

The DEByE Environment for Web Data Management*

Alberto H. F. Laender Altigran S. da Silva[†] Paulo B. Golgher
Berthier Ribeiro-Neto Irna M. R. Evangelista-Filha Karine V. Magalhães

Department of Computer Science
Federal University of Minas Gerais
31270-901 Belo Horizonte MG Brazil
laender@dcc.ufmg.br

Abstract

A large portion of the Web is composed of pages that can be regarded as containers of useful semistructured data that, although not readily available for automated processing, can be identified, extracted, and manipulated independently. In this paper we present DEByE, an approach for Web data management in which data obtained from the Web is represented by means of nested tables allowing variations. As we discuss throughout the paper, such tables are intuitive and simple enough to be used as a metaphor for user graphical interfaces, and flexible and powerful enough to represent hierarchical objects with structural variations typical of semistructured Web data. Based on this approach, we have designed and implemented a set of tools for extracting data from Web pages, querying the extracted data, and storing this data in relational databases. This set of tools composes what we call the DEByE environment for Web data management.

Keywords: DEByE, Web data management, Web data extraction, Semistructured Data;

1 Introduction

Over the last decade, the astonishing growth of the World-Wide Web (number of users, servers, on-line services, etc.) made it clear that it would soon become a huge repository of data of interest for a variety of application domains. However, the same features that have made the Web so useful and popular also impose important restrictions on the way the data it contains can be manipulated. Particularly, in the traditional Web scenario, there is an inherent difficulty in gaining access to structured data which is present in Web pages but is not readily available.

Indeed, despite the attempts of defining standards such as XML and RDF to provide some form of structure to Web data, most of the Web is still composed of HTML pages, either static or automatically generated. It is worth noticing that the spreading of such standards does

*This work was partially supported by Project SIAM (MCT/CNPq/PRONEX grant number 76.97.1016.00) and by CNPq (grant number 467775/00-1).

[†]On leave from the University of Amazonas, Brazil.

not provide a trivial solution to the problem of manipulating existing semistructured Web data, since the volume of HTML pages currently available is enormous and is still increasing. This suggests that most users will continue to access data in HTML format.

This inherent unstructured characteristic of the data found on the Web is in a great share due to the nature of its documents (e.g., online books, newspaper and scientific articles, technical brochures, etc.). However, a large portion of the Web is composed of pages that can be regarded as “data containers” in the sense that they implicitly contain data that can be identified, extracted, and manipulated independently. A common feature of such pages is that their data present an inherent structure that, although not explicitly described, can be recognized by a user looking at them through a browser because of visual “clues” such as colors, fonts, bullets, and indentations provided by the pages’ author to help the user examine their contents.

Because such pages are constructed in non-rigid textual formats (e.g., HTML, XML), the structure of the data they implicitly present is also non-rigid. This means that structural variations on the data can occur and should be tolerated and treated accordingly. In the recent literature data presenting implicit and irregular structure has been called *semistructured* [3], while the Web sites or services providing pages that implicitly contain data of interest are usually called *data rich* [7] or *data-intensive* Web sources.

One of the most important features of modern information systems is the capability of manipulating data from distinct Web sources. Indeed, the range of applications requiring the management of Web data is enormous and, therefore, this subject has attracted considerable attention from the database research and industrial communities [10].

Data rich Web sources providing semistructured data are the main target of the work we present in this paper. In particular, we are interested in the problems of recognizing the implicit structure of the data, extracting the data from its original sources, logically representing the extracted data in a suitable format, and further manipulating such data. For this, we base our work on an approach that relies on the user’s perception of the structure and semantics of the data as it appears in its original sources. This approach is called *DEByE* (*Data Extraction By Example*)¹[12], and is introduced in the immediately following.

1.1 Motivating Example

Examples of Web pages containing semistructured data are those returned by a query in the *Amazon.com* Web site. A snapshot of one of such pages is presented in Figure 1, which results from a query with the argument “Paul McCartney”. To simplify our example, we consider in this page only products from the “stores” *popular music*, *books*, and *auction*. Note that, for each store, the information on products is distinct.

Suppose an application in which it is necessary to monitor new releases, price changing, new auction offerings, etc., of products related to “Paul McCartney”. This would require a user to frequently visit the site, issue a query, record the data of interest returned, and compare it with data previously recorded. If the volume of products to be monitored grows, the amount of manual effort required might become nearly unbearable.

In our work, we propose that problems such as this should be solved by extracting the data from the Web site of interest and storing it locally in a suitable form (e.g., relational tables, XML, etc.), and then performing sophisticated queries and other operations such as

¹This name is an homage to Moshé Zloof, creator of QBE, who suggested us the paradigm we use to represent and specify the data to be extracted from Web sources.

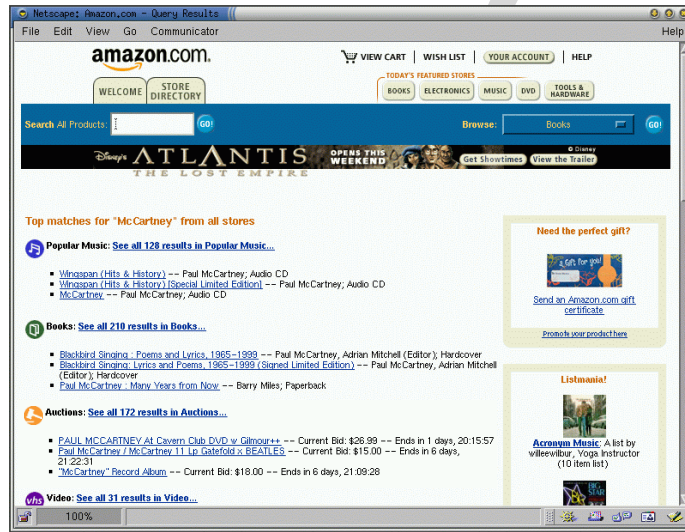


Figure 1: Snapshot of a page from the Amazon.com Web site.

data integration and view generation. According to our approach, this is done based on the user's perception of the structure of the data in the Web pages. This perception is captured by a *modeling* proposed by the user, in a way similar to what happens in traditional databases. To cope with the complexity and irregularities common to typical Web data, we propose this modeling to be done by means of *nested tables* that allow distinct "rows" to have "cells" with distinct internal structure for a same column. Experimental results on the use of such structures have shown that they are simple yet representative enough to model typical semistructured data found in Web pages [12]. Figure 2 shows a nested table containing part of the data in the page of Figure 1. Notice that, for each store, column Info stores an internal table with a distinct structure.

