multiple groups may exist in the regular expression, or the complete expression can match multiple times in one raw product value. The regular expressions come from the Ontoproduct Metadata, which is manually defined. When Property Matching has linked the key-value pair to a quantitative property, and no regular expression is attached to the property through the OntoProduct Metadata, then a default regular expression for parsing values is used. The default regular expression is a generic value extractor and is capable of extracting numerical values.

Extracting the Unit of Measurement. Usually, a quantitative value contains a unit of measurement. This unit of measurement is parsed in a similar fashion as parsing qualitative raw product values, which is described in Section 3.5.1. As discussed in Section 3.2, the quantitative properties refer to a fixed set of possible units of measurement. For every parsed numeric value from the raw product value, an associated unit of measurement is searched, and if possible, the new quantitative value individual is linked to this unit individual by means of the 'hasUnitOfMeasurement' property. Fig. 4 gives an indication of how a value individual is linked with the product individual and unit of measurement. When no unit of measurement is found, it is simply not instantiated.

3.5.3. Instantiation of Data Properties

The third and last type of instantiation is when the Property Matching process returned a data property. Data properties are less commonly used than object properties in OntoProduct. Mostly, they are used for Boolean assertions (i.e., 'hasTouchscreen'), numeric data without unit of measurement (i.e., 'region code'), and strings (i.e., 'product name'). The values can be parsed in two ways: using a regular expression that is attached to the property, or, using a specific parsing method based on the datatype range of the data property. When a key-value pair linked to a data property needs to be instantiated, and the property, say 'hasTouchscreen', appears to have a data range of `xsd:boolean`, a boolean parser is used. This parser aims to find terms in the raw product value, using exact lexical matching, that could indicate whether the data value should be true or false. Similar parsers are used for integers, floats, and strings (or literals).

3.5.4. Finalizing the Value Instantiation

Using all extraction rules described above, Value Instantiation is capable of converting a raw product key-value pair into ontology assertions. For each key-value pair of the raw product, the process, as made visible in Fig. 5, is repeated. Though there are various points at which parsers could fail, preventing actual instantiation, it is easy to keep track of all failures and handle these separately. An application could for example hand the problematic key-value pairs over to the user, which could then instantiate them manually.

4. Evaluation

This section presents an overview and discussion of the performance of the FLOPPIES framework on our data, by

means of a component-wise analysis of the various steps in the framework. First, we elaborate on how the experiment has been conducted, and which performance measures have been used throughout the evaluation. Afterwards we present the results, and discuss the performance of the framework by comparing it with the performance of a baseline approach.

4.1. Evaluation Design

This section discusses how the evaluation experiment has been set up. It provides a detailed overview of the used data and the methods employed to train the FLOPPIES framework.

The raw product data was obtained from two different Web sources, in order to increase the heterogeneity in the data set. Both sources are Web stores: Best Buy [47] and Newegg.com [48], which are large and well-known retailers in consumer electronics. As the research is focused on populating an ontology with product data, the Web crawler was intentionally kept simple. It crawls through eight predefined categories and obtains product data from them, using fixed extraction rules that are specific to each Web store. Seven of these categories are represented by a product class in the ontology, which means the products can be instantiated, whereas one category is not. By including a category that does not exist as a product class in the ontology, we can check whether the framework correctly refuses to instantiate the products from this category. For each product, the title and a tabular list, containing property information about the product as key-value pairs, were extracted from the Web store and stored along with product data from other products belonging to the same category. The end result consists of sets of products, each set describing a category from a specific Web store.

As mentioned earlier in Section 3.2.3, a part of the obtained product data is used to augment the ontology by enriching it with metadata. The metadata consists of lexical representations and regular expressions, which are manually annotated to ontology entities. The raw product keys are used to add lexical representations to properties, whereas the raw product values are used to construct regular expressions, which are also annotated to properties. The resulting metadata can be used by the FLOPPIES framework to match tabular data, originating from the Web, with properties in the ontology, and for instantiation of the values. For a proper evaluation of the FLOPPIES framework it is important to assess its performance on data that was not used to enhance the ontology. Therefore, each data set obtained by the crawler is split into a training and a test set, using a 60% – 40% split which randomly distributes the products in the file across both sets. This ensures that we have data available, for each category and from each Web store, that can be used for either training or testing. After splitting the raw product data, we obtain a training set consisting of 1046 products in total, whereas the test set contains 672 products.

Each step in the framework depicted in Fig. 2 is evaluated separately. In order to compute the performance measures we have to be able to compare the output of each step in the framework with a reference, known as the golden standard. The golden standard for the Classification process can be generated automatically in our case, as the products from each product class are stored in separate training or test data sets, and the name of each set corresponds to the correct product class in the ontology.

Unfortunately, creating the golden standard for the Property Matching process is far more complicated and therefore it cannot be generated automatically. Due to the sheer amount of different properties, either originating from the tabular data or the ontology, it is not feasible to provide a complete golden standard manually. Therefore, for evaluation of the Property Matching process, the software prompts the user for input whenever it comes across a mapping from the Property Matching process that it has not encountered before. The user can then select whether the mapping is correct or not and the user input is stored

in a knowledge base, which can be consulted the next time the evaluation is performed.

