Data Extraction from Semantic Annotated Deep Web Sites

Eduardo Martín Rojo and Vicente Luque Centeno

Universidad Carlos III de Madrid
Av. Universidad 30, 28911
Leganés (Madrid), España
{emartin,vlc}@it.uc3m.es

Abstract. Automatic navigating and gathering information from Deep Web sites requires the use of Web Wrappers in order to simulate human interaction with Web sites. Web Wrappers have some drawbacks: their implementations are specific to the accessed site and also their source code needs a constant maintenance in order to support new changes on Web site.
In this work we propose an annotation model for Deep Web sites that could be used for data extraction from the point of view of a Web client. Using these annotations will enable Web Wrappers to be more adaptable to Web site changes.

1 Introduction

Nowadays most popular Web sites provide to their users with development tools that make possible to create applications that access their services, like *eBay Developers Program*[1]. Also, sometimes they provide Web services that could be accessed by *Mashup* applications like *Google Mashups*[2], *Yahoo Pipes*[3] or *Microsoft Popfly*[4]. However, the great majority of Web sites do not provide development tools or Web services because they are only focused on being operated by human users. Automatic accessing Web sites that do not provide these facilities is achieved by using *Web Wrappers*[7].

Web Wrappers have some drawbacks: first, they require developers with strong knowledge on the accessed Web site because Web Wrappers are site specific solutions that have dependencies with Web structure; and second, Web Wrappers require constant maintenance in order to support new changes on the Web sites they are accessing. Web Wrappers development tools evolved trying to solve these drawbacks and also to make possible an easier integration of Web data from heterogeneous sources. Some examples of common Web Wrappers development tools are site specific solutions like *GreaseMonkey*[5] and *Ubiquity*[6],

[1] http://developer.ebay.com/
[2] http://www.googlemashups.com/
[3] http://pipes.yahoo.com
[4] http://www.popfly.com
[5] http://www.greasespot.net/
[6] http://ubiquity.mozilla.com/

user-friendly tools that allow natural language references like *Chickenfoot*[7], and also graphical IDE solutions like *OpenKapow RoboMaker*[8].

The main common characteristic among all current Wrapper development tools is that, although their operation is simple, they still have strong dependencies with Web site structure, originating a continuous effort of maintenance with the created Wrappers. In our work we will define a formalization of Web structure that can be used for semantic annotation of Deep Web sites in order to represent their navigation operation. These annotations combined with current Wrapper development tools will allow implementation of Wrappers focused on the Web site model, avoiding overlapping with site structure. The maintenance required by new Web site changes will be isolated inside this model layer. Any future modifications on the structure of the Web site will need only to change the semantic annotation for the Web site, and all the Wrappers that make use of these annotations will not need to be changed.

One of the main problems raised with the navigation model generation is the election of a point of view. In [11], it is indicated that a Web navigation model can be generated from three points of view: from server point of view, analyzing physical structure of Web site; from client point of view, analyzing client interaction with Web site; and from a hybrid point of view as a combination of client and server point of view. In our work we have chosen to follow the client point of view because of the following reasons:

- The navigation model must support the changeableness and diversity of Web content. Client side technologies like AJAX or Flash are widespread. The model must support also this kind of content.
- It is non intrusive; it does not require to change the Web site being annotated as other semantic annotation methods like RDFa[12] and Microformats[10].
- Client-only point of view allow third party and end users to participate and collaborate in the creation of annotations about any Web site because they will not have access to the server.

In this document, section 2 will introduce previous works related with Deep Web navigation modeling and data extraction. The annotation model for expressing navigation graphs that we present is described in section 3 by defining its main classes and properties. Also we present an example of annotation for a shop Web site that uses our model. Section 4, describes how can be used the annotations for the extraction of information by using an example that performs a query over different sources; and finally, future works and conclussions are described in section 5.

2 Related Works

The task of generating a navigation model from a Web Site has been previously faced, but previous works do not provide a Web model annotations formalization.

[7] http://groups.csail.mit.edu/uid/chickenfoot/
[8] http://openkapow.com/

There were attempts to automatically extract the model using real navigation examples like [1], where the authors make use of navigation examples that were extracted by recording and replaying the actions performed by a user and [11], focused on generation of models for visualization of Web sites hierarchy.

Deep Web navigation model generation is treated by [13], where the authors generate a navigation model where they use keyword-matching for identifying different Deep Web pages. Other work more focused on interacting with Deep Web sites is *Transcendence*[2], a system that allows users to generalize their queries for expanding their search scope. It allows also to define data extraction patterns for obtaining output data that may be combined with other data sources.

