Information extraction from the World Wide Web [⋆]

Hassan A. Sleiman

Universidad de Sevilla
ETSI Informática
Avda. Reina Mercedes, s/n. Sevilla 41012 Spain
hassansleiman@us.es

Abstract. The World Wide Web is an enormous and a growing source of information presented in a human friendly language called Html. Unfortunately, querying and accessing this information by software agents is not an easy task, so web information extractors are used. Currently, there is a variety of algorithms to build web information extractors, but none of them is universally applicable. There is not a common software framework to develop them. This has resulted in proposals that range in complexity, precision and recall, but having diverging interfaces, which makes it difficult to reuse or integrate them. As a result, few side-by-side comparisons are available, but none of them is complete. We argue that the key is the absence of a unifying framework in which researchers can develop their proposals so that they can be assessed properly. Devising and implementing such a framework would be an ultimate tool to help reduce costs at integrating web information into automatic business processes. In this paper we report on our first version of this framework for information extractors.

Key words: Information extraction, Enterprise Information Integration.

1 Introduction

The World Wide Web is a growing database in which a great amount of information is present. Unfortunately, querying and accessing these data by software agents is not a simple task since data present in web pages is provided using human friendly formats. This type of codification makes it easier for humans to understand and browse the Web, but makes the incorporation and the use of this data by automated processes very difficult. Some solutions for this problem can be the use of the semantic web [22], which is still a vision, or the use of web services [3], but till these proposals are taken into consideration by all the web sites and are completely specified, one solution is to use information extractors to fill the gap and help us to transform data present in the web to completely structured data usable by automated processes.

There exists a great number of proposals for information extraction algorithms, but unfortunately none of them can be considered as a perfect solution. Information extractors such as [15], [1], [6], [4], [10], [12] and [24], are usually designed and built providing distinct interfaces, thus complicating the task of integration of these algorithms inside enterprise applications.

[⋆] Partially funded by the Spanish National R&D&I Plan under grants TIN2008-04718, and the Andalusian Local Government under grants P07-TIC-02602, and P08-TIC-4100.

Suppose we wish to integrate web information into a business process of ours, then one of these tools should be integrated into our enterprise application, but before integrating it, we would like to evaluate existing algorithms to choose the one that fits our case better. One of these three tasks should be performed: study existing surveys that compare information extractors, ask authors for these tools and then work on integrating them into our system, or even develop these algorithms to test them over the web sites we are going to work over. At the end of any of the previous tasks, an information extractor is chosen and integrated, but all of these tasks are tedious and expensive.

The variety of information extractors motivated the existence of many surveys to compare these extraction algorithms and tools such as: [2], [11], [20] and [8]. However, these surveys use taxonomies do not provide new information about the algorithms, and although effectiveness and efficiency are studied, they is not a side-by-side comparison. Besides, the results presented in algorithms' proposals are performed over distinct web sites, so these results are not comparable between each other.

Since no decision can be taken based on the comparison presented by surveys, using information extraction tools for comparison should be considered. We need the tools and the implementations of all the extraction algorithms to test and compare their efficiency and effectiveness. Existing tools are developed using different technologies and different libraries, thus the results of our tests are not reliable since technology conditions are distinct. Also, these tools provide different interfaces and integrating the chosen one into our enterprise application could be time and resource costly. Implementing all extraction algorithms is a very high costing task, user should study deeply a great number of algorithms, and develop them since there is not any software framework to facilitate this job.

We think that the solution to this problem is the creation of an information extraction framework where researchers can develop their proposals and where users can compare these proposals and select the most adequate information extraction algorithm for their case. Our proposal is to create an information extraction framework for semi-structured web pages. This framework reduces the costs of the integration of web information into business processes, reduces costs for developing new extraction algorithms and for testing and comparing these algorithms.

This paper is organised as follows. After showing the motivation in the introduction Section 1, Section 2 introduces information extractors for semi-structured web pages and then section 3 describes our information extraction comparison framework. Section 4 presents the conclusions and our future work.