For evaluating the Value Instantiation process we manually instantiated products in the ontology beforehand, thus creating a golden standard. As manually instantiating products is a very time-consuming process, we decided to instantiate a subset of the data, namely TVs and MP3 players, consisting of 48 complete products from both Web stores. Because the golden standard is only available for the manually instantiated products and not for all the products, we only evaluate the performance of this step for these products. We have chosen for TVs and MP3 players because TVs are generally described with much detail in both Web stores, whereas the tabular data obtained from MP3 players is often pretty scarce and lacking in detail on the Web store page. In order to analyze how the two considered Web shops compare in terms of the product descriptions they use, we computed the overlap in product attributes and values. For the TVs category, there are on average 7.2% matching keys. We computed this average over all the pairs of product descriptions that describe the same product. For one product pair we compute the match value by dividing the number of matching keys by the maximum number of matches (i.e., $\min(|K_a|, |K_b|)$, where K_a and K_b represent the product attributes of description a and b, respectively). For MP3-players, the percentage of matching keys is much lower, i.e., 0.6%. Furthermore, we also computed, for the keys that matched, the overlap in the corresponding values. We found that for TVs 57.4% of these values match, while for MP3-players this is 12.8%.

For component evaluation of Property Matching, perfect class data was used as input, enabling a more accurate analysis of this component. This is done as the Property Matching process uses the product class as a filter, i.e., it only tries to match tabular data with properties from the ontology that are valid for the specific product class. By ensuring that the supplied input for the Property Matching process is completely accurate, we can evaluate the performance of this particular component in a more objective manner. Evaluation of the Value Instantiation is dependent on both Classification and Property Matching. As no golden standard for Property Matching is available, the Value Instantiation is evaluated with performance dependency of this step. Since the Classification process is seen as optional, the Value Instantiation will be evaluated both with perfect class input and with the result from the Classification process.

The FLOPPIES framework uses two different parameters, the *Average Information Gain Threshold* and the *Similarity Threshold*, for which the optimal values need to be computed. However, due to the interoperability between the Classification and the Property Matching processes, optimizing both parameters might seem like a convoluted process. Fortunately, because there is a golden standard for the Classification process, perfect class input for the Property Matching process can be used. This allows for the computation of the optimal value for the Similarity Threshold, as other variables are eliminated and thus the differences in performance are now solely caused by varying the Similarity Threshold value. Afterwards the optimal value for the Average Information Gain Threshold can be computed, given the optimal Similarity Threshold.

It is preferable to compare the results obtained by the FLOPPIES framework with another approach. However, as there is no freely available implementation of other relevant ontology population frameworks, and not enough information to precisely recreate a framework, we decided to create baseline approaches as well.

The baseline Classification process computes the lexical similarity, using the longest common substring as measure, between the raw product title and each product class label name in the ontology for the classification. The baseline Property Matching process tries to find the highest normalized Levenshtein similarity score between a key-value pair from the raw product data and the lexical representations of a property from the ontology. The used

baselines are straightforward and based on purely lexical approaches. We have chosen these as we want to investigate if the addition of semantics in the ontology population processes can provide benefits compared to lexical-based approaches. This type of baselines have been used also in the past for comparing lexical and semantic approaches (e.g., TF-IDF versus CF-IDF [49]).

We have opted not to evaluate the performance of the FLOPPIES framework against a different process for the Value Instantiation process, because it is more like a collection of different value extraction rules rather than a single unified algorithm. Together they form the logic to parse and instantiate a wide array of values, but removing some rules for creating a simpler process would obviously only yield lower results and therefore would not really contribute to a useful evaluation of the framework.

We have implemented the software and the experiments in Java. For the storage and retrieval of RDF data, we have used the Jena library [50]. Furthermore, we have used the Google Guava [51] library for caching and improved type primitives support.

4.2. Performance Measures

This section describes the performance measures that were used to evaluate the FLOPPIES framework and explains the used definitions for each step in the framework. For the evaluation of the framework we use a binary classification scheme, which is commonly used for evaluating the performance of classification and mapping algorithms. We employ the standard measures that can be computed with such a scheme, e.g., precision, recall, and the F_1-measure we have [52]. However, in this case we need to use a slightly adapted form, as it is not a pure binary problem.

For the Classification process, a *true positive* (TP) indicates that the framework has mapped a raw product to the correct product class. Unlike regular binary classification, where a *false positive* (FP) would mean that the framework mapped something which it should not have mapped at all, here it could also mean that it should have mapped the raw product, but it mapped to a wrong product class instead. A *true negative* is a raw product that has been correctly mapped to `null`, whereas a *false negative* (FN) indicates a raw product that should have been mapped to a product class, but the framework mapped it to `null`.

The evaluation of the Property Matching process basically follows the same definitions as the Classification process, but it maps key-value pairs to properties, rather than mapping raw products to a product class. Note that a single key-value pair can be mapped to multiple properties, which could result in a slightly different amount of mappings per algorithm run, depending on the used parameter values.