One of the main problems of Deep Web remains in that a Deep Web page cannot be always represented uniquely with an URL[9]. The problem of referencing Deep Web pages is addressed in [8], where the authors present a way of creating bookmarks of a Deep Web page using the sequence of steps specified with *Chickenfoot* scripts that simulate the user interaction in order to reach the bookmarked page. This is combined with images that show the visual representation of the bookmarked page.

Extracting semantic data from Web sites is a problem that have been faced in *Marmite*[16], a system that provides an end-user interface that allows him to construct the workflow needed for extracting the data, or *PiggyBank*[9], a system where the user can make use of scripts called *Screen Scrapers* for converting HTML to semantic data in RDF. Related with this last work is Sifter, a tool that

In our demo system we use *Chickenfoot* as Wrapper development tool. The main characteristics of this tool and its natural language references to HTML elements are presented on [3] and [4]. *Chickenfoot* enable the specification of client-side Web interactions with a simple Javascript based language.

3 Deep Web Annotation Model

The objective of a Web client is to reach a specific state through interacting with the Web. Our model represent the states and transitions that composes the navigation as a graph that describes all possible situations that could occur in a Web site from client's point of view. In the graph, vertexes represents all possible logical states that require an action produced by the user, while edges represent the actions produced that allow the transitions between logical states.

Every state could be divided in elements called fragments, and also these fragments could be divided into new fragments. Every fragment represents a type of semantic content that can be extracted by selecting part of the element that it belongs. Figure 1 shows an example of state division in different fragments. Fragments of logical states will allow us to extract semantic information from Deep Web pages if we know the location of the state inside the navigation graph of the Web site.

[9] Uniform Resource Locator, defined on http://tools.ietf.org/html/rfc1738

Fig. 1. States and Fragments

A transition represents the actions available for the user. These actions will allow changing among different states in the Web site. Transitions are composed of, first, an ordered sequence of interactions that originates the change of state, and second, a list of all possibe destinations that could be feasible by using the transition. There could be different possible destinations because the Web site could act in different ways as a consequence of dynamic factors like the state of the service framework, the interactions historical, date and time, etc.

Our model of states and transitions for the navigation graph is represented by the following types of elements:

1. PageState: a uniquely identifying state that represents a Deep Web page. A state contains fragments.
2. Fragment: represents an element inside a PageState or another Fragment. It is domain of the following properties:
   (a) fragmentOf: Defines this fragment as part of another fragment or part of a PageState as a hierarchy.
   (b) locatedBy: a XPath expression that identifies the position of the fragment inside the XHTML representation of the PageState.
   (c) numResults: an integer that defines how many times this fragment appears inside his father Fragment or PageState in hierarchy.

   (d) optional: a boolean that indicates if the fragment appears optionally inside his father Fragment or PageState. It may happen when a page has variable elements.
   (e) semantic: a set of RDF triples that represents the knowledge refered by this fragment. This property is also defined in Input and Action.
3. transitions: represents all possible transitions to other states that could be followed from this state. It has the following properties:
   (a) actions: any transition is originated by an ordered sequence of actions (instances of Action class)
   (b) sources and destinations: all possible states that could be source or destination of this transition respectively.
   (c) precondition and postcondition: preconditions and postconditions required in order to use this transition for travelling from sources to destinations. Postconditions can be used for distinguishing between different possible destinations.
4. Action: an interaction that could be performed over a fragment. It contains the following properties:
   (a) hasType: type of interaction (clicking element, selecting element, entering text...). It is represented by a *Chickenfoot* command in our demo annotations.
   (b) inputs: all the form values needed for performing the interaction are refered in this property as a list that contains elements of type Input. For each input it must be provided a name and a description.

We have represented this formalization in an OWL[10] ontology that can be accessed in [15].

In figure 2 we have represented an example of the annotated navigation model for searching products inside a typical Shop of Books site. From initial page at this site represented by node state0, there are two fragments: element0 (a searching textbox) and element1 (a button that initiates the searching task). From state0 it is possible to go to state1 through transition0. This transition requires to perform two actions (represented by the orderedActions list of actions): the first action is performed by inserting a search string inside element0 textbox and the second action is performed by clicking element1 button. At state1, there are three fragments that can be accessed: element2, element3 and element4. Because each of them is a part of other fragment, the XPath expression referred by the locatedBy property is relative to its father's locatedBy property. For example, element4 is constructed with element3 and element2 locations, and so its location should be //div[@id='atfResults'][$INDEX]//div[@class='productPrice'].

