IRIS: A Protégé Plug-in to Extract and Serialize Product Attribute Name-Value Pairs

Tuğba Özacar

Department of Computer Engineering, Celal Bayar University
Muradiye, 45140, Manisa, Turkey
tugba.ozacar@cbu.edu.tr

Abstract. This article introduces IRIS wrapper, which is developed as a Protégé plug-in, to solve an increasingly important problem: extracting information from the product descriptions provided by online sources and structuring this information so that is sharable among business entities, software agents and search engines. Extracted product information is presented in a GoodRelations-compliant ontology. IRIS also automatically marks up your products using RDFa or Microdata. Creating GoodRelations snippets in RDFa or Microdata using the product information extracted from Web is a business value, especially when you consider most of the popular search engines recommend the use of these standards to provide rich site data for their index.

Keywords: product, GoodRelations, Protégé, RDFa, Microdata

1 Introduction

The Web contains a huge number of online sources which provides excellent resources for product information including specifications and descriptions of products. If we present this product information in a structured way, it will significantly improve the effectiveness of many applications [1]. This paper introduces IRIS wrapper to solve an increasingly important problem: extracting information from the product descriptions provided by online sources and structuring this information so that is sharable among business entities, software agents and search engines. The information extraction systems can be divided into three categories [2]: (a) *Procedural Wrapper:* The approach is based on writing customized wrappers for accessing required data from a given set of information sources. The extraction rules are coded into the program. Creating wrappers are easier and it can directly output the domain data model of application but each wrapper works only for an individual page. (b) *Declarative Wrapper:* These systems consist of a general execution engine and declarative extraction rules developed for specific data sources. The wrapper takes an input specification that declaratively states where the data of interest is located on the HTML document, and how the data should be wrapped into a new data model. (c) *Automatic Wrapper:* The automatic extraction approach uses machine learning techniques to learn extraction rules by examples. In [3] information extraction systems are classified

into two: solutions treating Web pages as a *tree*, and solutions treating Web pages as *data stream*. Systems are also divided with respect to the level of automation of wrapper creation into *manual*, *semi-automatic* and *automatic*. IRIS is a declarative and manual tree wrapper [1], which has a general rule engine that executes the rules specified in a template file using XML Path Language (XPath). Manual approaches are known to be tedious, time-consuming and require some level of expertise concerning the wrapper language [4]. However, manual and semi-automatic approaches are currently better suited for creating robust wrappers than the automatic approach. Writing an IRIS template is considerably easier than most of the existing manual wrappers. Besides, it can be predicted that to improve the reusability and the efficiency, the users of the IRIS engine will share templates on the Web.

There are works which directly focus on the problem of this paper. [5] uses a template-independent approach to extract product attribute name and value pair from Web. This approach makes hypothesis to identify the specification block but since some detail product pages may violate these hypothesis, the pairs in these pages cannot be extracted properly. The second work [6] needs two predefined ontologies to extract product attribute name and value pairs from a Web page. One of these ontologies is built according to the contents of the page but it is not an easy task to build that ontology from scratch for every change in the page content. The system presented in this paper differs from the above works in many ways.

First of all the system transforms the extracted information into an ontology to share and reuse common understanding of structure of information among users or software agents. To my knowledge [7], IRIS is the first Protégé plug-in that is used to extract product information from Web pages. Designed as a plug-in for the open source ontology editor Protégé, IRIS exploits the advantages of the ontology as a formal model for the domain knowledge and profits from the benefits of a large user community (currently *230,914* registered users).

Another feature is support for building an ontology that is compatible with GoodRelations Vocabulary [8], which is the most powerful vocabulary for publishing all of the details of your products and services in a way friendly to search engines, mobile applications, and browser extensions. The goal is to have extremely deep information on millions of products, providing a resource that can be plugged into any e-commerce system without limitation. If you have GoodRelations in your markup, Google, Bing, Yahoo, and Yandex will or plan to improve the rendering of your page directly in the search results. Besides, you provide information to the search engines so that they can rank up your page for queries to which your offer is a particularly relevant match. Finally, as an open source Java Application, IRIS can be further extended, fixed or modified according to the needs of the individual users.

