Noname manuscript No.
(will be inserted by the editor)

A Reverse Engineering Approach for Automatic Annotation of Web Pages

Roberto De Virgilio[1], **Flavius Frasincar**[2], **Walter Hop**[2], **Stephan Lachner**[2]

the date of receipt and acceptance should be inserted later

Abstract The Semantic Web is gaining increasing interest to fulfill the need of sharing, retrieving, and reusing information. Since Web pages are designed to be read by people, not machines, searching and reusing information on the Web is a difficult task without human participation. To this aim adding semantics (i.e meaning) to a Web page would help the machines to understand Web contents and better support the Web search process. One of the latest developments in this field is Google's *Rich Snippets*, a service for Web site owners to add semantics to their Web pages. In this paper we provide a structured approach to automatically annotate a Web page with Rich Snippets RDFa tags. Exploiting a data reverse engineering method, combined with several heuristics, and a named entity recognition technique, our method is capable of recognizing and annotating a subset of Rich Snippets' vocabulary, i.e., all the attributes of its *Review* concept, and the names of the *Person* and *Organization* concepts. We implemented tools and services and evaluated the accuracy of the approach on real E-commerce Web sites.

Keywords RDFa, Rich Snippets, DRE, Web Site Segmentation

1 Introduction

Ever since Tim Berners Lee presented, in 2006, the design principles for Linked Open Data[1], the public availability of Semantic-Web data has grown rapidly. Today, practitioners, organizations and universities are all contributing to the *Web of Data* by building RDF repositories either from scratch or by publishing data stored in traditional formats such as relational databases and HTML documents [1]. In addition, ontology languages such as RDFS and OWL (with all their relatives and

[1]Dipartimento di Informatica e Automazione
Universitá Roma Tre, Rome, Italy
E-mail: devirgilio@dia.uniroma3.it
[2]Erasmus University Rotterdam
Erasmus School of Economics
PO Box 1738, NL-3000 DR, Rotterdam, The Netherlands
E-mail: {w.w.hop, s.lachner}@student.eur.nl, frasincar@ese.eur.nl

[1] http://linkeddata.org/

versions) support this trend by providing a means to annotate Web data with metadata enabling different forms of reasoning, depending on the expressiveness of the adopted language.

In this scenario, the *Semantic Web* [2] is an extension to the World Wide Web in which information is defined semantically (as concepts with meaning) instead of presented visually. It will allow the Web to match requests of people and machines to Web content in a more accurate way. Although it may unfold in interesting new functionality involving finding, sharing, and combining information on the Web, wide adoption of the Semantic Web is yet to be waited for.

One of the latest developments in this field is Google's *Rich Snippets* [3], a service for Web site owners to add semantics to their (existing) Web pages using the Google's vocabulary [4] (i.e. a list of concepts and their attributes). Although the existing vocabulary is limited to a small number of simple concepts (i.e *Person, Review, Review Aggregate, Product,* and *Organization*) it is likely only a matter of time before new concepts will be introduced. Fig. 1 shows an example of a Rich Snippet in Google's search results. When a Web site uses Rich Snippets on its pages, recognized concepts will be highlighted in Google's search results using visual cues and a brief display of the concepts' attributes.

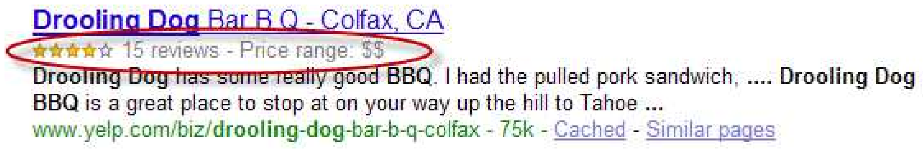

Drooling Dog Bar B Q - Colfax, CA
★★★★☆ 15 reviews - Price range: $$
Drooling Dog has some really good BBQ. I had the pulled pork sandwich, ... Drooling Dog BBQ is a great place to stop at on your way up the hill to Tahoe ...
www.yelp.com/biz/drooling-dog-bar-b-q-colfax - 75k - Cached - Similar pages

Fig. 1 An example of a *Review Aggregate* Rich Snippet in Google search results.

Since a highlighted and more explanatory result will stand out in long uniform lists of search results, it is hoped that this feature will incentivize Web site owners to start using Rich Snippets on their Web sites. Future usage of annotated Web pages is not limited to displaying Rich Snippets in search results. It is only a small step to introduce more advanced search capabilities. For example, you might search for Web pages about the company "Philip Morris" or the programming language "Java", while ignoring Web pages about unrelated entity types (such as persons and geographical regions) with the same name. Another example would be to query Google for products sold in a certain price range with positive reviews.

The success of Rich Snippets depends on the support of search engines on one hand, and the coordinated adoption by a loosely-knit community of Web site owners and software companies on the other hand. Although Rich Snippets were introduced quite recently (i.e. May 12, 2009), it appears that Google seriously commits itself to a future of semantic search. For Web site owners however, adopting Rich Snippets still requires a considerable effort. If a Web site owner retrieves their information as structured data from a database, annotating pages with Rich Snippets is a simple exercise, as it is sufficient to identify its concepts in generated pages and add attributes to the generated HTML output. Fig. 2 shows an example of semantic annotation of a *Review* entity in a Web page. It is supported by the RDFa [5] format.

Nevertheless, automatic pre-generation of annotations is not always possible. Problems appear for instance when information is not available in structured databa-

```
<span xmlns:v="http://rdf.data-vocabulary.org/#" typeof="v:Review">
   <span property="v:itemreviewed">Komala Vilas</span>
   <span property="v:reviewer">Meenakshi Ammal</span>
   <span property="v:rating">3.7</span>
   <span property="v:dtreviewed">1st April 2005</span>
   <span property="v:summary">Best south Indian vegetarian food in South Bay</span>
</span>
```

Fig. 2 A Rich Snippets annotation of a *Review* entity, using the RDFa format.

se form, or when the Web site owner does not have full control over the Web page generation process. Other problem instances arise if the site owner must start from static documents such as those originating from legacy software systems or OCR methods. Semantically annotating these Web pages may then become a matter of time-consuming manual labor.

In this paper, we define a Data Reverse Engineering (DRE) process to read existing Web pages and automatically annotate them with the necessary RDFa attributes defined by Google Rich Snippets. We have explored recognizing the Rich Snippets concepts *Reviews*, *People* and *Organizations*. Therefore we employed a DRE algorithm [6] and Named Entity Recognition (NER) techniques, implementing a tool that takes a URL as an input, and outputs a Google Rich Snippets-compliant Web page. Our approach is related to recent techniques for extracting information from the Web [7]. As for most of these proposals, we start from the observation that data published in the pages of large sites usually (i) come from a back-end database and (ii) are embedded within shared HTML templates. Therefore, the extraction process can rely on the inference of a description of the shared templates. Although this approach is applicable on Web documents, it does not exploit the hypertext structure of Web documents. Our work focuses on discovering this structure as well. Some research efforts show that users always expect that certain functional parts of a Web page (e.g., navigational links, advertisement bars and so on) appears at certain positions of a page[2]. Additionally, there exist blocks of information that involve frequent HTML elements and have a higher coherence. That is to say, in Web pages there are many unique information features, which can be used to help the extraction of blocks involving homogeneous information. Once extracted, such informative blocks are labeled by NER methods supporting recognizing the Rich Snippets concepts. The overall process results completely automatic.