In this figure we have used an hypothetical ontology for defining concepts of Shops, but the annotation model presented can be combined with any ontology for defining semantic knowledge inside the Fragments, Actions or Inputs of the model. This knowledge can be provided as RDF data inside the property semantic of these elements.

[10] http://www.w3.org/TR/owl-features/

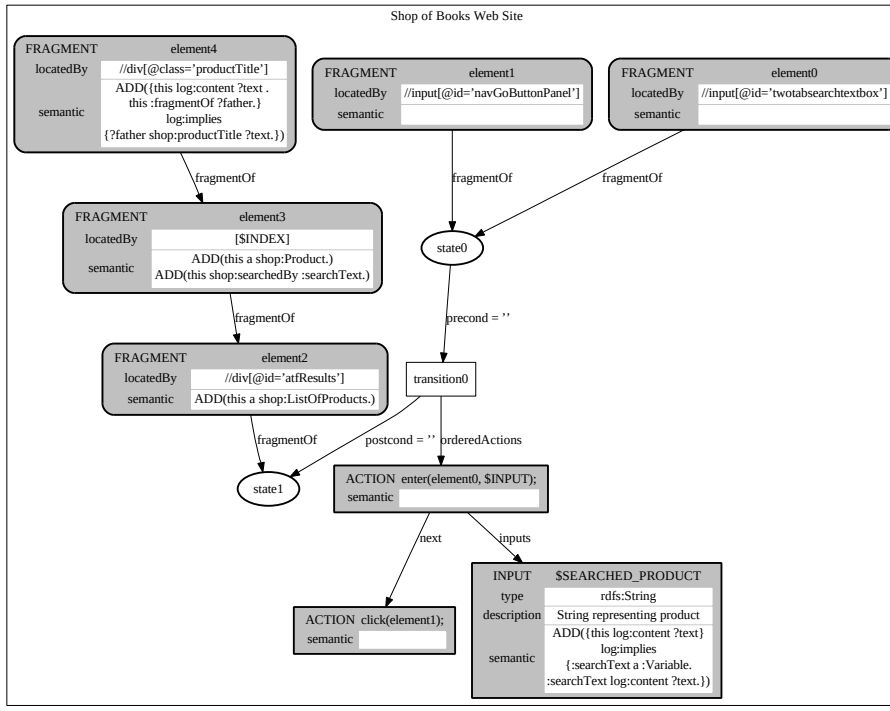

Fig. 2. Annotated navigation model for searching at Shop of Books site

Representing the semantic of a specific element inside a state for a Deep Web site requires not only the knowledge inferred from the specific state, but also the knowledge of all the transitions that have been followed for reaching the state. This knowledge may be transformed during the transitions, and new knowledge may be added or removed from the client's working memory of assertions, depending on the client interactions with the Deep Web site, as in a Production System or Rule-Based System [5].

As can be seen in figure 2, Fragment, Input and Action have defined the semantic property that indicates which knowledge (represented as RDF triples or inference rules) is added (ADD) or deleted (DEL) from the working memory. Adding a RDF triple simply adds itself to the working memory, while adding a RDF rule executes it inside working memory and every time new RDF triples are added, the rule must be reapplied. Deleting a triple or a rule eliminates its effect from the working memory. The effect of adding or removing RDF data of client's working memory follows these rules:

1. If the client is located at a State that contains fragments, the semantic property of the fragments is processed inside working memory from outer fragments to inner fragments following the fragmentOf property. Fragments at the same level of indexation can be processed in any order.
2. If the client has travelled through a Transition that contains a ordered list of actions, the semantic property of the actions is processed following the same order of the ordered actions list.
3. If the client uses an input defined in a fragment of a State or in an action of a Transition, the semantic property of the input is processed after all the other semantic properties of the State or Transition.

Semantic information is associated with every instance of Fragment, Action or Input by using the semantic property. With this property, the user that is annotating a Web site indicates which kind of information does the fragment, action or input represents. Its content is expressed in RDF.

For defining semantic information, in figure 2 we make use of the following properties and concepts (based on the built-in CWM reasoner functions[11]):

- log:implies → Property that relates antecedents and consequents of a RDF expressed rule. Antecedents and consequents are defined between brackets { and }.
- log:content → This property relates the logical representation of an element with its string representation. It is used for defining the text content of an input, or for defining the specific values of an information.
- this → When it is used inside the semantic property of an element, it represents the element itself. It is used for adding knowledge to an element.