2 Types and Techniques in Web Information Extraction

We can differentiate between three types of web pages: non-structured, semi-structured and structured web pages. Non structured web pages are those in which the info of interest is within free text. A common property for these types of pages is that given a set of similar non-structured web pages, no common pattern can be induced. In other words, free texts have no common pattern or rule that may help us to extract information from them. For this type of web pages, special information extractors are needed such as [23], [18] and [21]. These information extractors analyse and preprocess text, such as

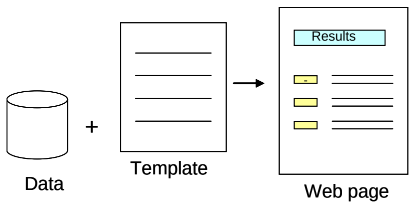


Fig. 1. Semi-structured web page creation.

performing Part of Speech Tagging (POS Tagging) and gaseeters or Entity recognisers, and then identify verbs, adjectives, actions, subjects, places etc.

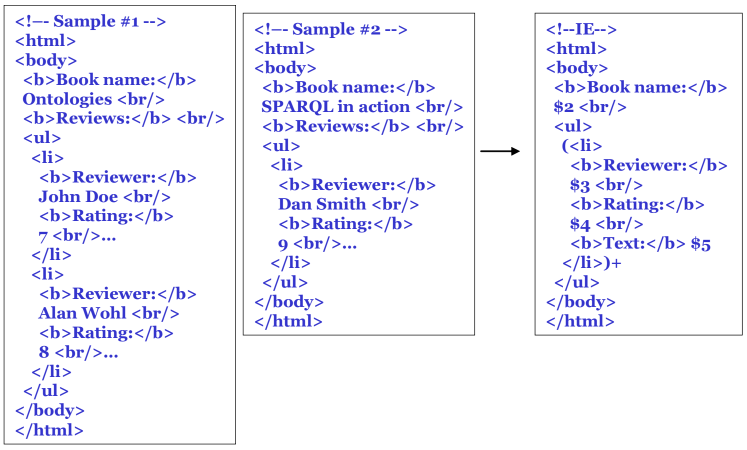
The second type of web pages is the one that is usually used to show a list of results as a response to a search such as a list of books at amazon.com or any other web store. This type of web pages is typically generated by using a template and a set of data, see Figure 1. In this case, given a set of semi-structured web pages, we can infer one or more patterns that may be used to extract information from this type of web pages. A common objective of many information extractors is to discover the template used to generate these web pages, and use it to detect the data and extract it.

The last type of web pages is the structured type. A structured web page is a web page that besides the presenting information in Html for human browsing, offers structured data that can be processed automatically by machines and are easily integrated into business processes.

Our study focuses on information extraction from semi-structured web pages. As we mentioned before, given a set of semi-structured web pages, information extractors for this type of pages try to extract useful data which is the result returned from the data set before presenting it. Information Extractors apply many techniques in order to separate data from html tags and useless content, such as advertisements. These techniques usually try to detect one or more patterns to identify and extract interesting data from these web pages.

Methods used by information extractors to learn patterns or to extract directly from web pages vary too much. Some information extractors consider web pages as a set of tokens and try to detect repeating patterns using string techniques (for example: [12] and [6]), others use DOM trees and try to detect repeating branches and children in these trees [5], some create and use statistical information to identify repeating zones and extract them directly [16] while others learns prefixes and suffixes for interesting data marked by user to apply them them over new web pages [15].

