Enabling Product Comparisons on Unstructured Information Using Ontology Matching

Maximilian Walther, Niels Jäckel, Daniel Schuster, and Alexander Schill

Technische Universität Dresden, Faculty of Computer Science, Institute of Systems Architecture, Helmholtzstr. 10, 01062 Dresden, Germany,
maximilian.walther@tu-dresden.de

Abstract. Information extraction approaches are heavily used to gather product information on the Web, especially focusing on technical product specifications. If requesting different sources for retrieving such specifications, the outcome is of varying formats (different languages, units, etc.). The problem of how to bring such information sets into a unique, interchangeable format is not considered in many extraction systems.
We develop a generic process for semantically integrating heterogeneous product specifications with the help of a product information ontology. The approach is based on a number of measures for detecting the right product attributes in the ontology to be matched with the extracted specifications and finally normalizing the specifications' values (e.g., concerning units). The feasibility of our approach is proven in a federated product search prototype called Fedseeko.

Key words: information extraction, federated search, ontology matching, product information management

1 Introduction

Today's World Wide Web offers a wide amount of product information with disparate structure and location which is not easy-to-handle for the average online consumer anymore. This led to a need for platforms offering effective product comparisons. In many cases, the product information maintained on such platforms is still acquired by hand. Figure 1 shows examples of technical specifications lists for two digital cameras $dc1$ and $dc2$ being represented in different producer-dependent manners. For being able to compare both products on a dedicated platform, an employee has to extract those specifications, match them with a corresponding product model and normalize included values.

Hence, many approaches for automating the product information collection on the Web have been developed which mostly focus on extracting product information from unstructured or semi-structured sources like the producer websites shown in the figure. The part of integrating extraction targets with a central knowledge model is not considered in many cases. However, this step is important since it makes products comparable. This applies especially to electronic products where technical product specifications allow effective comparisons and strongly affect a potential consumer in choosing a particular product.

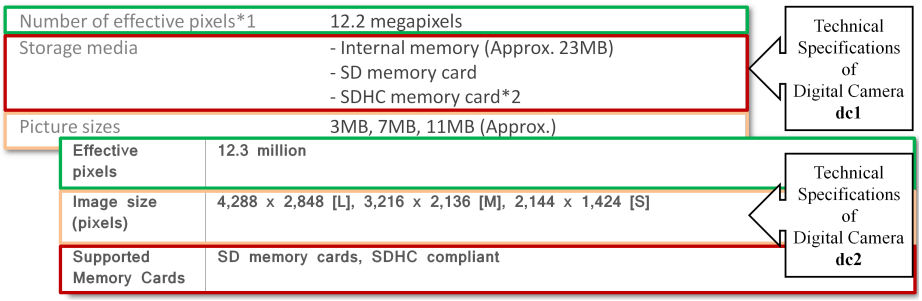

Fig. 1. Example for varying product specification formats of two digital cameras.

We develop a generic approach for integrating technical product specifications with a given product information ontology. Our contributions consist of an appropriate process for extracting, classifying, matching and normalizing product specifications using a central knowledge model and a number of domain-specific measures required for the product specification matching process. In the following, the topic of ontology matching will be emphasized.

2 Ontology Matching

Although product information is generally not available in the form of an ontology online, each web source's product information has a distinct structure described by some internal schema. Thus, it is reasonable to assume that extracted specifications are implicitely modeled by an ontology, shifting our problem to the area of ontology matching. *Ontology Matching*[1] describes the process of finding correlations (*Ontology Alignment*) between entities of different ontologies that can be used for transforming one ontology into another (*Ontology Mapping*). Ontology matching is closely related to schema matching. The characteristic sequence for matching schemas consists of the actual *matching* step, an *aggregation* and a *selection* step. Newer systems[2] diversify this sequence for offering more flexibility.

State of the art matching systems usually apply several elementary matchers for finding an alignment. Considering their matching granularity, they can generally be divided into element-level and structure-level matchers. A more detailed matcher categorization is given in [3]. Depending on whether the system executes its matchers sequentially or in parallel, the overall matcher is called hybrid or composite, respectively. When talking about extracted product specifications, structural information is not available, thus reducing the set of elementary matchers to the element-level ones. For compensating this major drawback, as many characteristics of product specifications as possible have to be identified to be exploited by adequate element-level matchers. The fact that not only the specifications' schema (tbox), but also corresponding instances (abox) are extracted, is helpful in this case.