Store	Info		
<i>Popular Music</i>	Title	Artist	AudioType
	<i>Wingspan (Hits & History)</i>	<i>Paul McCartney</i>	<i>Audio CD</i>

<i>Books</i>	Title	Authors	BookType
	<i>Blackbird Singing ...</i>	<i>Paul McCartney Adrian Mitchell</i>	<i>HardCover</i>

<i>Auctions</i>	Item	Bid	Time
	<i>PAUL MCCARTNEY At ...</i>	<i>\$ 26.99</i>	<i>1 days, 20:15:57</i>

Figure 2: Example of a nested table.

The DEByE approach adopts nested tables as its fundamental concept. In fact, it proposes a data model in which implicit complex objects commonly found in Web pages can be seen as rows of a nested table allowing internal variations, such as the one presented in Figure 2. Based on this approach, we have designed and implemented a set of tools for extracting data from Web pages, querying the extracted data, and storing this data in relational databases. This set of tools compose what we call the DEByE environment for Web data management.

1.2 Overview of the DEByE Environment

In Figure 3 we illustrate the tools we have implemented based on our approach and show how they are related to each other. In what follows, we briefly describe these tools.

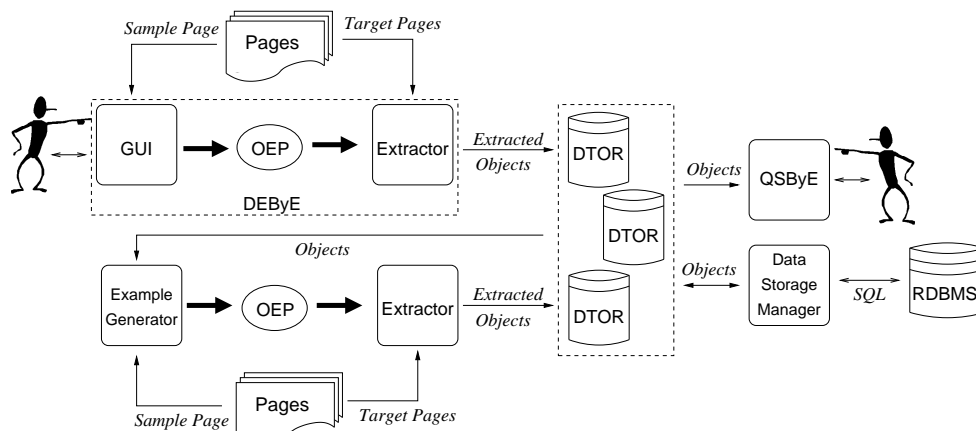


Figure 3: The DEByE Environment.

For the task of data extraction, we have developed a tool, called *DEByE*, that, given as input a sample Web page and a set of examples of the objects of interest present in this page, generates an *object extraction pattern* (*oe-pattern* or just *OEP*), which indicates how to locate and structure the objects implicitly present in the pages of interest. For helping users in the task of specifying example objects, this tool features a graphical user interface (GUI) that allows one to assemble rows of a nested table by cutting and pasting data values from the sample page, each row corresponding to an example object. The *oe-pattern* is fed to a generic *Extractor* and serves as a guide for extracting new objects from pages similar to the sample page. The *Extractor* outputs the extracted objects in an XML-based format creating what we call a *DTOR* (*DEByE Textual Object Repository*).

Example objects can also be automatically generated if a data repository containing objects from the same domain of the sample Web page is available. This is possible by using a method we developed for bootstrapping the example-based data extraction [11]. This method allowed us to implement the *Example Generator*, which helps making the extractor resilient (i.e., immune to changes in the formatting features of the source pages) and adaptive (i.e., capable of working with pages from distinct sources belonging to a same application domain). Based on the example generated, the *Example Generator* outputs an *oe-pattern* (in the same way as the *DEByE GUI*) to be used by the *Extractor*.

One of the advantages of using nested tables for representing the extracted data is that they allow us to extend well known query operations for nested tables to deal with inter-

nal variations as defined by the DEByE data model. These operations were implemented in a graphical query interface suitable for semistructured Web data, which we call *QSBByE* (*Querying Semistructured data By Example*) [8]. QSBByE combines features of QBE (*Query By Example*) with typical features of query languages for semistructured data [1]. In particular, QSBByE provides the structure of the data as a nested table “skeleton” so that users do not have to uncover the structure of the data by themselves.

Similarly, the underlying tabular structure of the data we manipulate simplifies the task of storing it into relational databases. Relational databases have been explored as an alternative to store semistructured data [6, 9] because they can efficiently manage huge volumes of data. Therefore, taking advantage of our underlying data model, we implemented a mechanism for storing and retrieving semistructured data in relational databases, which we call the *DEByE Data Storage Manager* [14].

1.3 Paper Outline

In this paper, we discuss the DEByE approach and the DEByE environment, with emphasis on the issues of using nested tables allowing variations as a paradigm for supporting several solutions we propose to problems related to Web data management. The focus therefore is not on the technical novelties of the DEByE approach, which have been presented elsewhere [8, 11, 12, 14], but rather on the deployment of this approach as a suitable alternative for dealing with semistructured data from the Web. Additionally, we present the various tools we have developed based on the DEByE approach, locating them within a single framework, what places them in a perspective not readily apparent in our previous publications.