The remainder of this paper is organized as follows. Section 2 introduces the state of the art. Section 3 illustrates an architecture of reference to provide a functional global view of the approach. Section 4 describes step-by-step the annotation process. Section 5 discusses the implementation of our framework and evaluates the performance of our algorithm. Finally, in Section 6 we outline conclusions and future work.

2 Related Work

Named Entity Recognition. Our work is related to a research field called Named Entity Recognition (NER). NER aims at processing natural text and identifying certain occurrences of words or expressions as belonging to particular categories of

[2] For more details see http://www.surl.org/

named entities [8]. These named entities belong to predefined entity types (categories) such as persons, organizations, locations, expressions of time, etc. Although most techniques rely on gazetteers (lists of names of people, organizations, locations, and other named entities), NER is not simply a matter of searching text for known words. Such a word list would be enormously long and not feasible to compose. Moreover, certain words can belong to multiple concepts of different entity types. The text "Philip Morris" might be in the list of names as well in the list of companies, leading to an ambiguity issue. Therefore, NER involves more advanced techniques.

Most approaches to NER problems can be classified as statistical, grammar-based, or hybrids of these two approaches. Statistical systems typically make use of annotated training data from which word lists are generated and features are selected. The classifier then uses a statistical model to compute the probability of a word belonging to one of the output classes based on its context. As mentioned above, NER usually depends on extensive gazetteers. However, there are researchers that argue against the use of large gazetteers. Participants in the Named Entity recognition competition (part of MUC-6) report that gazetteers did not make a significant difference to their systems [9,10]. In [8] the authors avoid large gazetteers by combining rule-based grammars with statistical (maximum entropy) models. They show that it is sufficient to use relatively small gazetteers of well-known names, rather than large gazetteers of low-frequency names.

Source code analysis. Our project makes use of NER techniques to recognize and label the names of *Person* and *Organization* entities, two basic concepts from Google's Rich Snippets vocabulary. However, not the recognition of every entity that is in the Google vocabulary can be reduced to a NER problem. Review text bodies for example won't let themselves be captured by applying NER. To be able to automatically recognize reviews on a Web page, a set of rules or patterns to extract a review has to be found. This so-called 'pattern-matching' on a Web page can be done either by inspecting the source code of the page or by analyzing linguistic properties.

Ranking text sections on a Web page in terms of importance is an important topic, popularized by search engines like Google, which uses HTML tags such as `h1`, `h2`, etc. to determine the relevance of information found on a Web page. This simple heuristic is also useful to find summaries and titles of text bodies. While actual algorithms used by search engine companies remain unpublished, we have implemented a tag ranking heuristic based on current assumptions by search engine specialists [11]. An example of more extensive source code analysis is the work of De Virgilio and Torlone [6]. In this study, the authors reverse engineer data that is contained in (data-intensive) Web pages by analyzing the structure of that page and how it would be visually formatted on a screen. The underlying idea is that semantically identical data is mostly displayed in visually grouped object blocks. Another example in this field is recognizing a postal address from a Web page. An interesting algorithm, based on first assessing visual similarity of Web page elements and then using a grammar-based approach, is sketched in [12]. This method could be helpful in detecting postal addresses of *Person* and *Organization* entities.

Search engines. Google Rich Snippets is not the only service aimed at integrating semantic information into search engines. A similar initiative is Yahoo! SearchMonkey, a service that also traverses Web sites to find RDFa or Microformats annota-

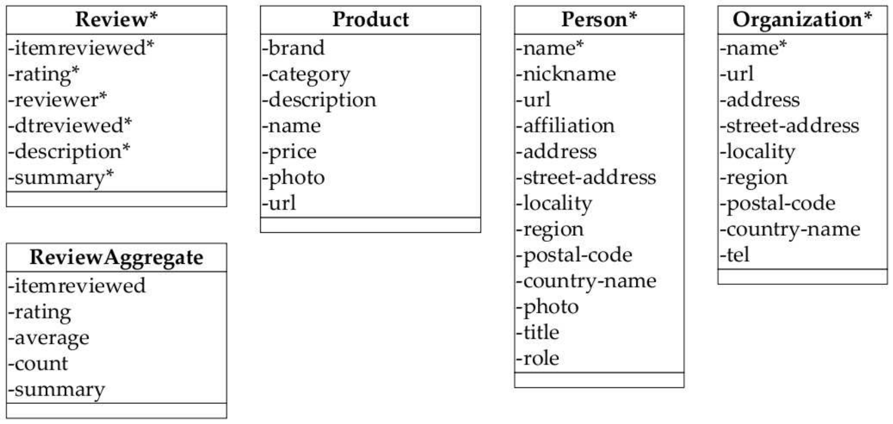


Fig. 3 Vocabulary supported by Google Rich Snippets. Entities and attributes marked with an asterisk (*) will be annotated by our method.

tions for concepts such as Reviews, Persons and Organizations. Additionally, it allows Web site owners to create applications that build "enhanced results" using this information [13]. Fortunately, Google Rich Snippets and Yahoo! SearchMonkey have overlapping vocabularies which both include the entities recognized by our method, which means that the resulting annotated Web pages can be interpreted by both search engines.

3 The overall process

Google Rich Snippets supports a vocabulary concerning Reviews on different Products, as shown in Fig. 3.

Our main focus is on recognizing Review entities and their attributes. In particular in detail we extract the following subset of entities and attributes from the vocabulary

$$\begin{aligned} \textsc{Review} &= (itemreviewed, rating, reviewer, dtreviewed, description, summary) \\ \textsc{Person} &= (name) \\ \textsc{Organization} &= (name) \end{aligned}$$

Our framework for automatically adding Google Rich Snippets annotations to a Web page is composed of a number of stages. Fig. 4 sketches the main steps of our framework and their interdependencies.

The process starts with a *Preprocessing step* to clean and to make uniform the (X)HTML code of Web pages. Unfortunately, currently a large number of Web pages are invalid (more than 50% in some survey [14]) which may not pose a problem for the human user but makes automatic processing harder. Further, Web pages can be transmitted in various encodings. All encodings should be converted to a common format, i.e. UTF-8, in order to allow uniform processing.