Rather than individually evaluating all RDF triples created by the Value Instantiation process, we adopt a graph-based evaluation approach. The reason for this is trivial: consider a key-value pair like `['Product Width', '1.09m']` from the raw product data. This key-value pair should be instantiated with multiple RDF triples, as depicted by Fig. 4, because we need to instantiate the value, the unit of measurement and the property assertion separately. Leaving out one of the triples would mean that the other triples loose most of their meaning, as a value is rather meaningless without a unit of measurement and vice versa. Therefore, we combine the triples of a quantitative value and evaluate them as a whole. In other words, for each triple where the instantiated product individual is the subject, we evaluate its subgraph as a whole.

As we manually instantiated 48 products for the golden standard, the instantiated products by the FLOPPIES framework can be compared to the products in the golden standard. Within this context a true positive means that a property was correctly instantiated, as it also occurs in the golden standard. A false positive indicates that the property should not have been instantiated at all, or that the associated value, unit of measurement, or individual, is

wrong or missing. Whenever the golden standard contains a property that the instantiated product by the framework does not have, it is counted as a false negative. Note that there are no true negatives in the evaluation of the Value instantiation process, as the instantiated ontology is only being compared to the golden standard ontology, and non-existing assertions cannot be counted. One could propose to count the number of key-value pairs from the raw product data, for which no instantiation has been made while manually creating the golden standard ontology. However, since there is no direct relation between the number of key-value pairs and the number of instantiated facts, it is impossible to count the number of true negatives using this way. This is because one key-value pair can contain any number of facts that require to be separately stored in the ontology.

Using the aforementioned definitions, the following performance measures can be computed:

$$\text{recall} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$
$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}}$$
$$\text{specificity} = \frac{\text{TN}}{\text{N}} = \frac{\text{TN}}{\text{FP} + \text{TN}}$$
$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$
$$\text{F}_1\text{-measure} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$
$$\text{F}_{0.5}\text{-measure} = 1.25 \cdot \frac{\text{precision} \cdot \text{recall}}{0.5 \cdot \text{precision} + \text{recall}}$$

The F_1-measure is the harmonic mean of precision and recall, which means that both precision and recall are equally important. However, for the evaluation of the optional Classification process, the $\text{F}_{0.5}$ score is also computed, for which the precision is twice as important as the recall. This score might be more preferable to use as performance measure, as instantiating raw products with the wrong product class would pollute the ontology and does not contribute to solving the search problems on the Web. It is envisioned that the Classification process uses a conservative approach and prompts the user for input when it cannot determine the correct product class with enough certainty. The $\text{F}_{0.5}$ score is more useful for this usage scenario, but we also include the F_1-measure for the Classification process in the results for completeness.

4.3. Results

This section presents the obtained results for each step of FLOPPIES, along with an in-depth discussion of these results.

4.3.1. Training set results

First of all, the two parameters that are used by the FLOPPIES framework need to be optimized. Therefore, we run the algorithm with different parameter values on the training set. Due to the interoperability between the Classification process and the Property Matching process, we first optimize the Similarity Threshold parameter in the Property Matching process, using the golden standard from the classification step as input. In order to find the optimal value, we raised the threshold from 0 to 1 in steps of 0.05. Table 1 shows the results of the Property Matching process on the training set, both for the FLOPPIES framework and the baseline algorithm.

At first, the framework obtains a better F_1-measure by increasing the Similarity Threshold, until the score stabilizes, between 92% and 95%, from a Similarity Threshold of 0.70 onwards. As expected, the precision increases and the recall decreases when the Similarity Threshold is increased, due to the stricter lexical matching. At the optimal Similarity Threshold level (0.80), the number of false positives has declined to 395 out a total of 28038 mappings, whereas the number of false positives at a Similarity Threshold of 0.60 was quite a bit higher: 1462 out of 28146 mappings. Note that the small discrepancy between the total number of mappings is caused by the fact that a single key-value pair can be mapped to multiple properties if their similarity scores are both the same. Although the number of false positives continues to drop when increasing the Similarity Threshold beyond 0.80, the sharp increase

Table 1: Training set results for Property Matching using golden standard classification

Process	Similarity Threshold	Precision	Recall	Accuracy	Specificity	F_1-measure
Baseline	-	49.07%	100.00%	49.07%	0.00%	65.84%
FLOPPIES	0.60	71.21%	97.91%	78.01%	55.78%	82.45%
FLOPPIES	0.65	82.54%	95.67%	86.71%	76.14%	88.62%
FLOPPIES	0.70	90.90%	94.93%	92.03%	88.54%	92.87%
FLOPPIES	0.75	92.90%	94.40%	93.14%	91.69%	93.64%
FLOPPIES	0.80	97.28%	93.47%	95.07%	96.94%	95.34%
FLOPPIES	0.85	99.05%	90.78%	94.60%	99.00%	94.73%
FLOPPIES	0.90	99.87%	90.66%	94.96%	99.86%	95.04%
FLOPPIES	0.95	99.89%	88.10%	93.62%	99.89%	93.62%
FLOPPIES	1.00	99.90%	85.86%	92.43%	99.90%	92.35%

in false negatives prevents it from obtaining a higher F_1-measure. A total of 987 false negatives has been measured at the optimal value of 0.80, which gradually increases to 2109 when a Similarity Threshold of 1.00 is used.