2.1 Element-Level Matchers

The most basic element-level matcher type is the one of string-based matchers. Such matchers only compare given strings, e.g., by calculating the Levenshtein distance. COMA[4], a composite schema matcher that allows the combination of different matching algorithms, includes four such matchers. Cupid[5] uses two different string-based matchers for detecting prefixes and suffixes in its first operation step. String-based matchers are used in nearly every matching system.

Language-based matchers use NLP techniques for identifying individual words or phrases or execute a morphological analysis on given strings. In S-Match[6] such matchers are used for detecting the meaning of given concepts.

The third type are matchers using linguistic resources such as WordNet, e.g., for retrieving synonyms of given schema strings. OWL Lite Aligner[7] (OLA) uses WordNet while systems like Naive Ontology Mapping[8] (NOM) and its successor Quick Ontology Mapping[9] (QOM) apply application-specific vocabularies. If the vocabulary used in a schema is not too particular for a special domain, such matchers can be quite powerful.

Constraint-based matchers have the ability to detect similar datatypes or multiplicities. E.g., such a matcher could figure out that a datatype day and workingday are quite similar. Similarity Flooding[10] is a hybrid schema matching system and uses constraint-based matchers while doing fix-point computations on the graph representations of its input schemas.

The last element-level matcher type being interesting for this work is the one of alignment reusing. Such matchers try to find alignments between the input schemas and other schemas. If for example both input schemas would already have been matched with a third schema and the resulting alignments are known to the matcher, a transitive mapping approach would support the matching process. COMA was the first system to offer alignment reuse.

Although the matchers presented here are taken from a survey[3] of schema-based matching approaches, they may as well be applied to instance data. Thus, each similarity measure presented later-on will reference one of these matcher types. In the following, systems directly working on instance data are described.

2.2 Instance Matching

Instance matching has been adopted in a set of different approaches. ASID[11] is a relational database schema mapper. It divides its matchers in strong and weak matchers while the weak matchers exploit available instance data that is to be cleaned in advance. Automatch[12] is solely based on instance data of different schemas. The instance data is compared with a unique internal schema using techniques of the machine learning domain. GLUE[13] is a system for semi-automatic taxonomy and ontology matching. The first of two basic learning strategies is the content learner that captures word counts of instance data.

These systems are designed quite generically for being able to match a lot of different schemas. Their approaches can be adapted to better fit our product specifications domain.

3 Semantic Integration of Product Specifications

The objective of our system is to create integrated sets of product specifications originally gathered on the Web for being able to compare products efficiently. The overall process is presented in Figure 2.

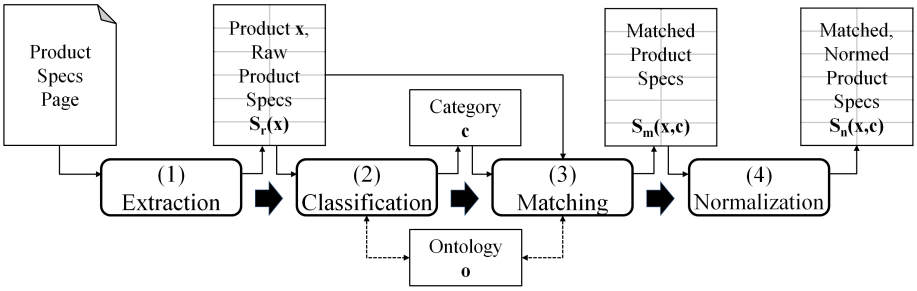

Fig. 2. Overall extraction and matching process.