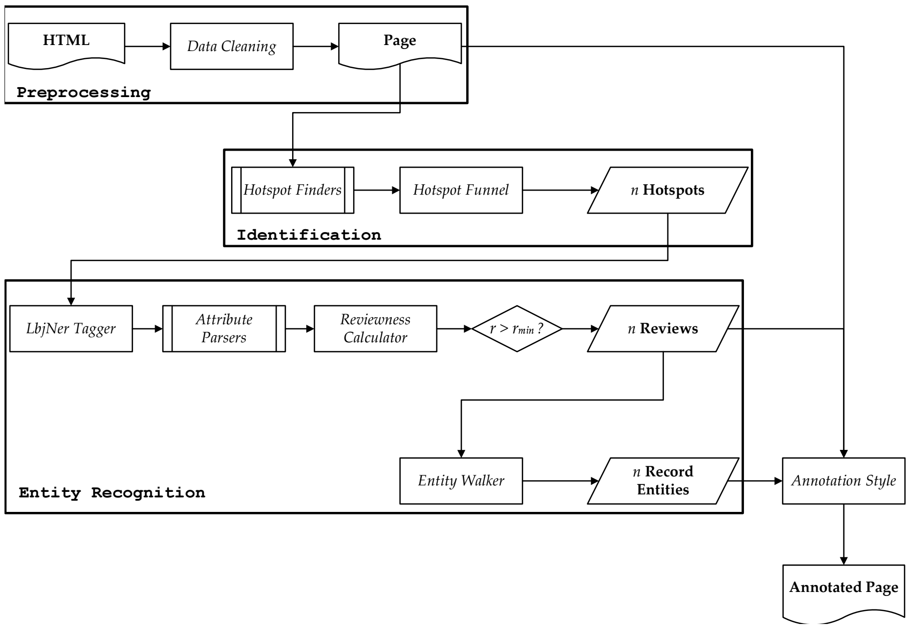


Fig. 4 Stages of our recognition algorithm, their interdependencies and intermediate products.

Requirements to the algorithm are a high sensitivity (low false negative rate), good specificity (low false positive rate), and reasonable robustness (i.e. the ability to be used on a variety of Web pages that do not match the trained set). Of high importance therefore is the task to choose the parts of a Web page to investigate. We use the term "*hotspot*" to identify an area of a page that is — before intensive analysis — most likely to be a place of interest (e.g a document section containing a product review to annotate). To this aim an *Identification* step is responsible to determine such hotspots. It is supported by the segmentation algorithm provided by De Virgilio and Torlone [6] that use a Web data model [15] to describe abstract structural features of HTML pages. Combining cognitive visual analysis and hierarchy-based algorithms, they identify blocks grouping semantically related objects occurring in Web pages, and generate a logical schema of a Web site. Exploiting such segmentation, we select blocks of interest, corresponding to the most relevant hotspots, and introducing effective heuristics we filter false positive portions of page. Then the *Entity Recognition* phase exploits a named entity recognition algorithm to extract entities and attributes of pages matching the portion of Review vocabulary discussed above. Finally, the *Annotation* takes place supported by RDFa format to model the annotations in the page.

In the following section we will describe in detail the entire process in terms of both identification of hotspots and entity recognition.

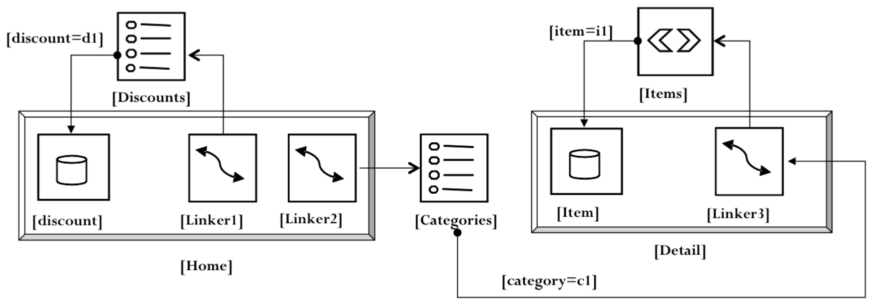


Fig. 5 The WSM representation of a commercial Web Site

4 Automatic annotation of Web pages

4.1 Hotspot identification

Our need to identify hotspots on a Web page starts first from the necessity to demarcate recognized entities. A recognized review must have a start and end location that together span over all its properties. Also, we may want to perform transformations or computationally expensive analyses on texts. It is not necessary to know for sure if a hotspot definitely corresponds to a review: it is not problematic if we are too eager in recognizing an element, but it is unrecoverable if we now skip an element for further processing.

Following the approach in [6], we have to select portions of an HTML pages and map them to *constructs* of the Web Site Model (WSM) [15]. WSM defines a conceptual modeling of the main structures occurring into a Web page. More precisely, it organizes a page in a set of *meta containers* related by *links*. A metacontainer is *atomic* if it represents an atomic portion of a linked page and includes a direct reference to the elements of a content from which it takes data. It can be *basic* if it shows information about a single object (e.g. an instance of an entity) or *multi* if it shows information about a set of objects. A metacontainer is a *linker* if it contains an anchor between containers. Otherwise a metacontainer is *complex* if it is articulated in other containers (atomic, complex or linker). So a page represents a complex container. We can surf the containers through several *navigational structures* such as *Indexes* to show a list of objects without presenting the detailed information of each one, *Guided Tours* to show commands for accessing the elements of an ordered set of objects, and *Entries* to show edit field for inputting values used for searching within a set of objects meeting a condition. For instance, Fig. 5 shows a commercial Web site about several discounted items grouped in categories, represented in the WSM model. In the Figure, `Home` and `Detail` correspond to complex metacontainers. In particular `Home` describes different discounted items as basic containers (i.e. `discount`) that can be navigated by an Index structure (i.e. `Discounts`).

Implementing the segmentation algorithm of [6], we determine several blocks into a Web page identified by the HTML tags pattern to reach them. As in [6], the idea is to identify a set of *container tags* representing candidates to be mapped. In particular we refer to HTML tags that bring to information content in a Web page such as `UL, TABLE, DIV, BODY, ...`. We call *non informative tags* HTML tags that don't bring to informative content (such as `SPAN, B` and so on). Each pattern rooted in a tag container will be translated into a metacontainer using a particular navigational structure (Index, Guided Tour or Entry). Let's consider to define a *target pattern* t for each construct. For instance the Index navigational structure presents `UL-LI-A` as target pattern t_{Index}. If we want to map a source pattern s to t_{Index} we have to find an alignment between s and t_{Index}. To this aim we will use the dynamic programming algorithm to calculate the optimal score and to find the optimal alignment between two patterns (i.e. alignment between two strings) [16].