Also worthy to note is the enhanced precision of the FLOPPIES framework compared to that of the baseline algorithm, scoring 97.28% at the optimal Similarity Threshold against 49.07% respectively. This is due to the fact that the baseline algorithm uses an optimistic approach, which enables it to actually score better on true positives than the FLOPPIES framework: 16971 against 14136. However, it comes at the expense of a large number of false positives, which considerably lowers the precision and therefore also the F_1-measure.

Using the optimal Similarity Threshold of 0.80, obtained from the first step, the Average Information Gain Threshold of the Classification process can now be optimized. By keeping the Similarity Threshold constant and varying the Average Information Gain Threshold, raising it from 0 to 1 in steps of 0.05, the results in Table 2 are obtained. As is evident from the results, the Average Information Gain Threshold functions as a parameter for finding the optimal trade-off between precision and recall. Generally speaking, the precision will increase when the threshold is increased as well, at the expense of a decline in recall. In other words, increasing the threshold means that the algorithm cannot classify as many products as before, but the ones it did classify are more likely to be correct. This is due to the fact that a higher threshold means that the properties of a product need to convey more specific information about the product, in order for the algorithm to map them to a product class from the ontology. Therefore, a product with a high Average Information Gain can be more reliably classified than a product with a lower Average Information Gain.

In contrast to the Similarity Threshold in the Property Matching process, the optimal value for the Average Information Gain Threshold is relatively low. The Similarity Threshold is a threshold operating on a lexical matching score, whereas the Average Information Gain Threshold operates on an average, namely the Average Information Gain for all key-value pairs from a raw product. This explains the difference in the optimal value, especially considering that nearly every product also has very generic key-value pairs, like the weight of a product, that help bring down the Average Information Gain. Also interesting to note is the difference between the F_1 and $F_{0.5}$ scores. Because the $F_{0.5}$ score emphasizes the precision, the highest $F_{0.5}$ score of 70.18% is obtained with an Average Information Gain Threshold of 0.20, whereas the

Table 2: Training set results for Classification using optimal Similarity Threshold of 0.80

Process	Average IG Threshold	Precision	Recall	Accuracy	Specificity	F_1-measure	$F_{0.5}$ score
Baseline	-	29.83%	100.00%	29.83%	0.00%	45.95%	34.70%
FLOPPIES	0.00	49.33%	100.00%	49.33%	0.00%	66.07%	54.89%
FLOPPIES	0.05	49.33%	99.81%	49.33%	0.00%	66.07%	54.93%
FLOPPIES	0.10	54.93%	83.53%	50.67%	5.24%	66.27%	58.97%
FLOPPIES	0.15	68.91%	69.20%	57.07%	29.81%	69.06%	68.97%
FLOPPIES	0.20	72.17%	63.19%	56.31%	39.13%	67.38%	70.18%
FLOPPIES	0.25	70.00%	48.37%	46.94%	43.01%	57.21%	64.25%
FLOPPIES	0.30	58.05%	28.68%	32.50%	43.01%	38.39%	48.18%

highest F_1-measure is achieved using an Average Information Gain Threshold of 0.15. As argued in Section 4.1, achieving a high precision is paramount for the Classification process, as it is better to ask the user for input rather than instantiating products with the wrong product class. Therefore, we consider an Average Information Gain Threshold of 0.20 as optimal for the training set, because it achieves the most precision and the highest $F_{0.5}$ score.

In addition, the results show that it can be quite difficult to classify products based on their properties alone. While this may seem a trivial task to humans, the differences in product properties between multiple product classes is often smaller than you would imagine. For instance, consider a camcorder and a digital photo camera: both are small, have a lens, connect to a computer through USB, use memory cards to store information, and so on. They share many characteristics, but there is essentially only one defining characteristic that separates them: a camcorder is meant for shooting video, whereas a digital photo camera is meant for shooting pictures. And even in this example the line between the two is blurry, as many digital photo cameras nowadays are perfectly capable of shooting videos as well. This high degree of function integration between products can be found in numerous products within the domain of consumer electronics, which makes the classification of products, based purely on product properties, a non-trivial task. Fortunately, practically every Web store contains a product category hierarchy, which can be used for the classification of products. That is why the Classification process in the FLOPPIES framework is optional and is only meant as a backup whenever insufficient information is available.

