WebJDBC Relational Data Extractor

Daniel Santana
daniel.santana@ist.utl.pt
Instituto Superior Técnico

Abstract. The main goal of a JDBC driver is accessing databases in a Java application. This work proposes accessing web pages using a relational database abstraction, which is very important when building applications that require integrating data from multiple data sources such as web data. However, extraction of data from web pages lists and tables is not a trivial task. There are many web pages with different HTML structure and consequently with different data representations. Our goal is contributing with the construction of a WebJDBC driver that allows the systematic and structure access of data from web pages in Java applications. The WebJDBC driver construction was split in two different tasks: (i) the implementation of a relational data extractor for web pages lists and tables (ii) the implementation of a query processor that is capable of evaluating SQL queries over data returned by the extractor. This work focuses the implementation of the relational data extractor and the studies existing approaches for web data extraction. The work achieved the integration of multiple techniques to the Relational Web Data Extractor, that component is designed to ease the integration of new techniques.

1 Introduction and Motivation

The JDBC driver is a software component that enables a Java application interaction with a database. JDBC driver allows the access of distinct information sources, usually relational databases such as MySQL, Oracle, SQLServer, by using an uniform interface\(^1\). For each type of information source, there is the need of a driver construction accordingly JDBC API \([8]\) specifications.

Initially, the drivers were essentially constructed with the objective of integrating relational databases. However, some JDBC drivers appeared, with the purpose of integrating other sources of information, such as CSV and XML files\(^2\) \([12]\).

There is no information about JDBC drivers for web pages or its study. Web pages can be seen as HTML\(^3\) documents rich in information. Their abundant

---

\(^1\) See more information about JDBC in [http://java.sun.com/products/jdbc/overview.html](http://java.sun.com/products/jdbc/overview.html)


\(^3\) HTML definition in [http://www.w3schools.com/html/html_intro.asp](http://www.w3schools.com/html/html_intro.asp)
existence over internet is a great factor of motivation for the implementation of an WebJDBC driver that allows integration of web page information in Java.

This work focus on data extraction from the Web. There are many web pages that present list or tables of products, like www.amazon.com or www.ebay.com and their data can be extracted. Our main challenges were the study of the existing extraction techniques and the creation of a Relational Web Data Extractor that allows the integration of new extraction techniques. Other objective is that the integration of the Extractor with the WebJDBC driver causes the minimum impact on the driver usability and functionality.

The document is structured in the following way: Section 2 presents some relevant concepts related with web relational data extraction and the state of art of existing extraction techniques or algorithms. Section 3 presents details about the created solution, along with the decisions taken in the implementation and design process. Section 4 presents a description about the evaluation methodology to validate the Extractor and the obtained experimental results. Finally, Section 5 presents some of the important conclusions and ideas of this work and possible future work.

2 Concepts and Related Work

This section describes some important concepts about web data extraction and the related work, which will be important to be aware for better understanding of the following sections.

Web page is a document written in HTML code. HTML is a markup language and uses markup tags to describe web pages. Web pages can represent data in a structured, semi-structured or unstructured way. We considered mainly the structured or semi-structured representations. An example of a web page is shown in Figure 1.

Data records are items of a data and are normally contained in a Data Region. For better understanding of the following descriptions, the Figure 1 must be seen. The Data Record concept is known in [17,18,21,10,1] works. We can extract relational tuples from data records. For example, from a data record having a product and containing fields representing name, brand and price, the following relational tuple can be extracted: product(name,brand,price)

The Data Region is of the biggest regions in a web page, which is rich in data. This concept is known in [1,17,18,21,10,20] works. They can represent lists or tables of items. The identification of these regions is useful for web data record extraction, in order to avoid irrelevant information for the user (such as advertisements or navigation bars), and focus only in finding relevant information (such as data records).

Some web data extraction approaches use a wrapper. This is a procedure that automatically extracts data from web pages. Wrappers can be constructed manually or automatically. Many approaches generate the wrappers automatically based on examples or on the discovering of patterns in web pages [6,5,22]. One example of an wrapper is a state machine.
Fig. 1. Web Page sample - This figure shows a list of products, which have some information that can be extracted. Data Regions are composed by Data Records. Flat Data Record contains only one record. Nested Data Record contains multiple records. Data Record Fields compose the data records and are worthy for extraction.

Usually, either the web data extraction approaches use the raw text of HTML code from webpages or an HTML tag tree obtained by parsing the HTML code of a web page. A DOM Tree is an example of this type of representation.4

There was a study of existing techniques to support the creation of an Extractor, in order to adjust the Extractor to the reality of the web data extraction. This Extractor must also be adapted to the necessities of a WebJDBC driver. In [15] we can see a brief description of some existing techniques grouped by classes, such as: HTML-Aware techniques, Natural Language Process based techniques, Wrapper induction techniques, Modelling based techniques, Ontology based techniques. We proposed another classification for this techniques that is more appropriate for our goals, because we must be aware of the differences between the techniques concerning input, output and the way they work.