Besides classifying information extractors by input features, we can consider the the degree of automation for learning as another feature for classification. Information extractors usually learn a set of rules to use them later to extract information. We identified four types of learning algorithms: handcrafted, supervised, semi-supervised and unsupervised. Handcrafted algorithms are written manually for a special type of web pages and can only be used for similar pages having the same structure. Crafting such an information extractor is a hard task and any change in a web page may lead the information extractor to failure. Some information extraction algorithms need the user to



```
<!–- Sample #1 -->
<html>
<body>
 <b>Book name:</b>
 Ontologies <br/>
 <b>Reviews:</b> <br/>
 <ul>
  <li>
   <b>Reviewer:</b>
   John Doe <br/>
   <b>Rating:</b>
   7 <br/>...
  </li>
  <li>
   <b>Reviewer:</b>
   Alan Wohl <br/>
   <b>Rating:</b>
   8 <br/>...
  </li>
 </ul>
</body>
</html>
```

```
<!–- Sample #2 -->
<html>
<body>
 <b>Book name:</b>
 SPARQL in action <br/>
 <b>Reviews:</b> <br/>
 <ul>
  <li>
   <b>Reviewer:</b>
   Dan Smith <br/>
   <b>Rating:</b>
   9 <br/>...
  </li>
 </ul>
</body>
</html>
```

```
<!--IE-->
<html>
<body>
 <b>Book name:</b>
 $2 <br/>
 <ul>
  (<li>
   <b>Reviewer:</b>
   $3 <br/>
   <b>Rating:</b>
   $4 <br/>
   <b>Text:</b> $5
  </li>)+
 </ul>
</body>
</html>
```

Fig. 2. RoadRunner: an unsupervised information extractor.

mark the important data on some example web pages that are then used in the learning process. It is supervised if the user is required to mark the data to be extracted on sample pages; few authors distinguish between semi-supervised method in which few marking is required, and supervised, in which more marking is required. The last type contains the unsupervised information extractors, in this case, given one or more web pages, the learner automatically and without any user interference, identifies and extracts important information from these webs. An example of an unsupervised information extractor can be seen at figure **??** that induces extraction rules without and user interference.

3 Framework

Here we present the first approximation to our information extraction framework. We identified several modules, each one has a determined task. Some tools are integrated into our framework and are used to prepare web pages for learning extraction rules or even for extracting information from input web pages. These tools are: a tokeniser and an annotating tool [13]. Our tokeniser allows the user to define the tokenisation necessary for his or her extraction algorithm, this gives flexibility to integrate new algorithms into our framework and to define new ones. The annotating tool can be used to mark entities inside a web page and assign them a class or a relation in a predefined ontology. User can configure these tools according to the case of use, this provides high flexibility for developing and using new extraction algorithms. Here we describe the the most important elements:

– Preprocessor: Many information extraction algorithms preprocess web pages before learning or even before extracting information. We have identified two types of preprocessors: Transformer and Segmenter. A Transformer cleans Html code or performs part of speech tagging in case of free text information extractors, for this purpose HtmlTidy [17] is usually used. The segmenter depends on the extraction