The following section (with three subsections) includes the system's features and a scenario based quick-start guide. Section 3 concludes the paper with a brief talk about possible future work.

[1] Download link: https://github.com/tugbaozacar/iris

2 Scenario-based System Specification

IRIS system gathers semi-structured product information from an HTML page, applies extraction rules specified in the template file, and presents the extracted product data in an ontology that is compatible with GoodRelations Vocabulary. The HTML page is first parsed into a DOM tree using HtmlUnit, which is a Web Driver that supports walking the DOM model of the HTML document using XPath queries. In order to get product information from Web page, the template file includes a tree that specifies the paths of HTML tags around the product attribute names and product attribute values. Figure 1 shows the architecture of the system briefly. User builds a template for the pages containing the product information.

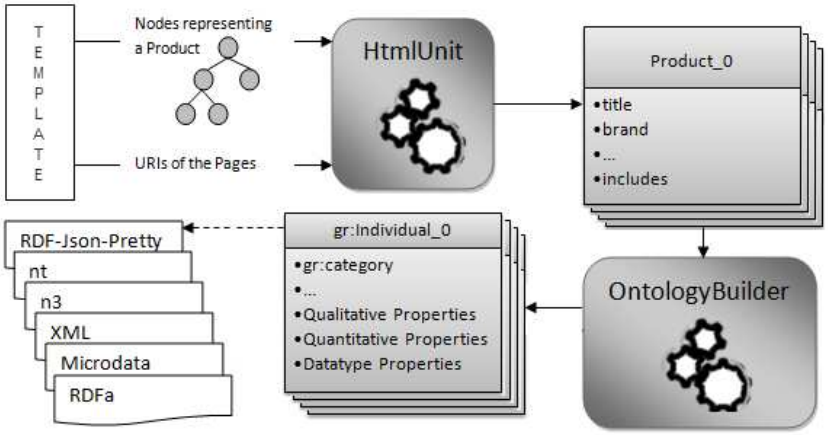

Fig. 1. Architecture of the system.

Then HtmlUnit library parses the Web pages. The system evaluates the nodes in the template and queries the HtmlUnit for the required product properties. At the end of this process, the system returns a list of product objects. To define a GoodRelations-compliant ontology the user maps the product properties to the properties of the "gr:Individual" class, saves the ontology and serializes the ontology into a series of structured data markup standards. The system makes serialization via RDF Translator API [9]). Each step is described in the following subsections.

2.1 Create a Template File

The information collected is mapped to the attributes of the *Product* object including title, description, brand, id, image, features, property names, property values and components. A template has two parts; the first part contains the tree that specifies the paths of HTML tags around the product attribute names and values. The second part specifies how

the HTML documents should be acquired. The product information is extracted using the tree. The tree is created manually and its nodes are converted to XPath expressions. HtmlUnit evaluates the specified XPath expressions and returns the matching elements.Figure 2 shows the example HTML code which contains the information about the first product in "amazon.com" pages that contain information about laptops. Figure 3 shows the tree which is built for extracting product information from the page in Figure 2.

```
<div id="result_0" class="fstRowGrid prod celwidget" name="B00D3F7H0K">
    <div class="linePlaceholder"></div>
    <div class="image imageContainer">
      <a href="http://www.amazon.com/Toshiba-Satellite-C55D-A5240 ... >
      <img  src="http://ecx.images-
      amazon.com/images/I/4108rtR%2BQWL. AA160_.jpg" class="productImage"
      alt="Product Details"/>                                    product.imgLink
      </a>
      <div class="smallVariationsBox"> </div>
    </div>
    <h3 class="newaps">
      <a href="http://www.amazon.com/Toshiba-Satellite-C55D-A524 ... >
       ...
      <span class="lrg bold">Toshiba Satellite C55D-A5240NR 15.6-Inch
      Laptop (Satin Black in Trax Horizon)
      </span>                                                    product.title
      </a>
    </h3>
    ...
</div>
```

Fig. 2. The HTML code which contains the first product on the page.