The remainder of this paper is organized as follows. Section 2 presents the DEByE data model. In Section 3 we discuss issues related to data extraction in the DEByE approach. Our bootstrapping method is addressed in Section 4. In Section 5 we describe the querying facilities provided by our environment. In Section 6 we discuss how semistructured Web data organized according to the DEByE data model can be stored in relational databases. Related work is discussed in Section 7. Finally, Section 8 summarizes the paper and presents some conclusions.

2 The DEByE Data Model

In this section, we discuss the data model we adopt in our environment. This data model is based on the notion of *nested tables*, extended to allow internal variations. Our main goal is to use nested tables for representing and querying semistructured Web data.

Representing semistructured Web data as nested tables is useful because such structures are simple, intuitive, and convenient for treating multi-level hierarchical data. The main problem we have in achieving our goal is dealing with irregularities typical of semistructured data. As a solution, we propose a generalization of regular nested tables in which we allow a column to have two or more distinct substructures. An example of this solution is presented in the nested table in Figure 2.

Despite the relative simplicity of representing semistructured data as nested tables, the representation is not as expressive as general semistructured data models (e.g., OEM [1]) or XML. Nested structures are expressive enough, however, for representing data available in Web pages, like the one shown in Figure 1.

To characterize these ideas more precisely, we define the notion of a table scheme. A *table scheme* τ is defined using the notation: $\tau = (C_1 : [\tau_1^1; \dots; \tau_1^{n_1}], C_2 : [\tau_2^1; \dots; \tau_2^{n_2}], \dots, C_m : [\tau_m^1; \dots; \tau_m^{n_m}])$, ($m \geq 2, n_k \geq 1, k = 1 \dots, m$), where C_j is called a *column* and τ_j^i denotes exactly one of the following: (i) an atomic value, represented by `atom`, (ii) a set of atomic values, represented by `{atom}` or (iii) a table scheme. For the sake of simplifying the notation, if $n_j = 1$, we can use $C_j : \tau_j^1$ instead of $C_j : [\tau_j^1]$. Also, if $C_j : \text{atom}$, C_j is said to be an *atomic type*.

Intuitively, a table scheme describes the structure of a kind of nested table in which a column C_j may store “values” or objects with distinct structure in distinct tuples. The structures of the possible objects are given by the alternatives $\tau_j^1, \dots, \tau_j^{n_j}$ which can be either atomic values, lists of atomic values, or other nested tables. For instance, the structure of the objects implicitly present in the page excerpt of Figure 1 can be described by the following table scheme:

```
ProductList = (Store:atom, Info: [(Title:atom, Artist:atom, AudioType:atom);
                                (Title:atom, Authors:{atom}, BookType:atom);
                                (Item:atom, Bid:atom, Time:atom)])
```

Notice that the nested table in Figure 2 is an instance of this table scheme.

3 Data Extraction

The first step to be addressed when managing Web data is the translation from its original textual format to a suitable format for further processing (e.g., XML, relational tables). This task is performed by the so-called *wrappers*. Formally, the problem of wrapper generation can be represented as follows. Given a Web data source S containing a set of pages T , determine a mapping w that is capable of populating a repository R with a set O of objects (data items) extracted from the pages in T .

The mapping w is, in general, a set of rules or text patterns used to recognize (among other uninteresting pieces of text) attribute values for objects of interest, associating an appropriate semantics to them. Based on this definition, we can say that a *wrapper* is an implementation of the mapping w . In our approach, the mapping w corresponds to what we call an object extraction pattern (oe-pattern).

Once generated, the oe-pattern is read by the DEByE Extractor, that parses its rules to guide the extraction process over pages given as input. Thus, the Extractor can be considered as a generic wrapper that is configurable. More precisely, an oe-pattern can be regarded as a grammar and the Extractor as a parser for this grammar [12].

Within the DEByE environment, the primary method for generating an oe-pattern is by using the DEByE GUI, through which users can provide examples by assembling a nested table that captures the structure of the objects to be extracted. This feature is key in our approach, since it allows the user to implicitly inform in a single example both the syntactic context of the objects of interest and their structure. For instance, the page of Figure 1, which contains non-homogeneous objects, requires the user to provide at least one example object for each one of the possible types of product. In this way, the user implicitly specifies each one of the possible structural variations an instance of `Store` may have. We notice that, in practice, few examples are needed as our experiments demonstrate [12].

In the following sections we first describe the generation of oe-patterns, then we discuss the extraction strategy used by the DEByE Extractor, and, finally, we briefly describe the DEByE GUI.

3.1 Object Extraction Patterns

Given the examples provided by the user, the DEByE GUI generates the corresponding object extraction pattern (oe-pattern). Essentially, an oe-pattern encodes two kinds of information necessary for guiding the process of data extraction: the structure of the objects of interest, in the form of a table scheme (see Section 2), and the textual surroundings (i.e., markups, symbols, keywords, etc.) of the atomic values of the attributes that compose such objects. We refer to these values as *attribute-value pairs*.

More precisely, an *oe-pattern* is a pair $\langle \tau, \rho \rangle$ where τ is a table scheme and $\rho = \langle a_1, \pi_1 \rangle, \dots, \langle a_n, \pi_n \rangle$ ($n \geq 1$) is a list of pairs $\langle a_i, \pi_i \rangle$ called *attribute-value pair patterns* or *avp-patterns*, where a_i is an atomic type that is part of τ and π_i is a regular expression for finding values of type a_i in the pages given as input.