First, we will compute the optimal alignment for every substring and save those scores in a matrix. For two strings, s of length m and t of length n, $D[i, j]$ is defined to be the best score of aligning the two substrings $s[1 \ldots j]$ and $t[1 \ldots i]$. A scoring matrix for scoring substitutions, matches, and gap creation is needed. The score is 2 for a match, 1 for a partial match and -1 for a mismatch. The match is based on tags comparison between patterns. A match (mismatch) is between container tags while a partial match is between a container tag and a non informative tag. Then we will consider global alignments: the conditions are set such that we compute the best score and find the best alignment of two complete strings. Therefore the best score for the alignment is precisely $D[m, n]$, the last value in the table. We will compute $D[m, n]$ by computing $D[i, j]$ for all values of i and j where i ranges from 0 to m and j ranges from 0 to n. These scores, the $D[i, j]$ are the optimal scores for aligning every substring, $s[1 \ldots j]$ and $t[1 \ldots i]$.

In general, dynamic programming has two phases: the forward phase and the backward phase. In the forward phase, we compute the optimal cost for each subproblem. In the backward phase, we reconstruct the solution that gives the optimal cost. For our string matching problem, specifically, we will: (i) use the recurrence relation to do the tabular computation of all $D[i, j]$ (forward phase), and (ii) do a traceback to find the optimal alignment.

The recurrence relation establishes a relationship between $D[i, j]$ and values of D with indices smaller than i and j. When there are no smaller values of i and j then the values of $D[i, j]$ must be stated explicitly in the base conditions. The base conditions are used to calculate the scores in the first row and column where there are no matrix elements above and to the left. This corresponds to aligning with strings of gaps. After the base conditions have been established, the recurrence relation is used to compute all values of $D[i, j]$ in the table. The recurrence relation is:

$$D[i, j] = max\{D[i-1, j-1] + sim\_mat[s[j], t[i]], D[i-1, j] + gap\_score, D[i, j-1] + gap\_score\}$$

In other words, if you have an optimal alignment up to $D[i-1, j-1]$ there are only three possibilities of what could happen next: (1) the characters for $s[i]$ and $t[j]$ match, (2) a gap is introduced in t and (3) a gap is introduced in s. It is not possible to add a gap to both substrings. The maximum of the three scores will be chosen as the optimal score and is written in matrix element $D[i, j]$.

The second phase is to construct the optimal alignment by tracing back in the matrix any path from $D[m, n]$ to $D[0, 0]$ that led to the highest score. More than one possibility can exist because it is possible that there are more than one ways of achieving the optimal score in each matrix element $D[i, j]$. In our case we have to select the target pattern that presents the best scoring matrix (i.e. the best alignment with the source pattern s).

For instance consider the pattern s, that is `UL-LI-DIV-A`. We have to map s into a metaconstruct of WSM. We have to find an alignment between s and a target pattern t. The best alignment is with the target pattern `UL-LI-A`. To this aim, we obtain the following scoring matrix.

t/s	UL	LI	DIV	A
UL	*2*	*-1*	*1*	*-1*
LI	*-1*	*2*	*1*	*-1*
A	*-1*	*-1*	*1*	*2*

The best alignment is `UL-LI-#-A`, therefore we will map the pattern s with an Index structure of WSM.

What then, should be considered a hotspot? First and foremost, any WSM element that contains a reasonable amount of natural text qualifies as a hotspot. However, this might miss very short or long elements; therefore we can also use cues such as HTML *name, id* and *class* attributes of elements, or their textual contents. It is simple to realize that an instance of the concept *Review* will most likely have a visual cue to the reader indicating this, or a descriptive element name on the part of the Web page designer. We can match these naively with a word list. To discover reviews we look for the terms `review` and `rating`. Other heuristics that can be used to find hotspots are the presence of repeating element structures, which are an indication of automatically generated content in a page, or similarity of HTML element *id* attributes (for instance, a common prefix or Levenshtein distance). However, we have not pursued these heuristics in our implementation because the first two tests already gave a very good coverage. After that, we must remove duplicates, as multiple hotspot finders will likely trigger on the same hotspots. Every hotspot finding measure increments the "hotness" of a hotspot. This measure is retained for subsequent filtering of hotspots before further processing.

A large portion of a Web page is limited to non-visual and non-textual items such as navigation sections, styling information, advertisements, etc. To the human user, these are immediately distinct from textual content, but to automated systems this may not be so clear. For instance, navigations certainly contain many terms that turn up in our cue word lists. We therefore introduce the measure *tag ratio* for a DOM node, which we define in Equation 1:

$$t_r = \frac{L_H - L_N}{L_H} \tag{1}$$

where L_H is the total character length of the DOM node and its descendants including all the HTML tags; and L_N is the character length of all natural text contained within the node. L_H is directly taken from the HTML document, while L_N is constructed by removing all the HTML tags, normalizing whitespace characters such as newlines and tabs to single spaces, and then trimming the output. An example of these measures is presented in Fig. 6.

If a hotspot has a high (near 1) value of t_r, then the element consists almost entirely of HTML tags. This is uncommon for textual content, so we should disqualify the element for further processing. A reasonable threshold value τ_{max} must be determined empirically from test data. At the same time, we want to stop false positive recognition of too short texts that stand on their own in the page. For instance,

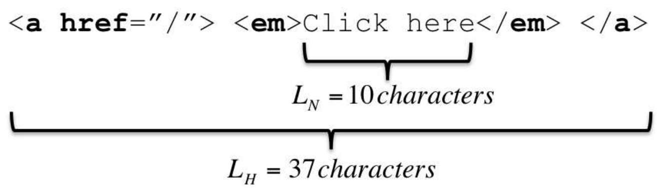


Fig. 6 Example of L_H and L_N measures of an HTML fragment.

a page title or caption may contain some very on-topic terms, causing it to be recognized as a hotspot. We will disqualify hotspots that contain less than a minimum amount of natural text, L_{min} (number of characters). In a similar fashion we will be throwing away hotspots that are displayed as *inline* (with respect to text) by a browser, i.e., they are part of a natural sentence flow. Examples of these are `a` or `b` tags. A hotspot demarcated by these elements is part of a larger *block*-level element and is not a distinct page area in its own right. In these cases, we expect the outer element to be recognized as a hotspot as well. Since we have multiple methods to find hotspots, we filter next on "best hotness". Different Web pages may conform more or less to our different hotspot-finding heuristics, but within a *single* Web page, different entities of the same type generally have the same hotness measures, especially when the Web pages are generated automatically. When additional hotspot finders would be added, this filter step could be changed to require for instance 75% of the maximum hotness found on the whole page, in order to make this step more robust. An important property of hotspots for a certain entity type is finally that they are disjunct. Only one review entity can be an "active" hotspot at any location in the document. As a Web page document is a tree, this means that hotspots cannot be contained in each other. For instance, it is thinkable that we would consider two consecutive page elements to be hotspots, but at the same time consider their combined parent element a hotspot as well, as it certainly matches most of the criteria that hold for its descendants. In these cases, we should throw away all the "super-hotspots" that contain other hotspots, so that only the most minimal valid hotspots remain. After this stage ends, we have identified a number of hotspot page elements that may correspond to review entities. We will now inspect these elements more closely.