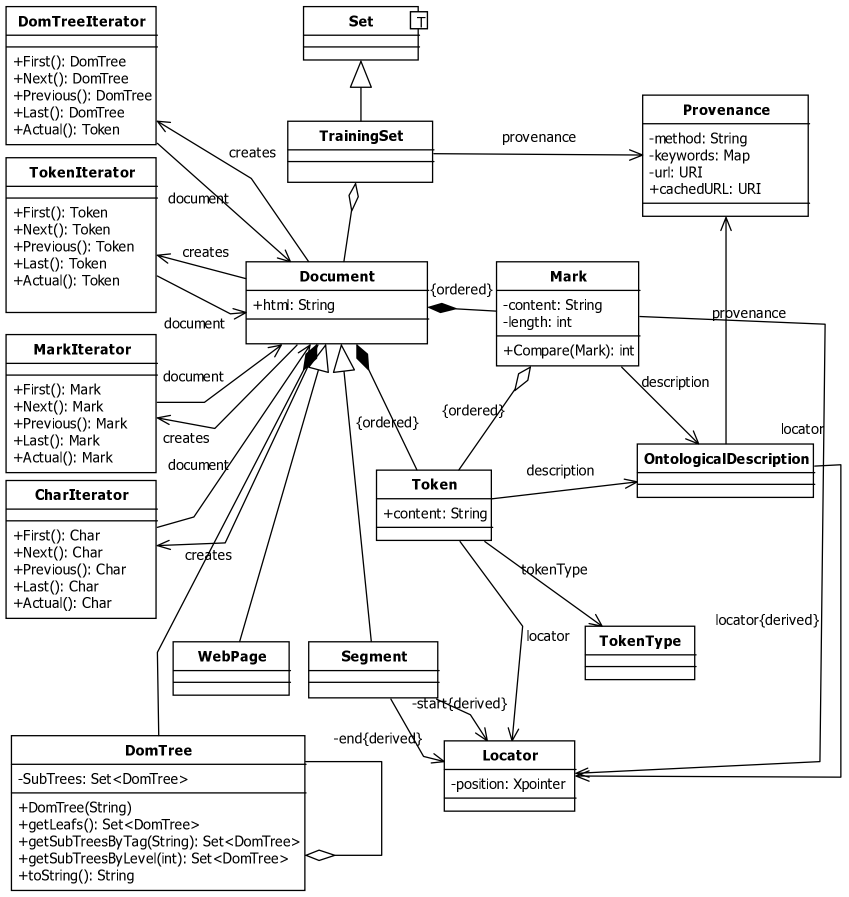


Fig. 3. TrainingSet is composed of a set of Documents.

algorithm. For example, some algorithms create clusters from web pages [14] as a first step, others separate DOM trees and remove unnecessary trees using [7].

– TrainingSet: This structure is the one that contains the web pages used to train information extractors creating extraction rules. See 3. A TrainingSet is a set of Documents which is the main class in this module. A Document is a web page or a segment of a web page. If an algorithm segments a web page before processing it, it creates new segments and adds them to the TrainingSet. The DOM tree of each document is obtained and the DOMTree structure is built. Before working with a Document, tokenisation should be performed using our previously mention tokeniser, creating and adding the sequence of tokens to each document. Besides tokenisation, the user should provide the Marks in the case the information extractor needs them to induce the extraction rules. These marks are added to the document creating instances of the class Mark whose OntologicalDescription is

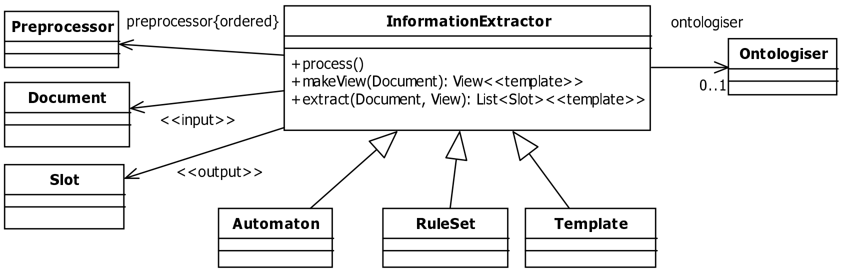


Fig. 4. Information Extractor.

assigned reading the marks ontology description provided by our annotation tool. Once a Document is marked and tokenised, the Learner Module can start working over the training set to discover extraction rules.

The ontologicalDescription class is used to indicate where the ontology that describes an item is, and which class inside this ontology or even which property inside a class this item represents. The Provenance of a web page indicates the URI of an Html file, and the way used to reach and download this web page. The Locator indicates the position of an item inside a Document or inside an ontology, Xpointer is for this purpose.

A segment has a Locator that indicates its start and end position in a given Document. The Locator also indicates which class or property inside an ontology provides semantics to a given mark. The TokenType is provided by our tokeniser and returns the type of the token depending on the it's configuration file. Iterators are used in order to get documents of a training set, marks or tokens of a certain document or even DOM subtrees of a given Document.

– Learner: Information extractors that learn rules use this module to deduce extraction rules. It provides to learners of extraction algorithms the capability of defining an ordered set of preprocessing steps before starting to extract from the input documents, and a set of classes with utilities such as a class to create clusters, generaliser or specialiser for regular expressions, also a String and a Tree aligner classes. Besides, this module contains template classes where predefined algorithms, such as Branch and Bound, are defined. Researchers should only give a body for some template methods in order to implement their proposals.