As can be seen, the first step consists in extracting the set of raw technical specifications $S_r(x)$ (e.g., "Number of effective pixels*1: 12.2 megapixels" for $dc1$ in Figure 1) of a product x (in this case, our $dc1$) from a dedicated web page (e.g., a page on $dc1$'s producer website). This step as well as the automated locating of such web pages has already been described in [14]. Since the product's category might be unknown, the second step consists in classifying the product to be located in a product category $c \in C$ (e.g., "Digital Cameras") from an ontology o. This step is essential since the matching algorithm needs to know which properties of o can potentially be matched with $S_r(x)$. The basic approach of how to classify products has already been described in [15]. We developed a number of refinements for this technique. However, since it is not the main focus of this paper, it will only be taken into account for the evaluation section. In the main step, the extracted specifications can be matched with c's set of abstract product specifications (e.g., "Image Resolution"). For disambiguation reasons, the abstract product specifications of o will be called the set of properties P in the following. The output of the matching step consists of a set of specifications $S_m(x, c)$ (e.g., "Image Resolution = 12.3 MP") being matched with and normalized by c. Finally, since each product specifications source uses its own style to represent specification values, a normalization of specification values is required.

Before examining the matching step in detail, the central concepts of our matching ontology are to be presented since they will be used throughout the whole process.

3.1 Domain Model

The ontology o represents the target schema for matching extracted product specifications $S_r(x)$ and was modeled in OWL. Only the tbox of o is of interest.

Generally, an ontology tbox for the product domain would contain a taxonomy of product types, relations between those types and corresponding product attributes. However, since we introduced the concept of a *property* in our matching process, we created a tbox meta-model being located above the normal tbox model that allows the description of product property attributes, such as the property's structure, type, etc. The meta-model is presented in Figure 3.

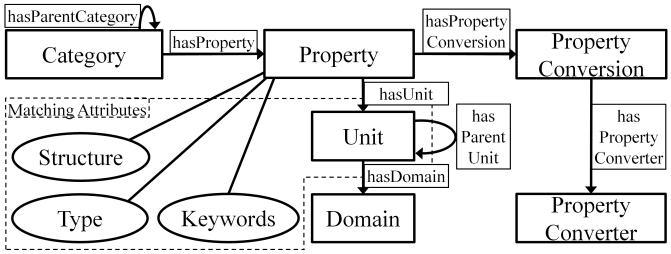

Fig. 3. Domain meta-model for the matching process.

As shown by the figure, the meta-model mainly consists of the mentioned categories C, corresponding properties P, a set of units U and a description of each unit's domain. The lower tbox model contains categories such as "Digital Cameras", properties such as "Image Resolution" or "Image Sizes", and units including "Metre", etc. The relations between those concepts can be seen in the figure as well. Additionally, each property has three different attributes, namely, a structure, a type and a collection of keywords. These attributes as well as the property's unit will be important for the matching process to be presented later on. An additional product conversion attribute is provided that describes how a property can be split up into basic properties or combined to a complex property using a property converter (a dedicated code snippet for executing the conversion process). These two classes are important for the normalization process.

The described meta-model as well as concrete categories and properties for those categories were modeled in OWL and build the tbox of o. For each category, up to 40 properties have been added. Figure 4 shows the specification set of $dc1$ after being matched with and normalized by our ontology.

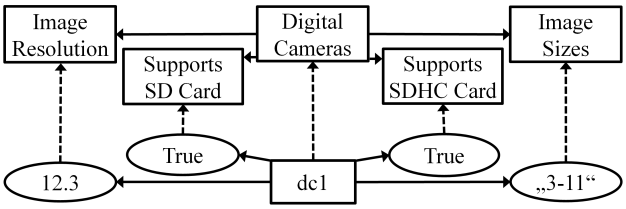

Fig. 4. Examples of matched product specifications for $dc1$ in Figure 1.

3.2 Product Specification Normalization

If an alignment between an extracted specification s_r and a property p has been detected, the correct key of s_r is the one of p. However, if the extracted specification has a complex structure, such as a list, a vector, or a range, contained values are not easily comparable. Thus, the complex specification is split into several elementary and comparable specifications. Furthermore, the contained values are cleaned by removing HTML snippets, deleting nonrelevant information (e.g., information in brackets), distilling numeric values or changing boolean values to "true" and "false", respectively. Furthermore, found units are changed to the current specification's standard unit followed by a recalculation of the associated numeric value (Figure 5).