For generating the regular expressions π_i , the surroundings of each value of a_i given in the example is generalized, in such a way that each π_i is able to recognize the complete set of values that share a common textual context. In the case of our example, for each atomic type composing the table scheme `ProductList`, there will be at least one avp-pattern that describes the context in which values of the type occur in the target pages. For the sake of simplicity, we omit the discussion on how the regular expressions π_i are generated. We refer the interested reader to [12] for a complete discussion on this issue.

3.2 The Extraction Strategy

Given an oe-pattern and a set of Web pages as input, the DEByE Extractor uses an efficient bottom-up procedure to perform the extraction process. The main feature of this bottom-up extraction procedure is that it first recognizes and extracts atomic components, and then uses them to assemble the complete object through a bottom-up composition operation. In what follows, we provide an example of how the bottom-up procedure works. A complete discussion of this procedure is presented in [12]

Consider we are interested in obtaining instances of objects of the following type (table scheme) `Author:(Name:atom,{Book:(Title:atom,Price:atom)})`, that is, the name of an author along with a list containing the title and the price of his books. Also, suppose that the extraction procedure receives as input the oe-pattern $\langle \tau, \rho \rangle$, where $\rho = \langle \text{Name}, \pi_{\text{Name}} \rangle, \langle \text{Title}, \pi_{\text{Title}} \rangle, \langle \text{Price}, \pi_{\text{Price}} \rangle$. The steps executed during the extraction process are illustrated in Figure 4. In this figure, circles represent objects extracted or being assembled.

The first step of the extraction process consists in using the avp-patterns to find instances of Name, Title, and Price. The result of this step, called the *Extraction Phase*, is illustrated by the lowest row of circles in Figure 4, which represents a list containing only attribute-value pairs. In this list, these pairs are ordered by a label l_i , that indicates the position in the source page of the string corresponding to each one of them.

In the second step, contiguous pairs of Title and Price values are combined to form Book instances. Each of these instances is labeled with the smaller l_i . Notice that the Book instance labeled l_9 is built with components whose order is distinct from the previous ones. Also, the Book instance labeled l_{11} is missing its Price component. The capability of dealing with such

situations is an important feature of our bottom-up strategy. The list of objects assembled after the completion of the second step is represented by the second lowest row in Figure 4.

In the third step, `Book` instances related to a same instance of `Author` are collected together in a list, referred to as `{Book}`. In the fourth step, each one of those lists is combined with an instance of `Name` (previously extracted), to assemble `Author` instances.

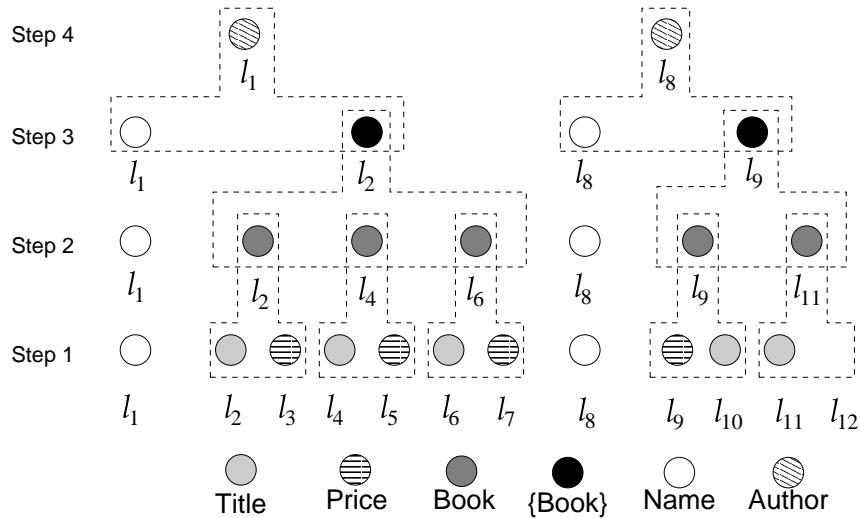


Figure 4: Execution of the bottom-up procedures.

3.3 The DEByE GUI

To help the user in the task of specifying example objects, the DEByE tool includes a graphical user interface (the DEByE GUI) that is based on nested tables. A snapshot of an example specification session for the sample page presented in Figure 1 is shown in Figure 5.

To assemble an example, the user marks pieces of data from the page in the *Source Window* and copies them into the columns of a table in the *Example Window*. This is done separately for each piece of text. The tool also provides a number of column operations such as *insert*, *remove*, *rename*, *group*, and *split* (not shown in Figure 5). These operations allow the user to build a nested table that embeds the structure of the example object. In Figure 5, the user has assembled three distinct rows, each one corresponding to a type of product. Note that the resulting nested table resembles the one presented in Figure 2.

Once the user has specified his example objects, he activates the *Pattern* button (in the function bar) to build the corresponding oe-pattern to be used by the Extractor.

4 Wrapper Resilience and Adaptiveness

Traditional example-based approaches for wrapper generation share two undesirable characteristics. First, changes in the format or layout of the target Web pages may result in a bad functioning wrapper, since the textual contexts used to recognize values of a given type may change. Second, wrappers are specific to a single Web data source. Thus, a wrapper for a given Web source cannot be used for another one, even if they have the same domain. These