The result of this module is an object of the class InformationExtractor, see Figure 4. An information extractor has an ordered set of preprocessors that should be executed over input documents before starting the information extraction, besides, it contains some template methods that should be defined by each extraction algorithm. Once extraction is performed, an information extractor creates slots that contain the extracted data and uses an ontologiser to identify and label extracted data. We have identified three types of information extractors: Those that work as an automaton such as [11] and [9], others identify local patterns for each element to be extracted, for example the two information extractores: [19] and [15]. The third

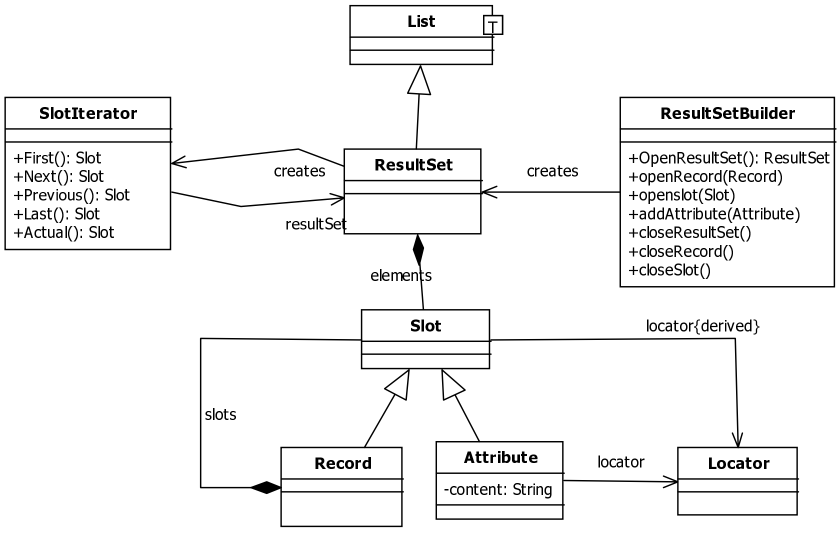


Fig. 5. ResultSet.

type of extractors include those that identify a template for the complete web page such as [24] and [14].

In the case of information extractors that work as an automaton, the created InformationExtractor is an automaton with a set of states and transitions with rules. A transition is performed when its rule can be applied, in this case, a command can be executed. This command indicates the action that should be executed when data is extracted, for example slots containing extracted data can be created and saved. InformationExtractor that learns patterns to identify some elements in a web page create a set of rules. These rules are noniterating rules, since each one is executed over all the document, extracting the zones where it matches, for example, a pattern to detect book titles matches 3 times in a web page and returns three titles. The third type identifies a template and is executed once over the input document, it identifies the element that should be extracted and returns a set of slots.

– WorkingSet: It contains a set of Documents, an InformationExtractor and a ResultSet. When an InformationExtractor is executed, the learned rules in a previous phase are executed over the set of Documents creating a ResultSet that is composed of a set of slots. To build a result set, ResultSetBuilder is used. Then, it uses the ontologiser to identify the classes and the relations between these slots, creating attributes and records then adding them to the ReultSet. At the end of the extraction, the ResultSet contains the extracted entities, each containing its attributes.

4 Conclusions and future work

Although there is a great number of information extraction techniques, none of them is applicable for all cases of information extraction. Users interested in these techniques should make a deep study of all systems before considering which information extraction technique should be used for their purpose. This study is tedious and is a resource consuming task. Besides, once the user has decided which information extractor use, integrating this tool into his business applications needs more effort increasing the web information integration costs.

Our framework contains common parts between the different information extraction algorithm, thus it supports reusability. New extraction algorithms can be implemented using our framework by reusing the developed modules and simply focusing on developing the main algorithm, so it reduces costs to develop new proposals. Testing and comparing information extractors are tasks that could be done using our framework since conditions are equal for all the tested algorithms. Using our information extraction framework, the effort used to compare different proposals is reduced since all algorithms are easily integrated, tested and compared between each others. Definitively, our framework can be considered as an important platform to reduce web information integration costs.