4.2 Named entity recognition

In the context of Google Rich Snippets' vocabulary, NER appears mostly useful in discovering the names of reviewed items and review authors. Similarly, NER can be used to discover names of person and organization entities.

We have experimented with adding more knowledge to an existing named-entity tagger by training it on review texts containing product names. These product names ideally should form a new entity type for the tagger. Merging them into an existing model encounters the nontrivial problem of "transfer learning", i.e., first training the model on one dataset and then trying to add more knowledge to the existing model. To update the models of the tagger, we would have to retrain the tagger from scratch using its original datasets in order to retain its original usefulness.

Tag name	Importance
`h1`	100
`h2`	90
`h3`	80
`h4`	70
`h5`	60
`h6`	50
`strong`	10
`b`	10

Table 1 Relative tag importance. When ranking two tags, only the *order* of the tags' scores is considered. The *absolute* values are not significant, but are chosen in a way that more tag rankings can later be added.

A complicating feature is that product names are often not mentioned in natural text portions of Web pages, but rather in separate page sections, which prohibits using natural language-based methods. From our testing data, it appears that named entity recognition alone is often not successful to determine names of reviewed items and review authors. The nature of review texts on the Web is such that often the name of the reviewed item is not mentioned at all. Therefore, additional heuristics are needed to provide better coverage for detection of the name of the reviewed item and the review author. These heuristics may use data from the NER phase, the review text itself, the Web page source code and its properties such as the page title.

As defined in the Google Rich Snippets summary, reviews have some predefined attributes, such as the *reviewed item, summary, rating, date* and *review author*. In our approach, each of these attributes has a separate attribute parser which follows its own heuristics.

Summary attribute. *Tag ranking* is a method that we use to discern the document element containing the most important title, heading, or summary of a page section. We rank element tags first according to their relative importance, which is modeled as shown in Table 1. If two elements are tied, we rank them using their position on the page, where higher positioned items have a better rank. This strategy corresponds to the approach that Google is currently assumed to use when it ranks the importance of information in a Web page for inclusion in its search index [11].

Author attribute. The author of a review is often supplied on the page within the review element. We employ two strategies for finding the author's name. First, we walk through the DOM subtree of the review to find any elements that most likely contain a name or nickname of the review author. We look for tags with HTML *class*, *id* and *name* attributes matching one of the following strings:

- `author`
- `username`
- `reviewer`

If this approach does not yield a positive result, we inspect the review text for named entities of type *Person*. This entity type has been recognized by the Named Entity Recognition stage earlier. We expect this approach to be more error-prone, as for instance other person entities may become incorrectly recognized as the author.

Product name attribute. The product name attribute contains the name of the reviewed item. During development of the methodology, we found that reviews on Web sites generally do not contain the name of the reviewed item. Often, a review Web page contains a variable number of reviews, while the product name is only mentioned once. As discussed earlier, training a NER tool to recognize product names as a proper entity class is outside the scope of this work. Therefore we derive the product name from the Web page title. The page title often contains unnecessary extra information, such as the name of the Web site and a descriptor text like "Product reviews". These texts should not be present in the product name. Therefore, we remove some strings from the result:

– variants of the name and the domain name of the Web URL of the Page (e.g., *amazon*, *amazon.com*, *www.amazon.com*);
– stop words such as `reviews` and `product`;
– separator characters such as `: - |`

Rating attribute. A pattern matching approach is used to discover the rating of a review, such as "4 out of 5". Recognizing ratings can be problematic, as there is no common standard for their notation; for instance, one site may use a 10-point numerical scale instead of Rich Snippets' default 5-point scale, while another site may use a graphical "stars" definition that usually embeds some kind of reference in the *img src* attribute. If the text matches a list of predefined regular expressions, such as:

– `4.0 out of 5.0`
– `4.0 / 5.0`

we are able to recognize the rating as well as the scale. If we cannot recognize the scale, we assume the lowest of a 5-point, 10-point, and 100-point scale, such that the rating is lower than or equal to the scale maximum. Without resorting to site-specific hints, it is expected that this approach will likely not be very robust or generalizable. At the same time, it will be possible to recognize multiple similar attributes within an entity's boundaries (e.g., two person names or two date strings), and we need to have a tie-breaking algorithm for which we currently do not have a method. At the moment, we use the naive strategy of taking the first occurrence as the most authoritative, but it is likely that there will be false positives.

Date attribute. The date of an entity can be gathered by using a series of regular expressions for common date formats. This list of regular expressions is now focused towards a range of date descriptions found in test Web pages, such as:

– `1-11-2009`
– `1 Nov 2009`
– `November 1st, 2009`

This list of date formats might be broadened to include phrases such as "3 months ago". Note that there are some ambiguities in general date formatting ("1-11-2009" might be in M-D-Y or D-M-Y notation). Google Rich Snippets does not pose any requirements to this format, so we simply retain the date as it was found on the page and leave the ambiguity to the interpreter.

4.3 Reviewness filtering

As we have discussed, the strategy during hotspot determination must be sufficiently eager to provide a wide selection of elements to process using the methods described above. After we have analyzed the elements further, they should now be annotated with various semantical attributes. In case an element slipped by the hotspot funnel that is however clearly not a review entity, it will most likely not have attributes such as a rating, date, summary or author. We use this property to perform a calculation of a review's "reviewness" r. This measure corresponds to the *number* of semantical attributes that have been recognized, excluding the product name attribute, as that attribute is derived from the page title and therefore its recognition always succeeds. Any element not satisfying the basic requirements of a review — in our current model, this is only the presence of a product name — receives a negative reviewness. If the basic requirement is met, reviewness starts out at zero, and one point is added for every semantic attribute that was successfully bound to it. After the calculation step, there is a final filtering step. Any reviews which do not satisfy a minimal reviewness of r_{min} will be ignored. The value of this parameter must be determined during testing.