Some approaches try to find patterns in a web page and they only need one single web page as input. Commonly, they have data region identification step and a data record identification and extraction step. We classified them as Autonomous Pattern Discovery Techniques. This kind of techniques make some assumptions about the web pages, based on observations. Most of them consider that there is a data region with data records in a web page [17,10,1,20]. They also assume that there is a similar pattern between data records, which leads the use of similarity and alignment techniques to identify data regions and data records [18,21]. There are also techniques without a conventional data region/data record identification [3,2]. These techniques use a suffix tree to find patterns in web pages. Their characteristics can be seen in Table 1.

---

4 DOM Tree information in http://www.w3schools.com/html/dom/dom_nodeTree.asp
Other class of techniques use a set of training examples for generation of a wrapper to extract data records or their fields. These techniques have a learning process or algorithm that receives a set of training examples as input and also have other process that use training examples for the creation of a wrapper. These techniques, we classified as the Machine Learning Techniques. Normally the training examples are web page samples labelled by users. The techniques that use labelled training examples, we classified as Supervised Machine Learning Techniques. The works [19,11,13,6,4,14,22] belong to this subclass. The techniques that do not receive labelled training data and learn based on a set of web pages, we classified as Unsupervised Machine Learning Techniques such as RoadRunner [5]. These technique characteristics can be seen in Table 2.

Finally, there is an alternative approach that uses semantic models. This approach uses a set of known entities and relationships, as ontologies, to identify data and infer relational tuples in a web page. These, we classified as Semantic Model Techniques. An ontology [9] is a representation of knowledge through a set concepts and the relationships between them. An ontology describes a certain domain through this concepts and relationships. Semantic knowledge bases, like YAGO, are repositories of entities and facts about them\(^5\). Semantic models are used in some approaches of web data extraction such as [7] and [16]. These technique characteristics can be seen in Table 3.

\(^5\) YAGO is an example of a semantic knowledge base, see [http://www.mpi-inf.mpg.de/yago-naga/yago](http://www.mpi-inf.mpg.de/yago-naga/yago)
### Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Semantic Model Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Complexity</td>
<td>VH</td>
</tr>
<tr>
<td>Output Format</td>
<td>data base tables</td>
</tr>
<tr>
<td>Labelling</td>
<td>yes</td>
</tr>
<tr>
<td>CPU Time Order</td>
<td>N/A, ms</td>
</tr>
</tbody>
</table>

Table 3. Semantic Model Features - N/A: Not Applicable, VH: Very High (Provide, edit create semantic models such as ontologies), ms: milliseconds, sec: seconds, min: minutes.

### 3 Solution Design and Implementation

The solution for this work was the creation of an extractor that is able to provide web data extraction services to a *WebJDBC driver*. Figure 3 shows that there are two important interfaces for *WebJDBC driver - Extractor* communication. One, is the *Model* that is created by using the input provided by a *SQL Query*. The *Model* is used by *Extractor* to create a set of parameters for the techniques that perform the extraction. The other is *MetaDataCursor* that is a representation of the logic for web data extraction for *WebJDBC driver*. The implementation of *MetaDataCursor* is responsible for the creation of connections with web pages from *Web*, and is responsible for the web data extraction. *WebJDBC driver* can request *MetaDataCursors* from *Extractor*.

![Diagram](image)

**Fig. 2.** General Project Integration - Diagram - *WebJDBC driver* can request *Models* creation and *MetaDataCursors* for extraction, accordingly of the input of a *SQL query*.

The extractor is composed by 3 main classes, *ModelManager, TechniqueFactory* and *Technique* as we can see in figure 3. *ModelManager* is responsible for the creation and storage of technique parameters. The creation of technique parameters is done by using models that are provided by the *WebJDBC driver*. *TechniqueFactory* uses the created parameters to instantiate a *Technique* whenever
WebJDBC driver requests table(s) from a web page, i.e. MetaDataCursor(s). Finally the extractor uses the Technique to create an array of MetaDataCursor(s), which are the representation of web page table(s) for the WebJDBC driver.

The separation of the technique parameters creation and the techniques creation/utilization allows to split the time and the complexity of parameters construction and techniques extraction and the reusing of technique parameters, which provides time saving. This way WebJDBC driver can provide two types of SQL Statements for the users, one for creation of the technique parameters (Models) and other for the technique utilization (Normal SQL queries). We can see the creation of technique parameters as creation of indexes for a database. Indexes increases the performance in some SQL queries, and technique parameters have similar function. They increase efficiency and efficacy by helping the creation of appropriated extraction techniques, accordingly the web page.