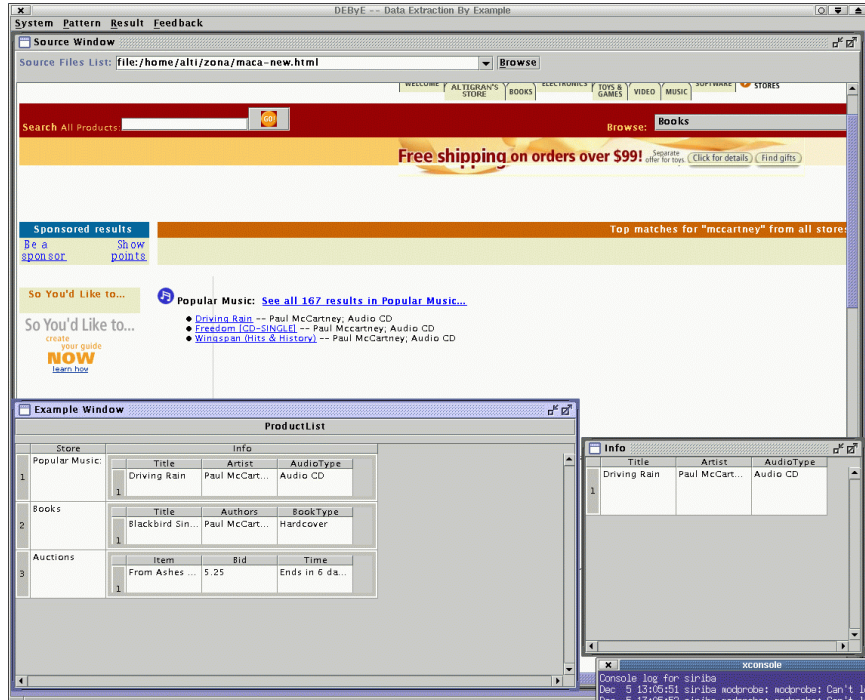


Figure 5: Main screen provided by the DEByE GUI.

limitations are due to the fact that human intervention is required during the generation of the wrappers, since the user has to provide examples during the wrapper creation and maintenance.

To overcome such limitations in the DEByE approach, we devised an Example Generator module that automates the task of generating examples. Thus our solution provides a human-independent bootstrapping method for example-based Web data extraction.

Using the Example Generator, we extended the DEByE approach to generate wrappers that are resilient and adaptive. Consider a wrapper w generated using a set of example objects E taken from a set of sample pages $T_o \subset T$, where T is a page set taken from a Web source S . We define:

- *Resilient Wrapper.* A data extraction process is said to be resilient if its extraction level remains the same for any page set T' , composed of pages that are new versions of the pages in T , taken from the same Web source S . It is assumed that the formatting features and the layout of the pages provided by S have changed and that the content (i.e., the objects of interest) of these pages remains the same.
- *Adaptive Wrapper.* A data extraction process is said to be adaptive if its extraction level remains approximately the same for any page set U , composed of pages taken from a Web source S' , distinct from S , but whose objects belong to the same domain of those present in T .

The framework for automatically generating examples for the wrapper generation process is illustrated in Figure 6. Suppose that it is necessary to generate a wrapper for extracting

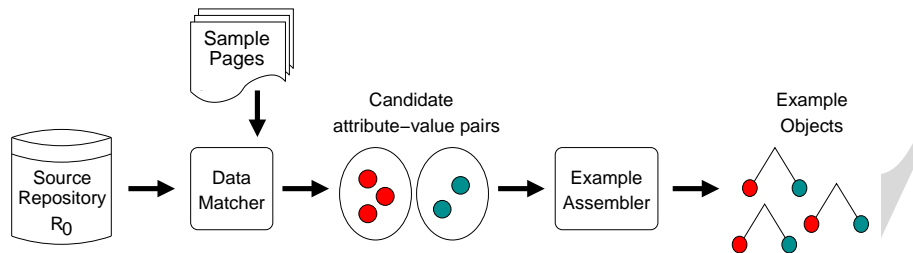


Figure 6: The framework for automatic bootstrapping.

data from a Web source S that contains objects (data) on a given domain (e.g., movies, books, vehicles, etc.). We assume the existence of a bootstrapping data repository R_0 , which we call the *source repository*, that contains a set of objects belonging to the same domain of S . The first step of the bootstrapping process consists in trying to recognize, in sample pages of S , matches for the values of the attributes of the objects in R_0 . This step is performed by the *Data Matcher* module and results in a set of candidate attribute-value pairs taken from the text of the sample pages. Next, the *Example Assembler* module selects those attribute-value pairs that are likely to compose objects in the domain of S (according to a variety of heuristics), and uses them to assemble example objects. We notice that the entire process is guided by the structure of the objects R_0 and that these objects also follow the DEByE data model.

One of the most challenging problems faced was recognizing the intersection among the data in R_0 and the target sample pages. For instance, R_0 might include the string “Paul McCartney” as the value of an artist name, while in a page this name appears as the string “McCartney, P.”. Although these two strings are not the same, they refer to the same person and, thus, should be regarded as part of the intersection between R_0 and the sample pages. To further complicate matters, there is no indication that the string “Paul McCartney” can even be considered as an artist name. For instance, notice that in the page in Figure 1 there is an album entitled “McCartney”.

To deal with this problem, we have devised an attribute matching technique that is capable of properly recognizing the intersection of data between R_0 and the sample pages, generating very few incorrect examples [11]. In the experiments we carried out with 35 Web sources, this technique achieved an average of 97.5% correct examples generated.

Our bootstrapping method is used to provide resilience in the following manner. Suppose that a wrapper was built for extracting data from Netcraft reports², such as the one shown in Figure 7(a). Using the data extracted from these reports as our source repository and a sample page from the new version of the report (shown in Figure 7(b)), our method successfully generates new examples that can be then used for automatically building a wrapper for the new version.