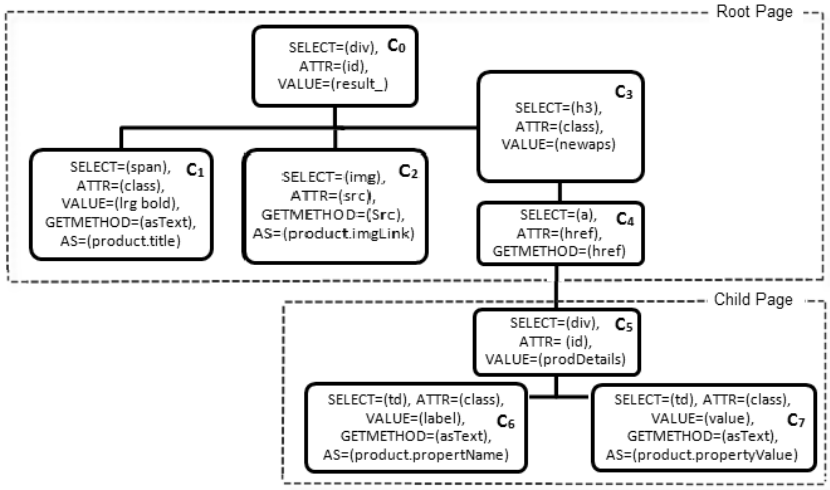

Fig. 3. The tree nodes in the template for "amazon.com" pages about laptops.

The leaf nodes of the tree (Figure 3) contains the HTML tag around a product attribute name or a product attribute value, and the internal nodes of the tree contains the HTML tags in which the HTML tag in the leaf node is nested. Therefore the hierarchy of the tree also represents the hierarchy of the HTML tags. c_1 contains the value of the title attribute, c_2 contains the image link of the product, and c_3 is one of the internal nodes that specify the path to its leaf nodes. c_3 specifies that all of its children contain HTML tags which are nested within the *h3* heading tag having class name "newaps". Its child node (c_4) specifies the HTML link element which goes to another Web page that contains detailed information about the product. The starting Web page is referred as root page and the pages navigated from root page are child pages. After jumping the page address specified by c_4, product properties and their values are chosen from this Web page which is shown in Figure 4.

```
<div id="prodDetails">
…
<tr><td class="label">Screen Size</td>
<td class="value">15.6 inches</td></tr>
<tr><td class="label">Screen Resolution</td>
<td class="value">1366 x 768</td></tr>
<tr><td class="label">Max Screen Resolution</td>
<td class="value">1366x768 pixels</td></tr>
…
</div>
```

Fig. 4. Product properties and their values.

The properties and their corresponding values are stored in an HTML table, which is nested in an HTML division identified by "prodDetails" *id*. Therefore c_5 specifies this HTML division and its child nodes c_6 and c_7 specifies the HTML cells containing product properties and their values. After determining the HTML elements which contain the product information, the user defines these elements in the template properly. Each node in the tree is a combination of the following fields:

SELECT-ATTR-VALUE These three fields are used to build the XPath query that specifies the HTML element in the page.

ORDER is used when there is more than one HTML element matching with the expression. The numeric value of the ORDER element specifies which element will be selected.

GETMETHOD is used to collect the proper values in the selected HTML element e. If you want to get the textual representation of the element (e), in other words what would be visible if this page was shown in a Web browser, you define the value of GETMETHOD field as "asText". Otherwise you get the value of an element (e) attribute by specifying the name of the attribute as the value of GETMETHOD field.

AS is only used with leaf nodes. The value collected from a leaf node using GETMETHOD field is mapped to the *Product* attribute specified in the AS field.