4 Similarity Measures

In the following, we develop a composite matcher that uses a set of element-level matchers specifically designed for the product domain. During the adoption of those matchers, various thresholds are employed to only select the most robust alignments. Finally, the resulting elementary alignments are aggregated.

We identified five characteristics of technical product specifications that may be used as indications for the matching process. One of them is the specification's key. The other characteristics focus on the specification's value and include the value's structure, the value's type, the value's unit and a collection of keywords potentially contained in the value. The following formula shows how to aggregate the different similarities to create a consolidated similarity value.

$$\begin{aligned} \Phi S_{spec}(s_r, p) = \Theta_{\tau_{spec}}\Big(&\Phi S_{key}(s_r, p) + \Phi S_{struct}(s_r, p) + \Phi S_{type}(s_r, p) \\ &+ \Phi S_{unit}(s_r, p) + \Phi S_{keyword}(s_r, p)\Big) \end{aligned} \tag{1}$$

A function $\Theta_{\tau_{spec}}$ is used to define a threshold. Each of the different similarity functions is to be implemented in its own elementary matcher and will be defined below.

Key Similarity. The key similarity $\Phi S_{key}(s_r, p)$ is to be calculated through a string comparison. The most basic way to compare a specification key and a property key is to check whether both keys are identical. A value of 1 is used as similarity value in that case. If $key(s_r)$ is contained in $key(p)$ or vice-versa, the similarity is the ratio of both values multiplied with a weight between zero and one. Additionally, in the latter case, a threshold function sets the similarity to zero if the contained string is too short. If the keys are not identical and no key contains the other, the Levenshtein distance (normalized by the length of the longer key) is used. Again a weight and a threshold function decide about the overall value. Finally, if neither of both similarities is above zero, alignments

from previous matching tasks are examined for detecting potential transitive mappings. The resulting matcher is therefore string-based and alignment reusing.

For the specifications of $dc1$ in Figure 1, the similarity of "Picture sizes" and the property name "Image Sizes" from our example tbox in Figure 4 could be detected with the Levenshtein function.

Structure Similarity. The first of four value similarity measures is the structure similarity. The structure of a product specification's value can assume four shapes, namely, range, vector, list and scalar. For being able to calculate the structural similarity of a product specification s_r and a property p from the ontology o, an extraction function $E : val(s_r), pat \mapsto val(s'_r), val(s'_r) \subseteq val(s_r)$ is needed that searches for patterns of all the mentioned shapes in a given specification value and extracts each part complying with such a pattern. Then, the length of the extracted pattern match is normalized by the complete value length.

The final similarity measure $\Phi S_{struct}(s_r, p)$ for the value's structure is the maximum of all four values multiplied with the structure's weight. If neither a range, nor a vector or list could be detected, the algorithm assumes to have found a scalar value. Since scalar values are quite meaningless, the similarity is set to zero in such cases. In Figure 1, a list could be detected when examining "Storage Media" or "Picture Sizes" from $dc1$. This would give a hint that "Storage Media" is the complex version of different properties including "Supports SD card". This complex property is not shown in the example tbox. A matcher implementing this similarity measure would belong to the contraint-based class.

Type Similarity. The type similarity $\Phi S_{type}(s_r, p)$ is the second value similarity measure and detects the datatype in a product specification's value. Thus, it is also part of a constraint-based matcher. We identified four types, namely, boolean, float, integer and string. The calculation is similar to the structure similarity measure. Again, by the use of an extraction function, datatype patterns are searched in the specification value. The highest extraction value is multiplied with the found datatype's weight. Since a string type is not that significant, its detection leads to a similarity of zero. Coming back to our example specifications in Figure 1, several datatypes could be detected such as a float value in "Number of effective pixels*1" or some integer values in "Picture sizes".

Unit Similarity. The unit similarity $\Phi S_{unit}(s_r, p)$ is also based on an extraction function that uses patterns for identifying units in given specification values. Since some units are more specific for a potential matching property, the result of the extraction function is multiplied with the found unit's specificity and a corresponding weight. A unit's specificity depends on how many properties in o use this property for their values. The more properties use a particular property, the less specific the unit is for each of these properties.