Moreover, using our bootstrapping method we can automatically adapt a wrapper built for a certain Web source for extracting data from other sources of the same domain. For doing so, we start from an existing wrapper and automatically build new wrappers for the other sources by just providing sample pages from them. In this case, the source repository is composed by objects extracted with the existing wrapper. Figure 8 illustrates this procedure

²<http://www.netcraft.com>

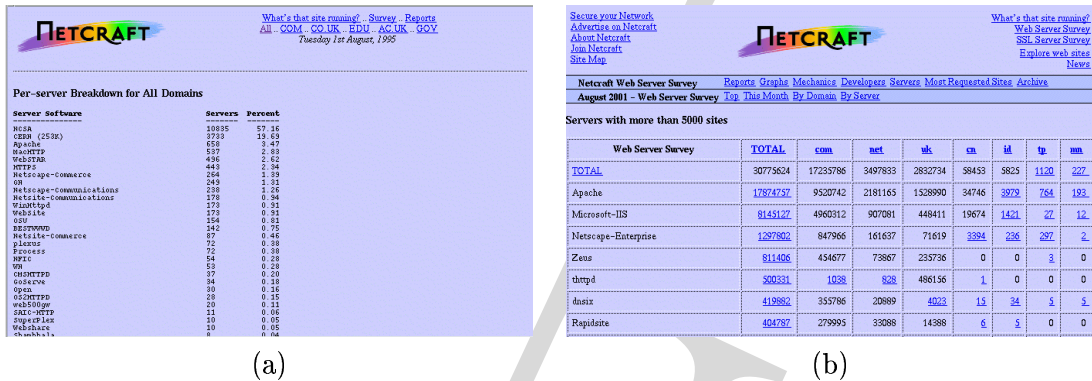


Figure 7: The two versions of the netcraft report.

for the bookstore domain.

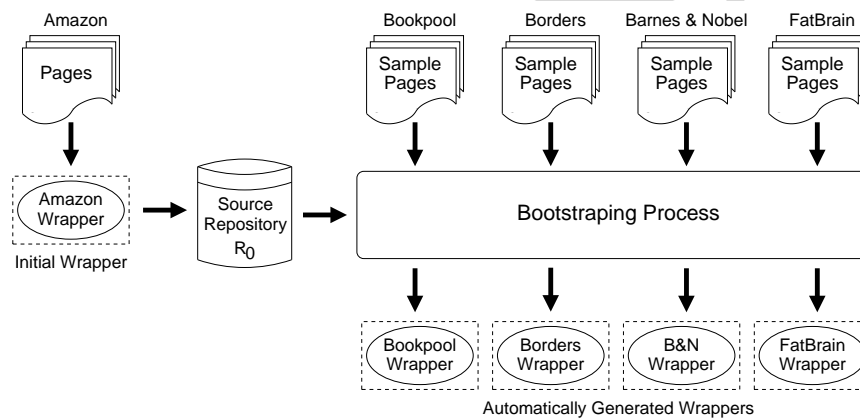


Figure 8: Automatically adapting wrappers for bookstores.

5 Querying Facilities

As we have already discussed (see Figure 3), the process of data extraction within the DE-ByE environment results in a DTOR, a textual data repository organized according to the DEByE data model. In this section, we show how to take advantage of the tabular structure provided by this data model to extend operations of the nested relational algebra for dealing with semistructured Web data. We also present QSBY (Querying Semistructured data By Example) [8], a query interface that implements these operations. QSBY combines features of QBE with typical features of query languages for semistructured data, such as type coercion and path expressions [1], with the advantage of being, in general, much simpler to use than them.

The query operations considered are *selection*, *projection*, *nest*, and *unnest*. These operations can be regarded as extensions to the recursive query operations for regular nested tables proposed by Colby [4]. In what follows, they are briefly described.

The selection operation allows selection conditions to be specified at any level of nesting. A selection condition is a boolean expression defined over the value in a column or over the

existence of a column in an internal table (i.e., a *structural condition*). Selection conditions are evaluated throughout all the nested table, so that tables at any level of nesting can be addressed. The projection operation is similar to the projection operation defined in relational algebra. We can say that this operation horizontally reduces a table by keeping only the columns specified by the user (referred to as the *projected columns*). To deal with the semistructured nature of the Web data, the projection operation recursively verifies the existence of each projected column in all rows of the table. The nest operation has two distinct semantics. When applied to a single column, this operation groups the values in the column that are associated with equal values occurring in other columns. Otherwise, when the nest operation is applied to a set of columns, it creates a new internal table that groups the values of these columns and, consequently, generates a new level of nesting in the table. The nest operation is recursively applied to the entire table. The unnest operation is the inverse of the nest operation. It also has two distinct semantics and must always be applied to a column whose contents is either a list of atoms or an internal table. If it is applied to a list of atoms, it will “ungroup” its elements, i.e., it will split them into different rows of the table. Otherwise, if it is applied to an internal table, it will eliminate a level of nesting. The unnest operation is recursive, similarly to the nest operation.

QSByE implements the four operators just described. Taking advantage of having the structure of the data repository uncovered by the extraction process, the interface provides the user with a *table skeleton* which graphically describes the possible variations for the data in that repository. To illustrate the process of query formulation with QSByE, we use as an example a query over a repository generated from Amazon.com pages like the one in Figure 1.

Suppose a user is interested in issuing the following query: *List the title and the type of the books written by John Grisham. Rearrange the result by nesting the values of the column BookType.* To formulate this query, the user first selects the repository containing the data of interest. This immediately provides the structure of the stored data in the form of a nested skeleton, as illustrated in Figure 9(a). Then, the user specifies the selection conditions on the columns Store and Authors to retrieve only books written by John Grisham. This is shown in Figure 9(a). Notice that the condition on Authors has been specified using the \sim operator that denotes approximate comparison by pattern matching. To show only the title and type of the books, the user specifies a projection operation by clicking on the header of the corresponding columns. This is also shown in Figure 9(a) where the headings of these columns are marked with a “(p)”. Figure 9(b) shows an excerpt of the query result. To finally satisfy the query specification, it is necessary to rearrange the resulting table of Figure 9(b), by nesting the values of the column BookType. This is accomplished as follows. By selecting the “Refine” option from the “Query” menu (Figure 9(a)), the user makes QSByE to replace the current skeleton table by the skeleton of the resulting table of Figure 9(b), as shown in Figure 9(c). Then, the user selects the “Nested” option from the “Table” menu and clicks on the column BookType. QSByE then marks the header of this column with an “(n)” (see Figure 9(c)). The table resulting from this operation is shown in Figure 9(d).