Appendix A gives the template (amazon.txt) which contains the code of the tree in Figure 3. The second part of a template file contains the information on how the HTML documents should be acquired. This part has the following fields:
NEXT_PAGE The information about laptops in "amazon.com" is spread across 400 pages. The link of the next page is stored in this field.
PAGE_RANGE specifies the number of the page or the range of pages which you want to collect information from. In my example, I want to collect the products in pages from 1 to 3.
BASE_URI represents the base URI of the site. In my example, the value of this field is http://www.amazon.com.
PAGE_URI is the URI of the first page which you want to collect information from. In my example, this is the URI of the page 1.
CLASS contains the name of the class that represents the products to be collected. In my example, "Laptop" class is used.

2.2 Create an Ontology that is Compatible with GoodRelations Vocabulary

First of all, user opens an empty ontology ("myOwl.owl") in the Protégé Ontology Editor and displays the IRIS tab which is listed on the TabWidgets panel. Then the user selects the template file using "Open template" button in Figure 5 (for this example: amazon.txt). Then the tool imports all laptops from the "amazon.com" pages specified in the PAGE_RANGE field. The imported individuals are listed in the "Individuals Window" (Figure 5). The "Properties Window" lists all properties of the individuals in "Individuals Window".
In this section, I follow up the descriptions and examples introduced in GoodRelations Primer [10]. First of all, the system defines the class in your template ("Laptop" class in example) as a subclass of "gr:Individual" class of the GoodRelations vocabulary. Then the properties of the "Laptop" class, which are collected from the Web page should be mapped to the properties of "gr:Individual", which can be classified as follows:
First category: "gr:category", "gr:color", "gr:condition", etc. (see [10] for full list). If the property p_x is semantically equivalent of a property from the first category p_y , then user simply maps p_x to p_y.
Second category: Properties that specify quantitative characteristics, for which an interval is at least theoretically an appropriate value should be defined as subproperties of "gr:quantitativeProductOrServiceProperty". A quantitative value is to be interpreted in combination with the respective unit of measurement and mostly quantitative values are intervals.
Third category: All properties for which value instances are specified are subproperties of "gr:qualitativeProductOrServiceProperty".
Fourth category: Only such properties that are no quantitative properties and that have no predefined value instances are defined as subproperties of "gr:datatypeProductOrServiceProperty".
To create a GoodRelations-compliant ontology, user selects the individuals and properties that will reside in the ontology. Then she clicks the "Use GoodRelations Vocabulary" button (Figure 5) and "Use GoodRelations Vocabulary" wizard appears. She selects the corresponding GoodRelations property type and respective unit of measurement.

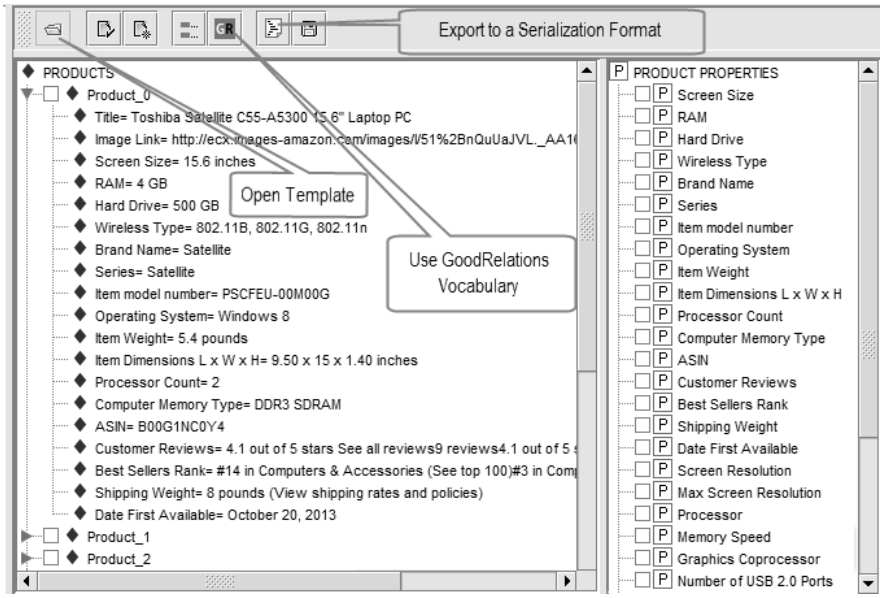

Fig. 5. The tool imports all laptops from the specified "amazon.com" pages.