To complete the evaluation on the training data, the Value Instantiation process is executed using the output from the previous steps in the framework. Table 3 shows the results of this process when using either the golden standard classification or the output from the Classification process. As the training set contains 27 out of the 48 products that were manually instantiated in the golden standard ontology, the performance on those 27 products is evaluated. At first glance the results seem counterintuitive, as the Classification process of the FLOPPIES framework actually has a slightly better F_1-measure than the Golden standard, scoring 83.79% and 83.64% respectively. However, this is caused by the method used to evaluate this part of the framework, which is explained in more detail in Section 4.2. Because the evaluation is performed on the instantiated products in the ontology, the products that were not instantiated are not evaluated. As the Classification process could not determine the product class of one MP3 player, due to the lack of specific product infor-

Table 3: Training set results for Value instantiation using optimal Average Information Gain Threshold (0.20) and Similarity Threshold (0.80)

Classification	Precision	Recall	Accuracy	F_1-measure	Product instantiation rate
Golden standard	82.11%	85.23%	71.89%	83.64%	100.00%
FLOPPIES Classification	81.67%	86.05%	72.11%	83.79%	96.30%

mation, the Value Instantiation process only instantiated 26 of the 27 products, resulting in a product instantiation rate of 96.30%. Using the golden standard means that the product does get instantiated, but the Property Matching and Value Instantiation process have relatively more difficulty with this particular MP3 player, which results in the slightly lower F_1-measure.

From these results we can conclude that the FLOPPIES framework as a whole performs rather well when instantiating TVs and MP3 players. However, it still fails to instantiate some properties or it is unable to instantiate them correctly.

Error analysis on the instantiated products reveals that occasionally the framework is not capable of extracting and instantiating all individuals from a list of qualitative values. For example, consider the key-value pair `['System Requirements','Windows: 2000 or later; Mac: OS X-10.4 or later']`, which can be instantiated with the property `ceo:hasCompatibleOperatingSystem`. Any person, who manually instantiates this key-value pair, would also instantiate property assertions for the versions of Windows and Mac OS X that were released after Windows 2000 and Mac OS X 10.4 respectively. However, for our Value Instantiation process it is difficult to determine for which individuals it should instantiate property assertions, as it is trying to match the value with the lexical representations of individuals from the ontology. Therefore, it is able to instantiate property assertions for the individuals `'ceo:Windows2000'` and `'ceo:MacOSXTiger'`, as their lexical representations are also present in the value of the key-value pair, but later versions are not recognized. Fortunately, because the Value Instantiation process is using a set of value extraction rules, we could easily add a new rule to replace 'or later' in the value with the lexical representations of the referred individuals. By adding a new property assertion between the individuals in the ontology, which states that a certain individual is the successor of the other individual, the Value Instantiation process could learn to instantiate property assertions for all compatible operating systems. We consider creating new value extraction rules and augmenting the ontology with more details about the relationship between individuals as useful future work for improving the framework.

4.3.2. Test set results

After optimizing both parameters of the FLOPPIES framework on the training set, the performance of the framework on the test data can be evaluated.

Table 4 shows that the performance of the FLOPPIES framework on the classification of products from the test data is equal to the performance on the training data. The F_1-measure dropped slightly, from 67.38% to 66.24%, while the $F_{0.5}$ score dropped from 70.18% to 69.18%. Relatively more products are marked as a false positive though: 124 out of 672 (18.45%) against 182 out of 1046 products (17.40%).

Although the Classification process is optional within the framework, more work on lowering the amount of false positives would be beneficial, as these errors could cause more problems later on in the Property Matching and Value Instantiation processes. One way to achieve this could be to also take the value of the key-value pairs into consideration for the information gain score. For example, many consumer electronics have an LCD display, which

Table 4: Test set results for Classification using optimal Average Information Gain Threshold (0.20) and Similarity Threshold (0.80)

Process	Precision	Recall	Accuracy	Specificity	F_1-measure	$F_{0.5}$ score
Baseline	29.64%	100.00%	29.46%	0.00%	45.52%	34.30%
FLOPPIES	71.30%	61.84%	53.27%	28.74%	66.24%	69.18%

means that the property currently does not yield much information gain for our framework. However, the value could help differentiate between product classes and increase the information gain for this property. For instance, both a TV and an MP3 player have an LCD display, but if the display size of a raw product is 40", it is most likely that the product is a TV. By comparing this numerical value with TVs and MP3 players that are already instantiated in the ontology, a higher information gain for this property can be achieved, thus making it easier to determine the correct product class. Therefore, we consider differentiating between values for the purpose of product classification as a useful future addition to the framework.

The Property Matching process also scores roughly the same on the test and training data, as can be seen in Table 5. The precision and recall have decreased a little bit, which is caused by the fact that the key-value pairs from the test data were not used to 'train' the ontology by adding lexical representations and regular expressions. Although the test data contains some new raw product keys, the Property Matching was still able to match many key-value pairs with properties, because the Similarity Threshold allows it to also map raw product keys with slight lexical variations. In practice, this means that a semi-automatic approach would only require training the algorithm with a few products from each product class in order to achieve satisfactory performance on Property Matching for all the products in a Web store.

By analyzing the results, we conclude that the regular expressions in conjunction with the lexical representations are often capable of correctly mapping key-value pairs to properties in the ontology. For example, the key 'Product Dimensions' is correctly mapped to `ceo:hasWidth`, `ceo:hasHeight`, and `ceo:hasDepth`, which demonstrates the usefulness of regular expressions for the Property Matching process.