6 Storing DEByE Objects into Relational Tables

In this section, we discuss how semistructured Web data, organized according to the DEByE data model, can be stored into relational tables. The main challenge for storing semistructured Web data in relational databases is the lack of expressiveness of the relational model to

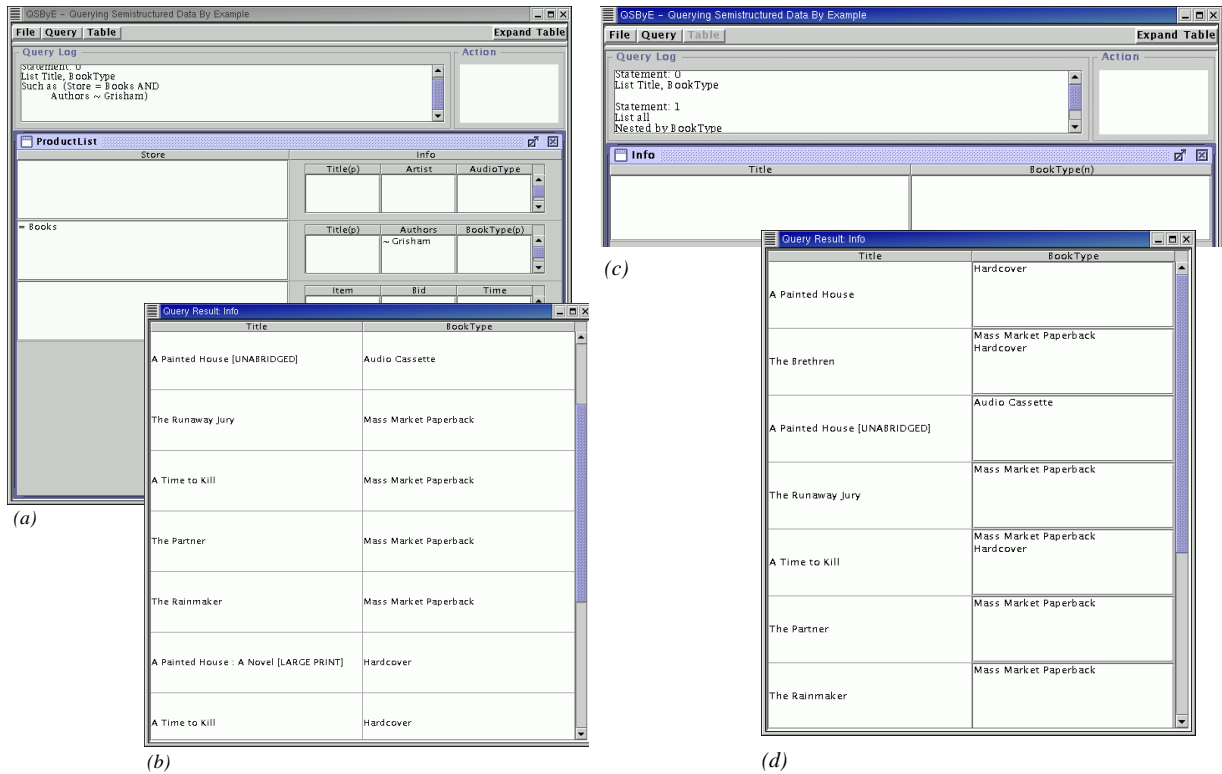


Figure 9: Specification and Result of the Example Query.

represent hierarchical data presenting irregularities. Thus, we developed a mapping method that aims at preserving the multi-level structure of the data using solely the resources provided by the standard relational model.

Our mapping method is described by the procedures **Map-Table** and **Map-Column** presented in Figure 10. In summary, the method consists in traversing the tabular structure of a data repository (a DTOR) given as input and creating a relation for every distinct table scheme found during the traversal. Relations are also created for columns containing lists of atoms. This may imply in splitting the representation of a single object among several relations. The hierarchical relationship between the objects and its sub-objects is enforced by standard referential integrity constraints. Since we cannot rely on the values of the attributes to define keys, we artificially introduce an attribute called *OID* as a key in every relation. This attribute is used to store *object identifiers*, so that references can be consistently made between the objects. We notice that, in both procedures, the value of such identifiers is generated by the function *OID-Gen*. For a more detailed discussion of our mapping method, we refer the reader to [14].

In Figure 11, we illustrate a relational database generated by using our mapping method over the repository corresponding to the nested table of Figure 2. In this figure, relations *Info1*, *Info2* and *Info3* represent the three alternative structures for the contents of the column *Info*. Notice that the three tuples of relation *ProductList* include a null value for *Info*. This is necessary because this relation must be prepared for the possibility of occurring an atomic value in this column.