4.4 Collecting record-based entities

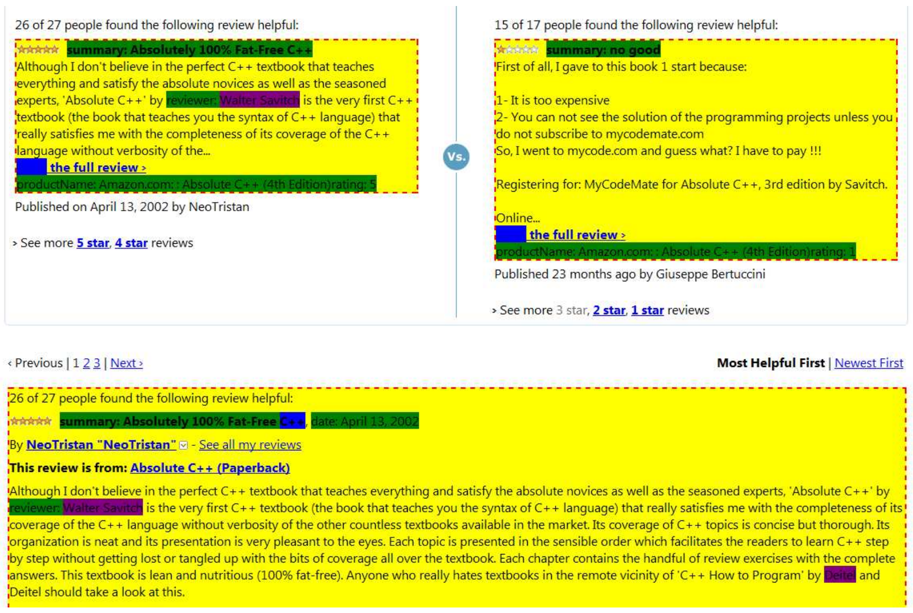
During the main process of recognizing entities, the named entity recognizer (NER) has been run on all the reviews. In the final recognition stage, the names of Person and Organization entities are harvested from the review texts. For record-based entities (Persons and Organizations), in our current method only the names of the entities (and not their attributes, such as phone numbers, postal addresses, etc.) are discovered. The names of these entities are simply retrieved from the NER output as we incorporate a NER tagger that supports these entity types by default.

4.5 Annotation

RDFa[3] is one of the two supported annotation styles in Google Rich Snippets; the other is Microformats[4]. The simpler Microformats style uses HTML *class* attributes populated with conventional names for certain properties, which has the advantage of being usable for Web page formatting and easy to write for humans. However, these advantages are largely irrelevant for our purpose. RDFa [5] benefits from RDF, the W3C's standard for interoperable machine-readable data. RDFa is considered more flexible and semantically rich than Microformats [17]. Additionally, Rich Snippets in RDFa allow for extended functionality, such as adding URL links to Rich Snippet properties. Therefore RDFa is a better choice for meta-data annotation than Microformats.

In addition to RDFa, we found it useful to implement a "layout" annotation style where concept boundaries and attributes are displayed visually using HTML *style* attributes. This aids debugging and makes it easier to quickly assess the algorithm's output on a Web page, as is shown in Fig. 7. We recall that the attributes

3 http://www.w3.org/TR/xhtml-rdfa-primer/
4 http://microformats.org/about



26 of 27 people found the following review helpful:

summary: Absolutely 100% Fat-Free C++

Although I don't believe in the perfect C++ textbook that teaches everything and satisfy the absolute novices as well as the seasoned experts, 'Absolute C++' by reviewer: Walter Savitch is the very first C++ textbook (the book that teaches you the syntax of C++ language) that really satisfies me with the completeness of its coverage of the C++ language without verbosity of the...

the full review ›

productName: Amazon.com: : Absolute C++ (4th Edition)rating: 5

Published on April 13, 2002 by NeoTristan

› See more 5 star, 4 star reviews

15 of 17 people found the following review helpful:

summary: no good

First of all, I gave to this book 1 start because:

1- It is too expensive
2- You can not see the solution of the programming projects unless you do not subscribe to mycodemate.com
So, I went to mycode.com and guess what? I have to pay !!!

Registering for: MyCodeMate for Absolute C++, 3rd edition by Savitch.

Online...

the full review ›

productName: Amazon.com: : Absolute C++ (4th Edition)rating: 1

Published 23 months ago by Giuseppe Bertuccini

› See more 3 star, 2 star, 1 star reviews

‹ Previous | 1 2 3 | Next ›

Most Helpful First | Newest First

26 of 27 people found the following review helpful:

summary: Absolutely 100% Fat-Free C++, date: April 13, 2002

By **NeoTristan "NeoTristan"** - See all my reviews

This review is from: Absolute C++ (Paperback)

Although I don't believe in the perfect C++ textbook that teaches everything and satisfy the absolute novices as well as the seasoned experts, 'Absolute C++' by reviewer: Walter Savitch is the very first C++ textbook (the book that teaches you the syntax of C++ language) that really satisfies me with the completeness of its coverage of the C++ language without verbosity of the other countless textbooks available in the market. Its coverage of C++ topics is concise but thorough. Its organization is neat and its presentation is very pleasant to the eyes. Each topic is presented in the sensible order which facilitates the readers to learn C++ step by step without getting lost or tangled up with the bits of coverage all over the textbook. Each chapter contains the handful of review exercises with the complete answers. This textbook is lean and nutritious (100% fat-free). Anyone who really hates textbooks in the remote vicinity of 'C++ How to Program' by Deitel and Deitel should take a look at this.

Fig. 7 Annotation of a Web page using the 'layout' annotation style. Review entities are displayed in yellow; review properties are green (and prefixed with the property name); person entities are purple; organization entities are blue.

of an RDFa-annotated entity must remain within its boundaries (the element annotated with *typeof*). If an entity attribute was defined intrinsically, we know its textual position in the source HTML and we will rewrite an RDFa tag in place. If the attribute does not correspond to an HTML fragment within the entity (for instance when a product name is derived from the page title), we will write the RDFa tag at the bottom of the element. Google Rich Snippets supports annotating graphical ratings by placing *class* and *alt* attributes on a rating image. This feature can be used to insert rating meta-data without affecting the Web page layout, as displayed here:

```
    <img class="rating" src="stars.gif" alt="4 Star Rating: Recommended" />
```

When parsing the page, Google inspects image elements marked with *class* `rating` and parses the *alt* tag in an undisclosed way. We choose not to implement this Rich Snippets feature, as using non-standard annotations outside the RDFa format is detrimental to further automatic processing of the generated document. In these cases, we do not modify the rating image and inject an additional RDFa-annotated property at the bottom of the element. Finally, after annotation, a `base href` HTML tag is injected at the top of the Web page body. This ensures that the resulting Web page can still be displayed using its original images and styling, even if the page is now served from a different location.

5 Experimental Results

Benchmark System. The approach was developed into a tool with a PHP front-end to reach it by Web. It is composed by three modules implementing the three main steps discussed above. The cleaning of seriously invalid (X)HTML is achieved by the PHP's *tidy* support. The hotspot identification module implements the segmentation algorithm provided in [6] and the dynamic programming algorithm discussed in Section 4. For named entity recognition purposes the *Lbj-Based Named Entity Tagger* (LbjNer) [18] was used. LbjNer is now one of the best performing solutions. It reached very promising F-measure (F1) scores. For instance it obtained a F1 score of 85.74 on the MUC-7 test set[5], and 90.74 on the Reuters2003 test set[6]. One of the competitors, the Stanford NER tool, achieved a F1 score of 80.62 and 87.04 on the MUC-7 and Reuters2003 test set, respectively. The LbjNer tagger is capable of recognizing named entities of types *Person, Organization, Location,* and *Misc* in natural English text with a very impressive accuracy. It employs various statistical methods as well as heuristics. The tagger comes pre-trained on a large corpus of data extracted from Wikipedia and other well-known sources. This makes the tagger attractive for reuse in other projects such as these.