Product specifications value units may vary even if they belong to the same property (e.g., metres, centimetres, feet). Thus, a unit model has to be included in the product information ontology (Figure 5). The relations between units of

the same domain allow a wider search for unit patterns. The formula's similarity measure is based on the best extraction result for all units related to the checked property while derived units are multiplied with a different weight. The unit similarity matcher belongs to the class of linguistic resources matchers.

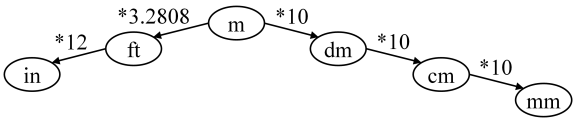

Fig. 5. Relations between length units.

Our running example includes "megapixels" in "Number of effective pixels*1" or "MB" in "Storage Media". These are hints for potentially matching properties.

Keyword Similarity. The last similarity we defined for matching product specifications with properties from an ontology is the keyword similarity $\Phi S_{keyword}(s_r, p)$, also belonging to the class of linguistic resources matchers. For each keyword defined for a corresponding property p, a pattern matching function calculates a relative pattern matching value. The keyword similarity is the sum of all keyword matches. The *dc*2's attribute "Supported Memory Cards" in Figure 1 includes keywords such as "SD" or "SDHC".

Having combined the different matchers for calculating the overall similarity value of a defined property p and an extracted specification s_r as shown above, an alignment $S_m(x, c)$ can be constructed taking the stable marriages problem into account. The normalization is the final step in our process chain.

5 Evaluation

We evaluated our matching process concerning seven different product categories from the electronics domain in a prototype called Fedseeko. First, a test set of products was used to determine the weights and thresholds of our algorithm empirically. Afterwards, 131 products equally distributed over all categories were gathered by a product crawler randomly selecting products from the Amazon portfolio and assembled manually to create a gold standard. Only product specifications being modeled as properties in our ontology o were taken into account. Then, our system tried to automatically execute the necessary matching tasks. Figure 6 shows the precision and recall values for product classification and property matching independently. The figure reveals that with our chosen thresholds the system did not always classify the given products (71.8%), but indeed for those 71.8% of the product set always chose the correct product category. The overall property matching gained quite similar values. Lower thresholds might have improved the recall for both the classification of products as well as the matching of extracted specifications at the expense of a worse precision.

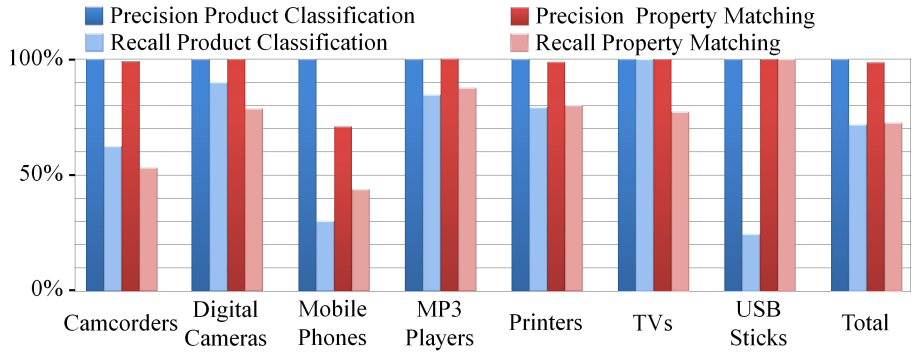

Fig. 6. Evaluation of precision and recall for classification and matching.

Figure 7 displays the F-measure. Additionally, the quality of product specification normalization is displayed. The overall F-measure values account 83.6% for the classification and matching, 92% for the normalization and 64.3% for the overall process. This value falls out of alignment since the normalization step dependends on a successful execution of all previous steps.