While the Property Matching process performs quite satisfactory on most key-value pairs, it sometimes gets confused between properties representing a quantitative measure without a unit of measurement. Consider raw product keys 'DVI Inputs' and 'HDMI Inputs', of which only 'HDMI Inputs' can be mapped with `ontoproduct:hasNumberOfHDMIInputs` in the ontology. Unfortunately, the Property Match process also creates a mapping from 'DVI Inputs' to `ontoproduct:hasNumberOfHDMIInputs`, as their lexical similarity is fairly high and they both describe a count of inputs. This could be avoided by raising the Similarity Threshold, which in turn would mean that the framework is less capable of automatically mapping slightly varying raw product keys. However, as shown in Section 4.3.1, stricter lexical matching degrades the overall performance of the framework.

When running the FLOPPIES framework in its entirety, the results on the test data in Table 6 are obtained. Unlike the previous steps in the framework, the performance of the Value Instantiation process on the test data is considerably lower than the performance on the training data: the F_1-measure dropped from around 83% to approximately 77%. This is because the test data contains a few keys and values from key-value pairs that have a considerably different lexical representation than those used for annotating the ontology. While the Similarity Threshold allows for some lexical variation to occur, a key-value pair with a considerably different lexical representation

Table 5: Test set results for Property Matching using golden standard classification and optimal Similarity Threshold of 0.80

Process	Precision	Recall	Accuracy	Specificity	F_1-measure
Baseline	48.30%	100.00%	48.30%	0.00%	65.14%
FLOPPIES	96.95%	93.27%	94.80%	96.58%	95.07%

Table 6: Test set results for Value instantiation using optimal Average Information Gain Threshold (0.20) and Similarity Threshold (0.80)

Classification	Precision	Recall	Accuracy	F_1-measure	Product instantiation rate
Golden standard	77.12%	76.09%	62.07%	76.60%	100.00%
FLOPPIES Classification	76.99%	77.41%	62.87%	77.20%	90.48%

still would not exceed the threshold, and thus it cannot be mapped to a property in the ontology. This means does not find as many mappings for the test set as for the training set. The effect can also be observed in the product instantiation rate when using the Classification process of the FLOPPIES framework to perform the classification, which drops from 96.30% to 90.48%. Two MP3 players from the total set of 21 products in the test set could not be classified by the Classification process. Regardless of the decline in performance though, the FLOPPIES framework still performs quite well, based on the obtained results, on instantiating TVs and MP3 players in the ontology.

5. Conclusions

This paper proposes FLOPPIES, a framework capable of semi-automatic ontology population of product information from Web stores. It employs a predefined ontology, compatible with the GoodRelations ontology for e-commerce, in order to formalize the raw product information contained in tabular format on product pages in Web stores. With product information formalized in an ontology, better product comparison or recommendation applications could be built, using full parametric search. Furthermore, it could facilitate the aggregation and exchange of product information between multiple Web sites without relying on Web stores to provide their data in a specific format, as is the case with current comparison platforms.

The framework consists of an optional Classification process, which can identify the product class of a raw product by analyzing its key-value pairs and computing an Average Information Gain between each product class in the ontology and the key-value pairs from the raw product. It uses the second step in the framework, the Property Matching process, to compute this score. The Property Matching process computes a Similarity Score between a key-value pair and properties in the ontology, using both lexical matching and pattern matching with regular expressions. After the key-value pairs have been mapped to properties in the ontology, the Value Instantiation process instantiates the product information. A set of different value extraction rules is employed in order to instantiate the correct values and units of measurement.

The performance of the framework is compared to the performance of a baseline approach, which merely uses lexical matching for the Classification and Property Matching processes. Product information from 1718 products, spread across eight different consumer electronic product categories from Best Buy and Newegg.com, was gathered and split into a training and test set. The training set was used to annotate the ontology with lexical representations and regular expressions, which are used to im-

prove the performance of the matching and parsing processes. Afterwards, it is used in the component-wise analysis to compute the optimal parameter values for the Similarity Threshold and Average Information Gain Threshold, which are used in the framework as a cut-off for the Property Matching and Classification process respectively. Last, using the optimal parameter values, the performance of the all the steps in the framework on the test data is evaluated.

It is shown that the FLOPPIES framework performs considerably better than the baseline approach for the Classification process, achieving an $F_{0.5}$ score of 69.18% against 34.30%, due to the better precision. The Property Matching process also scores better than the baseline approach with an F_1-measure of 95.07% against 65.14%, due to the use of both lexical matching and patter matching. The evaluation of the Value Instantiation process was performed using a graph-based approach, comparing it to a manually instantiated ontology with 48 products. Although running the framework with the optional Classification process resulted in a classification of only 45 out of 48 products, it did manage to achieve a similar F_1-measure as when using perfect classification input, scoring roughly 83% and 77% for the training and test set, respectively.