In order to facilitate sharing and reusing annotations, it is needed the use of ontologies for modeling this semantic content.

[11] http://www.w3.org/2000/10/swap/doc/CwmBuiltins.html

4 Data Extraction using Annotations

The content of the semantic property allows to locate a particular information inside the navigation map by performing a query that specifies the conditions in which the information can be found in the working memory. As our model deals with knowledge expressed as RDF triples, we have selected SPARQL[12] as query language for this purpose. SPARQL allows to indicate the Named Graph[6] that a set of conditions are referring. We make use of this functionality for relating annotations of different Deep Web sites in order to perform a distributed query among their graphs. Figure 3 shows an example of query that uses two different maps. The SPARQL query requires the price of a book that fulfills the following conditions expressed in the WHERE part of the query:

1. Select from RSS Web Site any element identified as Product that has a defined title
2. Select from Shop of Books Web Site any element identified as Product that has been searched in the site using the string that represents the title of the product from RSS Web Site previously obtained

A set of conditions expressed for a specific navigation model in the query can be achieved by a list of transitions. The actions of these transitions could need to provide client inputs. Also these inputs could be needed for accessing specific fragments in a state. In the example at figure 3, RSS map requires an input called INDEX for accessing a fragment inside rss_state0, and Shop of Books map requires the input SEARCHED_PRODUCT for performing the actions of transition0. Inputs are the points where the SPARQL query can relate data among different maps. Because inputs require that an information must be supplied, the SPARQL query must face with the possible situations with these rules:

- If the conditions states clearly the specific value of the input, use this value.
- If the conditions indicate that the information required by the input must be obtained from another graph, then the query must first perform its actions in that other graph. This happens in the example with Shop of Books map at condition ?product2 shop:searchedBy ?title because title is refered by the RSS map.
- If the information required can not be infered, then the SPARQL query must use all possible informations that could be used for the input. In the example, this happens with the input INDEX, that is not provided, so the query must iterate among all the elements.

We have developed a demo system that may be accessed through [14]. Our demo generates Web wrappers scripts composed of *Chickenfoot*[13] commands that could be executed in *Firefox* with *Chickenfoot* plugin installed. *Chickenfoot* is a programming environment that provides a set of special functions for

[12] http://www.w3.org/TR/rdf-sparql-query/
[13] http://groups.csail.mit.edu/uid/chickenfoot/

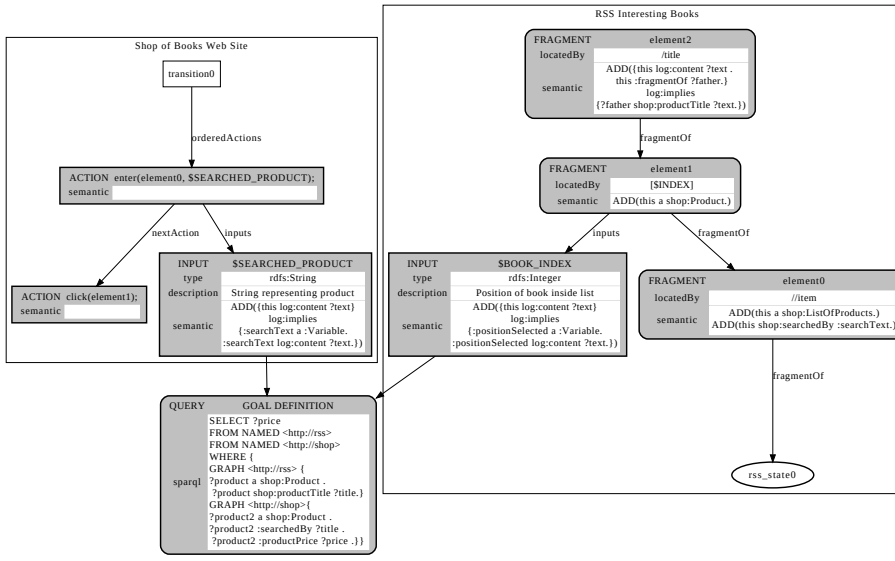

Fig. 3. Example querying over different maps

performing Web tasks like clicking a link, entering data in a HTML form, etc. *Chickenfoot* scripts are written in a superset of Javascript. One of its main characteristic is its support of natural language naming for HTML elements which eases development of Web automatic tasks to end users [3].