For our test, we used a dual quad core 2.66GHz Intel Xeon, running Linux Gentoo, with 4 GB of memory, 2 MB cache, and a 2-disk 1Tbyte striped RAID array.

Design. Overall, an important issue in designing the software was modularity and extensibility. As the work is mostly exploratory in nature, it is critical to be flexible during development, which means that the various parts of the tool should run in relative separation with few dependencies on each other. Therefore, we have split off the various stages of the algorithm into independently working classes. Where multiple strategies are used for a certain stage, such as during hotspot finding, they are implemented in a 'pluggable' fashion so that it is easy to add further heuristics that expand or focus the recognition. In Fig. 4, these stages are marked as a composite process (two vertical lines).

The class `PageAnnotator` takes care of driving the various phases of the algorithm and is useful as a starting point for reading the source code. Separate `Entity` as well as `Attribute` subclasses take care of their own parsing, so that it is tractable to extend the tool with other semantics.

Where empirically established constants are used, such as the maximum tag ratio τ_{max} or hotspot length limits, these are made explicit through class constants. Should it not be possible to find generally acceptable values for them, then a phase could be added in the algorithm to discover them on a per-page basis.

There are various use-cases for a tool such as demonstrated. One might be to use in an off-line fashion to annotate documents on a local computer, another might be to run as a Web service. We have implemented two "front-end" scripts, `cli.php` and `index.php` which run on the command line and a Web server respectively.

Named entity tagger. Statistical and grammar-based named entity recognition methods require natural text as an input in order to perform well. Also, the used named entity tagger *LbjNer* cannot yet handle HTML input, breaking validation of the output. Therefore, we must collect natural text first, and we should collect it from only the hotspots in the document, as these are the places where the NER tagger will work reliably.

[5] http://www.itl.nist.gov/iad/894.02/related_projects/muc/index.html

[6] http://www.daviddlewis.com/resources/testcollections/reuters21578/

Stripping HTML tags to generate a natural text body implies that we no longer have a 1:1 mapping between the tagger result and the original position of the tagged entities in the DOM. We will have to improvise in order to write tags back into our DOM data structure. A problem occurs when there are multiple occurrences of the recognized entity name in the original text: it becomes hard to choose which occurrence the entity points to. As a guess, we will place the entity at the first occurrence of the entity name in the DOM. This is not a perfect strategy, because we may replace a homonym instead of the right occurrence.

The LbjNer tagger has a large startup penalty due to loading of its gazetteers and statistical models. The LbjNer author has worked around this problem by running it as a daemon process controlled by a socket and an additional Perl client script. We provide a separate adapter to connect to this daemon process, which can be chosen at run-time. Since running the tagger on various hotspots separately would entail longer execution times, we coalesce the various hotspots to tag, and send these texts to the tagger in one batch. This requires a small amount of parsing in the LbjNer result interpretation.

Annotation styles. The debate between RDFa and Microformats is still very active [17], and the nature of the Web is such that often the simpler yet less powerful method gets the most traction in the community. Therefore, it is important that the annotation style is abstracted away from the recognition and processing logic. This makes it easy to add a visual annotation style, and should the marketplace decide in favor of Microformats, to annotate documents in that format instead.

Development set. We have developed the tool while testing on a number of Web pages containing product reviews, taken from the well known American shop site *amazon.com*. During the development of the methodology, we have established that a max tag ratio τ_{max} around 0.7 is a good cutoff value that invalidates recognition of most irrelevant page elements such as navigation sections, while retaining detection of most natural text hotspots. We have found that the algorithm has a tendency to over-recognize small page elements (such as simple page titles and captions) as reviews, especially in the absence of proper reviews which would lower the relative hotness of these elements in relation to the real reviews. This was resolved by setting the minimum natural text length L_{min} to 100 characters. The minimum "reviewness" r_{min} in the final filtering step was set to 2, meaning that for a review to be included in the annotation step, it must have a minimum of two recognized properties (such as rating, reviewer, summary and date). We have excluded the product name (reviewed item) from the reviewness calculation, as we almost always infer some product name from the page title.

Performance Evaluation. To assess the performance of the review recognition methodology, the tool was tested on a (non-randomized) selection of English product review Web pages from well known E-commerce sites listed in Table 2. The tested Web pages were not included in the development set which consisted of pages from *amazon.com*. Model parameters had been optimized earlier empirically using the development set.

We count the actual number of reviews on the Web page manually, then present the URL to the tagger. We then review the annotated Web page for:

- the number of correct review recognitions (actual reviews which are recognized by the method);
- the number of false positives (tagger-recognized reviews that were not reviews in the source);
- and the number of false negatives (actual reviews that were missed by the method).

The results of these tests are presented in Table 2.

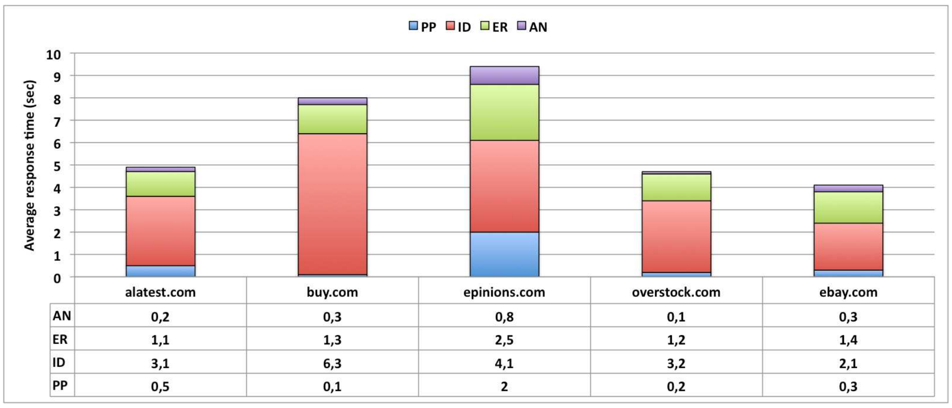
Web site	**Actual reviews**	**Correct**	**False pos.**	**False neg.**
alatest.com	*3*	1	1	1
buy.com	*11*	10	0	1
epinions.com	*15*	0	0	15
overstock.com	*5*	2	0	3
ebay.com	*5*	5	0	0

Table 2 Test results of review recognition. Web site language is English.

It appears that there is a large variance in performance between different Web pages, which is a result of the absence of standardization in laying out Web pages. The algorithm appears reasonably successful, but this property is certainly not ubiquitous. There is even one site, *epinions.com*, where reviews are not recognized at all. On all other sites, the results are reasonable for this first exploration of the field. False positives are almost never found; however there are some false negatives. A closer look into these false negatives suggests that we should explore more methods for finding hotspots, as some slip by the currently implemented hotspot finding heuristics. This happens on sites which do not use our listed cues for element names. Also, some of the model parameters are necessarily a compromise — a more dynamic or fuzzy approach to these parameters may be necessary to provide broader coverage.