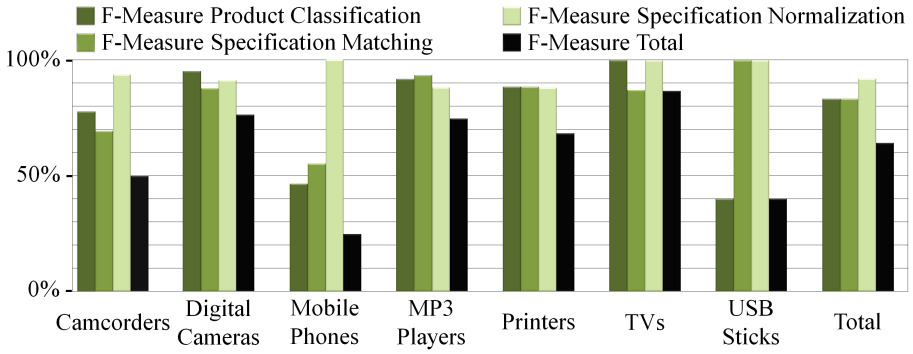

Fig. 7. Evaluation of F-measure values for the whole process.

The developed approach was mainly employed for creating specification sets of electronic end-consumer products since for those products specifications can generally be found online. From the chosen categories, mobile phones and USB sticks showed the worst results. This is mainly due to the incomplete sets $P(\text{"}MobilePhones\text{"})$ and $P(\text{"}USBSticks\text{"})$ which impair the quality of our classification process. Missing keywords and units of existing properties have a negative impact on the specification matching itself. Thus, a deliberately modeled, fully-fledged ontology would already improve the evaluation results significantly.

6 Conclusions

In our paper, we presented an approach for matching and normalizing technical product specifications. The capital contribution of our work is a collection of similarity measures for detecting alignments. These measures are important since they help to compensate the missing hierarchy in technical product specifications. The evaluation proved the suitability of our approach. However, automating the calculation of employed weights and thresholds would certainly improve the flexibility of our system as well as overall results. A feasible approach would be the adoption of semi-supervised or unsupervised learning techniques. In addition, language-based matchers using NLP and a dynamic adaptation of the matching process could further enhance the evaluation results.

References

1. Euzenat, J., Shvaiko, P.: Ontology matching. 1 edn. Springer (2007)
2. Peukert, E., Berthold, H., Rahm, E.: Rewrite techniques for performance optimization of schema matching processes. In: Proceedings of the 13th EDBT, ACM (2010) 453–464
3. Shvaiko, P., Euzenat, J.: A survey of schema-based matching approaches. Journal on Data Semantics **4** (2005) 146–171
4. Do, H.H., Rahm, E.: Coma - a system for flexible combination of schema matching approaches. In: Proceedings of the 28th VLDB, VLDB Endowment (2002) 610–621
5. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic schema matching with cupid. In: Proceedings of the 27th VLDB, Morgan Kaufmann Publishers Inc. (2001) 49–58
6. Giunchiglia, F., Shvaiko, P., Yatskevich, M.: S-match: an algorithm and an implementation of semantic matching. In: Proceedings of the 1st ESWS. (2004) 61–75
7. Euzenat, J., Valtchev, P.: Similarity-based ontology alignment in owl-lite. In: Proceedings of the 16th ECAI, IOS Press (2004) 333–337
8. Ehrig, M., Sure, Y.: Ontology mapping - an integrated approach. In Bussler, C., Davies, J., Fensel, D., Studer, R., eds.: The Semantic Web: Research and Applications. Volume 3053 of LNCS. Springer (2004) 76–91
9. Ehrig, M., Staab, S.: Qom: Quick ontology mapping. In: Proceedings of the 3rd ISWC, Springer (2004) 683–697
10. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: Proceedings of the 18th ICDE. (2002) 117–128
11. Bozovic, N., Vassalos, V.: Two-phase schema matching in real world relational databases. In: Proceedings of the ICDE Workshops. (2008) 290–296
12. Berlin, J., Motro, A.: Database schema matching using machine learning with feature selection. In: Proceedings of the 14th CAiSE, Springer (2002) 452–466
13. Doan, A., Madhavan, J., Dhamankar, R., Domingos, P., Halevy, A.: Learning to match ontologies on the semantic web. The VLDB Journal **12**(4) (2003) 303–319
14. Walther, M., Hähne, L., Schuster, D., Schill, A.: Locating and extracting product specifications from producer websites. In: Proceedings of the 12th ICEIS, INSTICC (2010)
15. Walther, M., Schuster, D., Juchheim, T., Schill, A.: Category-based ranking of federated product offers. In: Proceedings of the 8th WWW/Internet, IADIS Press (2009)