The scripts are generated from SPARQL queries that can use annotations expressed in the formalization that we have presented in this article, also accesible as OWL at URL [15]. Although with SPARQL we can not define very complex tasks, we think it is a good starting point for defining a more powerful language that may be used in a semantic-oriented Web Wrappers development. We think this new approach may be useful for solving the drawbacks of previous Web Wrappers development on Deep Web sites, like dependencies with Web structure, constant maintenance of wrappers scripts, the need of strong knowledge of the physical structure of accessed Web sites, etc.

5 Future works

Accessing and integrating data from non-structured heterogeneous sources is a problem that currently is solved with Web Wrappers despite its drawbacks. Web Wrappers are also tools that may be used to give structure to that non-structured information and made it available to the Web of Data. We think that wrapper development must be adapted to the new environment of linked data using semantic-oriented wrapper definitions, and not site-specific implementations. In order to achieve this, we are interested in researching more powerful

languages and development tools for this new semantic approach of semantic wrapper development.

We think our model will improve development of Wrappers, because all the Web site structure changes will require only to modify the annotation model, and all Wrappers that make use of the annotations will be corrected. However, it would be adequate an exhaustive evaluation of user effort with our annotation model, that will be accomplished in future work.

We are interested in continuing working on ontologies integration with semantic wrapper implementations in order to exploit the common concepts among different Web sites. The same semantic wrapper implementation may be used in any Web site that is annotated using the same ontology.

6 Acknowledgements

This work has been partially funded by the spanish Ministry of Education and Science, project ITACA No. TSI2007-65393-C02-01

References

1. Robert Baumgartner, Michal Ceresna, and Gerald Ledermuller. Deep web navigation in web data extraction. In *CIMCA '05: Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-2 (CIMCA-IAWTIC'06)*, volume 2, pages 698–703, Washington, DC, USA, 2005. IEEE Computer Society.
2. Jeffrey P. Bigham, Anna C. Cavender, Ryan S. Kaminsky, Craig M. Prince, and Tyler S. Robison. Transcendence: enabling a personal view of the deep web. In *IUI '08: Proceedings of the 13th international conference on Intelligent user interfaces*, pages 169–178, New York, NY, USA, 2008. ACM.
3. Michael Bolin and Robert C. Miller. Naming page elements in end-user web automation. *SIGSOFT Softw. Eng. Notes*, 30(4):1–5, 2005.
4. Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. Automation and customization of rendered web pages. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 163–172, New York, NY, USA, 2005. ACM.
5. Ronald Brachman and Hector Levesque. *Knowledge Representation and Reasoning (The Morgan Kaufmann Series in Artificial Intelligence)*. Morgan Kaufmann, May 2004.
6. Jeremy J. Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named graphs, provenance and trust. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 613–622, New York, NY, USA, 2005. ACM.
7. Sudarshan Chawathe, Hector Garcia-molina, Joachim Hammer, Kelly Irel, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. The tsimmis project: Integration of heterogeneous information sources. In *In Proceedings of IPSJ Conference*, pages 7–18, 1994.

8. Darris Hupp and Robert C. Miller. Smart bookmarks: automatic retroactive macro recording on the web. In *UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 81–90, New York, NY, USA, 2007. ACM.
9. David Huynh, Stefano Mazzocchi, and David Karger. *Piggy Bank: Experience the Semantic Web inside your web browser*, volume 5, pages 16–27. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, 2007.
10. Rohit Khare and Tantek Celik. Microformats: a pragmatic path to the semantic web. pages 865–866, 2006.
11. Dirk Kukulenz. Adaptive site map visualization based on landmarks. In *IV '05: Proceedings of the Ninth International Conference on Information Visualisation*, pages 473–479, Washington, DC, USA, 2005. IEEE Computer Society.
12. W3C. Rdfa primer. W3C Working Group Note 14 October 2008, October 2008.
13. Yang Wang and Thomas Hornung. Deep web navigation by example. In Tomasz Kaczmarek Marek Kowalkiewicz Tadhg Nagle Jonny Parkes Dominik Flejter, Slawomir Grzonkowski, editor, *BIS 2008 Workshop Proceedings, Inssbruck, Austria, 6-7 May 2008*, pages 131–140. Department of Information Systems, Pozna, University of Economics, 2008.
14. WebTLab. Site annotation demo. http://corelli.gast.it.uc3m.es/siteannotation.
15. WebTLab. Site annotation ontology. http://corelli.gast.it.uc3m.es/siteannotation/ontology.owl.
16. Jeffrey Wong and Jason I. Hong. Making mashups with marmite: towards end-user programming for the web. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1435–1444, New York, NY, USA, 2007. ACM.