Once the framework is completed, the second step is to create a taxonomy for classifying and comparing information extractors, but this time, more features will be comparable since the recall and precision will be calculated using our framework, under the same conditions. Also, the framework will be used to define new information extractors and compare them with the existing ones.

References

1. Altigran S. da Silva Alberto H. F. Laender, Berthier Ribeiro-Neto. Debye - data extraction by example. *Data Knowl. Eng., Vol. 40*, 2001.
2. Altogran S. da Silva Juliana S. Teixeira Alberto H.F. Laender, Berthier A. Ribeiro-Neto. A brief survey of web data extraction tools. *SIGMOD*, 2002.
3. Casati F. Kuno H.-Machiraju V. Alonso, G. *Web Services Concepts, Architectures and Applications*. 2004.
4. Charles Schafer Andrew Carlson. Bootstrappiing information extraction from semi-structured web pages. 2008.
5. MA David Karger Andrew Hogue. Thresher: automating the unwrapping of semantic content from the world wide web. *ACM*, 2005.
6. Hector Garcia-Molina Arvind Arasu. Extracting structured data from web pages. *SIGMOD*, 2003.
7. Yanhong Zhai Bing Liu, Robert Grossman. Mining data records in web pages. *SIGKDD*, 2003.
8. Moheb Ramzy Girgis Khaled F. Shaalan Chia-Hui Chang, Mohammed Kayed. A survey of web information extraction systems. *IEEE*, 2006.
9. Ming-Tzung Dung Chun-Nan Hsu. Generating finite-state transducers for semi-structured data extraction from the web. 1998.
10. Dawn G. Gregg. Exploring information extraction resilience. *Journal of Universal Computer Science*, 2008.

11. Craig Knoblock Ion Muslea, Steve Minton. A hierarchical approach to wrapper induction. *SIGGRAPH*, 1999.
12. Fred H. Lochovsky Jiying Wang. Data extraction and label assignment for web databases. *Journal of Universal Computer Science*, 2003.
13. Agustín Domínguez Nicolás Griñolo José L.Alvarez, José L. Arjona. Annotator: Herramienta para la anotación semántica de islas de datos amigables en la web. *JISBD*, 2009.
14. Khaled Shaalan Moheb Ramzy Girgis Mohammed Kayed, Chia-Hui Chang. Fivatech: Page-level web data extraction from template pages. *IEEE*, 2007.
15. Robert Doorenbos Nicholas Kushmerick, Daniel S. Weld. Wrapper induction for information extraction. *IJCAI*, 1997.
16. Dimitrios Skoutas Nikolaos K. Papadakis. Stavies: A system for information extraction from unknown web data sources through automatic web wrapper generation using clustering techniques. *IEEE*, 2005.
17. Dave Raggett. *Clean up your Web pages with HP's HTML Tidy*. Elsevier B.V., 1999.
18. Ellen Riloff. Automatically generating extraction patterns from untagged text. *AAAI96*, 1996.
19. Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning, vol. 34*, 1999.
20. Ross Tredwell Stefan Kuhlins. Toolkits for generation wrappers. *Net.ObjectDays*, 2002.
21. Jonathan Aseltine Wendy Lehnert Stephen Soderland, David Fisher. Crystal: Inducing a conceptual dictionary. 1995.
22. Ora Lassila Tim Berners Lee, James Hendler. The semantic web. *Scientific American*, 2001.
23. Sriram Raghavan Shivakumar Vaithyanathan Huaiyu Zhu T.S. Jayram, Rajasekar Krishnamurthy. Avatar information extraction system. *IEEE*, 2006.
24. Paolo Merialdo Valter Crescenzi, Giansalvatore Mecca. Roadrunner: Towards automatic data extraction from large web sites. *VLDB*, 2001.