For future research, there are several ideas that can further improve (semi-)automated product ontology population. First, FLOPPIES currently only uses the tabular data from product pages. However, often also textual descriptions are available, next to the semi-structured key-value pairs. Through text mining, one could try to use the descriptions to extract additional knowledge. Another unexplored possibility for the framework, is to use already instantiated ontology data for the instantiation of new data. Through data mining techniques such as clustering, the algorithm could for example learn when to match certain properties to key-value pairs.

The Classification process uses most of the time the raw product keys, via the Property Match Score. The raw product values however can sometimes also provide a good indication of the proper class. Take for example key-value pair `['Capacity','10-cup']`; the key is not very informative, however the value is a better indication for the fact that this key-value pair is one from a coffee machine page.

The Value Instantiation process could be enhanced by adding new value extraction rules and by creating new property assertions between individuals in the ontology that further specify the relationship between them. By formally defining in the ontology that 'Windows XP' is the successor to 'Windows 2000', the framework could also instantiate a property assertion for 'Windows XP' when it encounters a raw product value such as 'Windows 2000 or later'.

In the current framework, the regular expressions provide a reliable way for parsing values and filtering properties. However, regular expressions are labor-intensive to build, and the user needs quite some technical background in order to be able to make these. In the past years, there has been some successful research on automated generation of patterns like these. One could consider using such a technique for this framework, although it might affect the accuracy of the overall framework.

Acknowledgment

Damir Vandic is supported by an NWO Mosaic scholarship for project 017.007.142: *Semantic Web Enhanced Product Search (SWEPS)*.

References

[1] G.-Q. Zhang, G.-Q. Zhang, Q.-F. Yang, S.-Q. Cheng, T. Zhou, Evolution of the Internet and its Cores, New Journal of Physics 10 (2008) 123027.

[2] T. Berners-Lee, J. Hendler, O. Lassila, The Semantic Web, The Scientific American 284 (2001) 34–43.

[3] Bing, Google, Yahoo!, and Yandex, schema.org, 2013. `http://bit.ly/196asth`.

[4] Google Inc., Knowledge Graph, `http://bit.ly/18BtaMI`, 2012.

[5] C. Bizer, T. Heath, T. Berners-Lee, Linked Data - The Story So Far, International Journal on Semantic Web and Information Systems 5 (2009) 1–22.

[6] B. VijayaLakshmi, A. GauthamiLatha, D. Y. Srinivas, K. Rajesh, Perspectives of Semantic Web in E- Commerce, International Journal of Computer Applications 25 (2011) 52–56.

[7] J. B. Horrigan, Online Shopping, Pew Internet & American Life Project Report 36 (2008).

[8] Li, Beibei and Ghose, Anindya and Ipeirotis, Panagiotis G., Towards a Theory Model for Product Search, in: 20th International Conference on World Wide Web (WWW 2011), ACM Press, 2011, pp. 327–336.

[9] W3C OWL Working Group, OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition), Technical Report, W3C, 2012. http://bit.ly/c4CWDL.

[10] M. Hepp, GoodRelations: An Ontology for Describing Products and Services Offers on the Web, Knowledge Engineering: Practice and Patterns 5268 (2008) 329–346.

[11] D. Vandic, J. van Dam, F. Frasincar, Faceted Product Search Powered by the Semantic Web, Decision Support Systems 53 (2012) 425–437.

[12] W. K. Ng, G. Yan, E.-P. Lim, Heterogeneous Product Description in Electronic Commerce, ACM SIGecom Exchanges 1 (2000) 7–13.

[13] W. Holzinger, B. Krüpl, M. Herzog, Using Ontologies for Extracting Product Features from Web Pages, in: 5th International Semantic Web Conference (ISWC 2006), Springer, 2006, pp. 286–299.

[14] G. A. Miller, WordNet: A Lexical Database for English, Communications of the ACM 38 (1995) 39–41.

[15] C. Patel, K. Supekar, Y. Lee, OntoGenie: Extracting Ontology Instances from WWW, in: Workshop on Human Language Technology for the Semantic Web and Web Services. http://bit.ly/10eUcWH.

[16] D. Celjuska, M. Vargas-Vera, Ontosophie: A Semi-automatic System for Ontology Population from Text, Technical Report, KMi Institute, 2004. http://bit.ly/13EegA4.

[17] L. McDowell, M. Cafarella, Ontology-Driven, Unsupervised Instance Population, Web Semantics: Science, Services and Agents on the World Wide Web 6 (2008) 218–236.

[18] Y. Guo, J. Hu, Y. Peng, A CBR System for Injection Mould Design Based on Ontology: A Case Study, Computer-Aided Design 44 (2012) 496–508.

[19] Y. Guo, Y. Peng, J. Hu, Research on High Creative Application of Case-based Reasoning System on Engineering Design, Computers in Industry 64 (2013) 90–113.

[20] Y. Guo, J. Hu, Y. Peng, Research on CBR System Based on Data Mining, Applied Soft Computing 11 (2011) 5006–5014.