2.3 Save and Serialize the Ontology

User saves the ontology in an owl file and clicks the "Export to a serialization format" button (Figure 5) to view the ontology in one of the structured data markup standards.

3 Conclusion and Future Work

This work introduces a Protégé plug-in called IRIS that collects product information from Web and transforms this information into GoodRelations snippets in RDFa or Microformats. The system attempts to solve an increasingly important problem: extracting useful information from the product descriptions provided by the sellers and structuring this information into a common and sharable format among business entities, software agents and search engines. I plan to improve the IRIS plug-in with an extension that gets user queries and sends them to Semantics3 API [11], which is a direct replacement for Google's Shopping API and gives developers comprehensive access to data across millions of products and prices. Another potential future work is generating an environment for semi-automatic template construction. An environment that automatically constructs the tree nodes from the selected HTML parts will significantly reduce the time to build a template file. And yet another future work is diversify the supported input formats (pdf, excel, csv etc.).

Appendix A

```
SELECT=(div), ATTR=(id), VALUE= (result_) [
  SELECT=(span), ATTR=(class), VALUE=(lrg bold),
  GETMETHOD=(asText, AS=(product.title);
  SELECT=(img), ATTR=(src), GETMETHOD=(Src),
  AS=(product.imgLink);
  SELECT=(h3), ATTR=(class), VALUE= (newaps) [
    SELECT=(a), ATTR=(href), GETMETHOD=(href) [
      SELECT=(div), ATTR=(id), VALUE=(prodDetails)[
        SELECT=(td), ATTR=(class), VALUE=(label),
        GETMETHOD=(asText, AS=(product.propertyName);
        SELECT=(td), ATTR=(class), VALUE=(value),
        GETMETHOD=(asText, AS=(product.propertyValue)]]]]
NEXT_PAGE:{SELECT=(a), ATTR=(id), VALUE=(pagnNextLink),
           GETMETHOD=(href)}
PAGE_RANGE:{1-3}
BASE_URI:{http://www.amazon.com}
PAGE_URI:{http://www.amazon.com/s/ref=sr\_nr\_n\_1?rh=
          n\%3A565108\%2Ck\%3Alaptop\&keywords=laptop\&
          ie=UTF8\&qid=1374832151\&rnid=2941120011}
CLASS:{Laptop}
```

References

1. Tang, W., Hong, Y., Feng, Y.H., Yao, J.M., Zhu, Q.M.: Simultaneous product attribute name and value extraction with adaptively learnt templates. In: Proceedings of CSSS '12. (2012) 2021–2025
2. Han, J.: Design of Web Semantic Integration System. PhD thesis, Tennessee State University. (2008)
3. Firat, A.: Information Integration Using Contextual Knowledge and Ontology Merging. PhD thesis, MIT, Sloan School of Management (2003)
4. Muslea, I., Minton, S., Knoblock, C.: A hierarchical approach to wrapper induction, ACM Press (1999) 190–197
5. Wu, B., Cheng, X., Wang, Y., Guo, Y., Song, L.: Simultaneous product attribute name and value extraction from web pages. In: Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference, IEEE Computer Society (2009) 295–298
6. Holzinger, W., Kruepl, B., Herzog, M.: Using ontologies for extracting product features from web pages. In: Proceedings of the ISWC'06, Springer-Verlag 2006 (2006) 286–299
7. : Protege plug-in library Last accessed: 2013-09-24.
8. Hepp, M.: Goodrelations: An ontology for describing products and services offers on the web. EKAW '08 (2008) 329–346
9. Stolz, A., Castro, B., Hepp, M.: Rdf translator: A restful multiformat data converter for the semantic web. Technical report, E-Business and Web Science Research Group (2013)
10. Hepp, M.: Goodrelations: An ontology for describing web offers —primer and user's guide. Technical report, E-Business + Web Science Research Group 2008.
11. Semantics3 Inc.: Semantics3 - apis for products and prices (2013)