Fig. 3. Extractor UML Diagram - Extractor uses ModelManager to create and reuse TechniqueParameters that will be used by TechniqueFactory to create a Technique, which performs the web data extraction. Technique implementations also implement the MetaDataCursor interface to comply with the WebJDBC driver

There were implemented 3 techniques to integrate in the Extractor. One is an Autonomous Pattern Discovery Technique. The technique was based on work [17] on aspects of data region and data record identification. The technique is composed by components that can have multiple implementations. This allows the extraction process having multiple configurations and/or heuristics. One component detects repeated patterns in a web page using the similarity technique
of \textit{string edit distance}, others identify and choose a \textit{data region} or more from web page and finally there is one component that extract the \textit{data records}. This technique only needs an \textit{URL} as input and convert the \textit{HTML} code of the page in a \textit{HTML} tag tree, which is used in processing of data extraction.

Other technique that was implemented uses \textit{regex} to extract data from web pages. Basically, for each field, which we need to extract from the webpage, is assigned a \textit{regex}. The technique consists in the creation of \textit{regex} and its compilation and the use or reuse of the created \textit{regex}.

Finally, a \textit{Supervised Machine Learning Technique} was developed too. The implementation of this technique was based on work [11]. Examples are created in one example file by the users and then examples are used to create a state machine that is able to extract data from web pages. Examples are composed by tokens that exists in \textit{HTML} pages, such as alphanumeric words, numbers, symbols, tags etc. The state machine is created based on the example tokens to match and extract the data from the web page. The data is matched and extracted by detecting the tokens that precede the data and tokens that succeed the data.

4 Experimental Results

We choose to evaluate mainly the usability of the \textit{WebJDBC} driver because it is related with the type of input of the extraction techniques. The quality of the results obtained by \textit{WebJDBC} driver users and performance of each query of the \textit{WebJDBC driver} were also measured.

The evaluation process consisted in giving some tasks for a group of users. The task involved the creation of some \textit{SQL} queries that could retrieve specific information from a web page. To the user was given the target web page, which contains the information for extraction, and the pretended output information. Figure 4 represents the task flow diagram.

Three of the used web pages are from the web and other three, \textit{e.g} \textit{bigbook}, are from known data sets in information extraction domain. These data sets are known as \textit{RISE} and Patrick Marty Data sets and they are available online \footnote{Visit http://www.isi.edu/info-agents/RISE/repository.html and http://www.grappa.univ-lille3.fr/marty/corpus.php for more information about RISE and Patrick Marty data sets documentation and download, respectively.}

There were 6 tasks in total and they were performed by 8 users. Each task had a time limit of 10 minutes. A profile form was created for the users. Users had an age average of 23 years old and only one was a female. They all studied \textit{Informatics Engineering}. Finally, they all had knowledge and were comfortable with \textit{SQL} language but only 2 were comfortable with the use of \textit{regex}.

The results shown the users preference of the technique in this order (descending): 1\textsuperscript{st}. Autonomous technique, 2\textsuperscript{nd}. Supervised technique, 3\textsuperscript{rd}. Regex Technique. All users used Autonomous technique in the beginning of the tasks.
Although the profile form indicated that almost all users had at least basic knowledge on using regex, most of the users preferred widely the use of the Supervised Technique when confronted with the choice of using the Supervised or Regex Technique. This facts indicates that is more user friendly the creation of examples, by creating an example file using samples/tokens from the HTML code, than creating regex to extract the data.

Most of the syntax errors occurred in tasks which required more complex SQL statements. The users took more time in completing the tasks that require the creation of models and/or complex SQL queries. This time consumption in the creation of models happened because of the necessity of the HTML code analysis. Finally, all SQL statements performed in a time lower than 1 second. The final task involved a webpage with more complex structure, which had an impact in the performance of the Autonomous Technique. The average time of execution of the Autonomous Technique was around 9 seconds in this case, which is longer time than the time needed for the extraction of 53 records from a webpage by the other techniques. No user chose this other techniques.

5 Conclusions and Future Work

The study of different approaches of webdata extraction and the design and the development of our solution lead us to some conclusions.
Integrating techniques in a single component is possible. We created that component and it integrated multiple techniques, by the standardization of the creation of technique parameters, which required different types of input. This fact causes less impact in the WebJDBC driver functionality and usability.

Separating the techniques in multiple components, which can assume different implementations, is useful for creating a dynamic process of extraction. This dynamic allows the users or automatic algorithms to choose the best implementations accordingly the webpage structure.

The separation of the creation of technique parameters from the use of the techniques allows the reusing of the parameters and the splitting of the process of using the WebJDBC driver in 2 different phases that require different knowledge. The phase of creation of technique parameters could be done by users that can have a role similar of a database administrator. These users could have specific knowledge of the technique concepts. The other phase could be done by users that have more knowledge in SQL, allowing the use of standard SQL queries.

For future work, the addition of created or existing techniques is possible, as long they comply both of the interfaces provided by the Extractor and the cursor of the WebJDBC driver.

Currently, databases have a query optimizer that traces query plans, which allow choosing better algorithms to execute the query. Something similar can be done for the WebJDBC driver or the Extractor or both. The idea is creating algorithms that decide which is the best technique and technique configuration for each extraction. The users would have the options of using the algorithm or giving hints in the query to influence the selection of the technique.

References