[21] M. Ciaramita, A. Gangemi, E. Ratsch, J. Saric, I. Rojas, Unsupervised Learning of Semantic Relations Between Concepts of a Molecular Biology Ontology, in: 19th International Joint Conference on Artificial Intelligence (IJCAI 2005), Morgan Kaufmann Publishers Inc., pp. 659–664.

[22] Gene Ontology Consortium and others, Gene Ontology: Tool for the Unification of Biology, Nature genetics 25 (2000) 25–29.

[23] P. Holmans, E. K. Green, J. S. Pahwa, M. A. Ferreira, S. M. Purcell, P. Sklar, et al., Gene Ontology Analysis of GWA Study Data Sets Provides Insights Into the Biology of Bipolar Disorder, American journal of human genetics 85 (2009) 13–24.

[24] UNSPSC, United Nations Standard Products and Services Code, 2012. http://bit.ly/13Ef5Ja.

[25] M. Hepp, unspscOWL, 2010. http://bit.ly/11BDls8.

[26] eCl@ss e.V., eCl@ss — Classification and Product Description, 2012. http://bit.ly/11bB2zw.

[27] M. Hepp, Products and Services Ontologies: a Methodology for Deriving OWL Ontologies from Industrial Categorization Standards, International Journal on Semantic Web and Information Systems 2 (2006) 72–99.

[28] M. Hepp, eClassOWL, 2010. http://bit.ly/11xXhv2.

[29] ODP (or Dmoz), Open Directory Project, http://bit.ly/11bBOHO, 2012.

[30] S. Damodaran, B2B Integration over the Internet with XML: RosettaNet Successes and Challenges, in: 13th International World Wide Web Conference (WWW 2004), ACM, pp. 188–195.

[31] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau, Extensible Markup Language (XML), World Wide Web Journal 2 (1997) 27–66.

[32] M. Hepp, J. Leukel, V. Schmitz, A Quantitative Analysis of Product Categorization Standards: Content, Coverage, and Maintenance of eCl@ss, UNSPSC, eOTD, and the RosettaNet Technical Dictionary, Knowledge and Information Systems 13 (2006) 77–114.

[33] B. Adida, M. Birbec, S. McCarron, I. Herman, RDFa Core 1.1 W3C Recommendation 07 June 2012, Technical Report, W3C, 2012. http://bit.ly/18BvYJL.

[34] CEO, Consumer Electronics Ontology — An Ontology for Consumer Electronics Products and Services, http://bit.ly/12Ir4bG, 2009.

[35] Martin Hepp, The Product Types Ontology: High-precision identifiers for product types based on Wikipedia, http://bit.ly/GEbALr, 2013.

[36] Martin Hepp, Extensions for GoodRelations for Specific Industries, http://bit.ly/1gl6ZM0, 2013.

[37] Martin Hepp, The OPDM project, http://bit.ly/1b4YUHB, 2013.

[38] C. Chang, M. Kayed, R. Girgis, K. Shaalan, A Survey of Web Information Extraction Systems, IEEE Transactions on Knowledge and Data Engineering 18 (2006) 1411–1428.

[39] H. Kopcke, E. Rahm, Frameworks for Entity Matching: A Comparison, Data & Knowledge Engineering 69 (2010) 197–210.

[40] A. K. Elmagarmid, P. G. Ipeirotis, V. S. Verykios, Duplicate Record Detection: A Survey, IEEE Transactions on Knowledge and Data Engineering 19 (2007) 1–16.

[41] D. Berrueta, L. Polo, MUO — An Ontology to Represent Units of Measurement in RDF, http://bit.ly/11bA5qP, 2009.

[42] G. Schadow, C. J. McDonald, UCUM — The Unified Code for Units of Measure, http://bit.ly/11xXrCC, 2010.

[43] J. Friedl, Mastering Regular Expressions, O'Reilly Media, Inc., 2006.

[44] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, S. W. Tu, The Evolution of Protégé: An Environment for Knowledge-Based Systems Development, International Journal of Human-Computer Studies 58 (2003) 89–123.

[45] S. Aanen, L. Nederstigt, D. Vandić, F. Frăsincar, SCHEMA - An Algorithm for Automated Product Taxonomy Mapping in E-commerce, in: 9th Extended Semantic Web Conference (ESWC 2012), volume 7295, Springer, 2012, pp. 300–314.

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

[47] Best Buy Co., Inc., Large US Online Retailer in Consumer Electronics, http://bit.ly/11Y4tAn, 2012.

[48] Newegg.com Inc., Online Retailer in Consumer Electronics, http://bit.ly/14yChtr, 2012.

[49] M. Baziz, M. Boughanem, N. Aussenac-Gilles, Conceptual Indexing Based on Document Content Representation, in: Context: Nature, Impact, and Role, Springer, 2005, pp. 171–186.

[50] B. McBride, Jena: A Semantic Web Toolkit, IEEE Internet Computing 6 (2002) 55–59.

[51] Google, Guava - Google Core Libraries, 2013. http://bit.ly/11Y4ww0.

[52] R. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval: The Concepts and Technology behind Search, Addison-Wesley Professional, 2011.