In general the tool provides reasonable results, correctly recognizing reviews in review Web pages. We find that, for review detection on *random* (not review-specific) Web pages, our methods provide reasonable sensitivity, but not much specificity. This entails that also on Web pages that do not contain reviews, the heuristics may trigger and the algorithm may unintendedly recognize reviews. It remains a hard issue to test whether a Web page really concerns product reviews. Also of concern is the finding that many reviews do not even mention the name of the reviewed item. Therefore, heuristics for disqualifying a Web page as a review page are at this moment insufficient. This problem is not relevant when our solution is run on review Web pages solely. The required training phase of named entity recognition, as well as the recognition of element names and contents, as we have described in the previous section, possibly limits the application of our method to English Web pages. It is, however, tractable to train the NER tagger as well as revise our internal word lists in order to support other languages.

Finally Fig. 8 shows the average response time, in sec, to annotate one Web page for each Web site. We analyzed 1000 pages for each Web site, and we measured the time to process each step of our approach: the preprocessing (PP), the identification of hotspots (ID), the entity recognition (ER) and the production of the annotated page (AN). As shown in the Figure, the hotspot identification is the most complex



	alatest.com	buy.com	epinions.com	overstock.com	ebay.com
AN	0,2	0,3	0,8	0,1	0,3
ER	1,1	1,3	2,5	1,2	1,4
ID	3,1	6,3	4,1	3,2	2,1
PP	0,5	0,1	2	0,2	0,3

Fig. 8 Average response time (sec) to annotate one page

task (i.e. computationally) due to the analysis of blocks occurring into a page. However such processing allows us to reduce the complexity of the following entity recognition (i.e. 1 sec in average).

6 Conclusions and Future Work

In this paper, we have explored named entity recognition- and heuristic-based approaches for annotating Web pages with Google Rich Snippets-compliant RDFa attributes. Heuristics used include source code analysis, pattern matching and internal word lists for discovery of entity attributes. We conclude that these approaches form a potential strategy to recognize entities on a Web page and automatically add them to the page using RDFa attributes. The tool that was developed performs reasonably in detecting review entities on review pages, such as *amazon.com*. At the same time, further work is needed to (1) improve the specificity of entity detection and (2) fully recognize the structure and properties of record-based entities such as persons and organizations. In particular, one of the difficulties we have experienced involves recognizing the rating of a review. Although this property is not compulsory in the Rich Snippets definition, it is one of the key aspects used to display a rich snippet in Google search results. We have found that many reviews lack an explicit rating, such as a grade or a number of stars. Rich Snippets accepts a rating based on a scale of 1–5. To provide a rating for every single review, we would have to calculate a rating based on the review body text itself. This approach is known as sentiment classification. Earlier studies on various kinds of reviews show that satisfying results can be obtained when adopting sentiment classification [19,20,21,22]. Analysis and adoption of this approach might be an interesting future research direction.

References

1. Bizer, C., Cyganiak, R.: D2R server: Publishing relational databases on the semantic web. In: Proc. of the 5th Intl Semantic Web Conf. (ISWC 2006). (2006)

2. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American **284** (2001) 34–43
3. Goel, K., Guha, R.V., Hansson, O.: Introducing Rich Snippets. http://googlewebmastercentral.blogspot.com/2009/05/introducing-rich-snippets.html (2009)
4. Google: Google Webmaster Tools: About review data. http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=146645 (2009)
5. B. Adida and M. Birbeck: RDFa Primer: Bridging the Human and Data Webs. http://www.w3.org/TR/xhtml-rdfa-primer/ (2008)
6. Virgilio, R.D., Torlone, R.: A Structured Approach to Data Reverse Engineering of Web Applications. In: 9th International Conference on Web Engineering, Springer-Verlag (2009) 91–105
7. Laender, A., Ribeiro-Neto, B., Silva, A.D., Teixeira, J.S.: A brief survey of web data extraction tools. ACM SIGMOD Record **31** (2002) 84–93
8. Mikheev, A., Moens, M., Grover, C.: Named Entity Recognition without gazetteers. In: Ninth Conference on European Chapter of the Association for Computational Linguistics, Association for Computational Linguistics (1999) 1–8
9. Morgan, R., Garigliano, R., Callaghan, P., Poria, S., Smith, M., Urbanowicz, A., Collingham, R., Costantino, M., Cooper, C., Group, L.: University of Durham: Description of the LOLITA System as Used in MUC-6. In: Sixth Message Understanding Conference, Morgan Kaufmann Publishers (1995)
10. Krupka, G.R., Hausman, K.: IsoQuest, Inc: Description of the NetOwl(TM) extractor system as used for MUC-7. In: Seventh Message Understanding Conference. (1998)
11. Seomoz.org: Search Engine Ranking Factors 2009. http://www.seomoz.org/article/search-ranking-factors (2009)
12. Can, L., Qian, Z., Xiaofeng, M., Wenyin, L.: Postal Address Detection from Web Documents. In: International Workshop on Challenges in Web Information Retrieval and Integration, IEEE Computer Society (2005) 40–45
13. Yahoo!: SearchMonkey: Site Owner Overview. http://developer.yahoo.com/searchmonkey/siteowner.html (2009)
14. Electrum: Valid HTML Statistics. http://try.powermapper.com/demo/statsvalid.aspx (2009)
15. Virgilio, R.D., Torlone, R.: A meta-model approach to the management of hypertexts in web information systems. In: ER Workshops (WISM 2008). (2008)
16. Allison, L., Wallace, C.S., Yee, C.N.: When is a string like a string? In: AI & Maths. (1990)
17. Tomberg, V., Laanpere, M.: RDFa versus Microformats: Exploring the Potential for Semantic Interoperability of Mash-up Personal Learning Environments. In: Second International Workshop on Mashup Personal Learning Environments, M. Jeusfeld c/o Redaktion Sun SITE, Informatik V, RWTH Aachen (2009) 102–109
18. Ratinov, L., Roth, D.: Design Challenges and Misconceptions in Named Entity Recognition. In: Thirteenth Conference on Computational Natural Language Learning, Association for Computational Linguistics (2009) 147–155
19. Turney, P.: Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews. 40th Annual Meeting of the Association for Computational Linguistics, ACL (2002) 417–424
20. Pang, B., Lee, L., Vaithyanathan, S.: Thumbs up? Sentiment Classification using Machine Learning Techniques. Conference on Emprirical Methods in Natural Language Processing, ACL (2002) 79–86
21. Ye, Q., Zhang, Z., Law, R.: Sentiment Classification of Online Reviews to Travel Destinations by Supervised Machine Learning Approaches. Expert Systems with Applications, 36(3) (2009) 6527–6535
22. Kennedy, A., Inkpen, D.: Sentiment Classification of Movie Reviews Using Contextual Valence Shifters. Computational Intelligence, 22(2) (2006) 110–225