We notice that, despite the advantages of using relational databases to store semistruc-

```

Map-Table( $N$ :Relation name; $T$ :Repository)
begin
  Let  $T = \{\langle C_1 : v_1^1, \dots, C_m : v_m^1 \rangle, \dots, \langle C_1 : v_1^n, \dots, C_m : v_m^n \rangle\}, (n \geq 0)$ ;
  Create a new relation named  $N$  with scheme  $(OID, C_1, \dots, C_n)$  and key  $OID$ ;
  for  $i = 1, \dots, n$  do
     $o \leftarrow$  OID-Gen;
    Create a new tuple  $t$ ;
     $t[OID] \leftarrow o$ ;
    for  $j = 1, \dots, m$  do
      if  $v_i^j$  is an atom
        then  $t[C_i] \leftarrow v_i^j$ ;
        else  $t[C_i] \leftarrow$  NULL;
        Map-Column( $C_i + j, v_i^j, N, o$ )
      fi
    end
    Insert  $t$  into table  $N$ ;
  end
end

Map-Column( $N$ :Relation Name; $T$ :Object; $P$ :Relation Name; $r$ :oid)
begin
  Let  $T = \{\langle C_1 : v_1^1, \dots, C_m : v_m^1 \rangle, \dots, \langle C_1 : v_1^n, \dots, C_m : v_m^n \rangle\}, (n \geq 0)$ ;
  if there is not a relation named  $N$  then
    Create a new relation named  $N$  with schema  $(OID, REF, C_1, \dots, C_n)$  and key  $OID$ ;
    Create a referential integrity constraint  $N[REF] \subseteq P[OID]$ ;
  fi
  for  $i = 1, \dots, n$  do
     $o \leftarrow$  OID-Gen;
    Create a new tuple  $t$ ;
     $t[OID] \leftarrow o$ ;  $t[REF] \leftarrow r$ ;
    for  $j = 1, \dots, m$  do
      if  $v_i^j$  is an atom
        then  $t[C_i] \leftarrow v_i^j$ ;
        else  $t[C_i] \leftarrow$  NULL;
        Map-Column( $C_i + j, v_i^j, N, o$ )
      fi
    end
    Insert  $t$  into table  $N$ ;
  end
end

```

Figure 10: The mapping Method.

tured Web data [9], our method presents two major drawbacks, as shown by experiments we have performed with the DEByE Data Storage Manager [14]: (1) when compared with the textual repositories, additional disk space is required to store the data in the relational database (mainly due to index structures); and (2) additional time is required to assemble together components of objects that were split among the relations generated, whenever an original repository has to be (total or partially) recovered.

ProductList		
OID	Store	Info
o_1	<i>Popular Music</i>	NULL
o_2	<i>Books</i>	NULL
o_3	<i>Auctions</i>	NULL

Info1				
OID	REF	Title	Artist	AudioType
o_4	o_1	<i>Wingspan (Hits & History)</i>	<i>Paul McCartney</i>	<i>Audio CD</i>
...

Info2				
OID	REF	Title	Authors	BookType
o_7	o_2	<i>Blackbird Singing ...</i>	NULL	<i>HardCover</i>
...

Authors1		
OID	REF	Authors
o_5	o_4	<i>Paul McCartney</i>
o_6	o_4	<i>Adrian Mitchell</i>

Info3				
OID	REF	Item	Bid	Time
o_6	o_3	<i>PAUL MCCARTNEY At ...</i>	<i>26.99</i>	<i>1 days, 20:15:57</i>
...

Figure 11: Example of a mapping.

7 Related Work

In the recent literature, many systems and environments for Web data management have been proposed [10]. These systems have in common the fact that they deploy graph-based formalisms for representing the structure and the contents of Web sites and pages. This has yielded to elegant solutions to the problems of extracting, querying and integrating Web data, but that, unfortunately, require some knowledge on the formalism on the part of the user to be deployed in practice. More recently, systems for directly managing XML data, have also been proposed and implemented.

For the specific task of Web data extraction, several approaches have been recently proposed for the generation of wrappers. As discussed in [13], these approaches are based on a variety of techniques such as natural language processing, machine learning, data modeling, and ontologies. Some other approaches rely on inherent structural features of HTML documents for accomplishing data extraction. Although there are similarities between the DEByE approach and some of the approaches just mentioned, the use of nested tables allowing variations as our underlying data model makes our approach rather distinct in many aspects. In

particular, it allows providing users with a simple and intuitive metaphor for modeling the data they are interested in, and provides flexibility for the extraction process to recognize and extract data presenting nesting levels and structural variations. Experimental results have shown the effectiveness of our approach for Web data extraction [12].

From all of the approaches above, only the approach by Embley et al. [7] generates wrappers that are inherently resilient and adaptive. In the case of RoadRunner [5], the process of wrapper generation is fully automatic and no user intervention is required, so resilience and adaptiveness are not a issue. In this paper we have described a method for extending the DEByE approach to provide resilience and adaptiveness. This method is fully discussed in [11].

The development of query languages and tools for semistructured data is one of the most active research topics nowadays. Initially, languages such as Lorel [1] were proposed aiming at specific data models. Later, dozens of languages targeted to XML began to appear [2], and a standard XML query language, XQuery, is now a W3C Working Draft.

Concerning the storage and manipulation of textual repositories of semistructured data (e.g., XML documents) using relational DBMSs, we can distinguish three basic alternatives: (1) storing the whole repository (or document) within a single relation attribute (an option available in some commercial DBMSs such as Oracle and DB2); (2) treating the objects in the repository as graphs and representing their nodes and arcs by using relations [9]; and (3) mapping the objects in the repository into a corresponding relational database [6]. From these alternatives, only the latter allows to exploit the features of a relational DBMS such as querying optimization, concurrency control, etc. For this reason, we have adopted this alternative in our approach. The problem of mapping arbitrary semistructured objects into relational databases may involve revealing some regular structure that describe them, a problem identified as NP-complete in [6]. However, storing DEByE objects is much simpler, since we take advantage of the previous user modeling to implicitly identify such regular structures.

8 Conclusions

In this paper we have described DEByE, an approach for Web data management in which data obtained from the Web is represented by means of nested tables allowing variations. Based on this approach, we have designed and implemented a set of tools for extracting data from Web sources, querying the extracted data, and storing this data in relational databases. This set of tools compose what we call the DEByE environment for Web data management.

The DEByE approach adopts a data model in which implicit complex objects commonly found in Web pages can be seen as rows of a nested table allowing internal variations. Nested tables are simple, intuitive, flexible, and powerful enough to represent hierarchical objects with structural variations typical of semistructured Web data. It must be noted that this kind of representation for semistructured is not as expressive as general semistructured data models or XML. However, it is expressive enough for representing data available in typical data rich Web pages, as demonstrated by experiments we have performed with the tools we have developed [12]. In fact, we made the choice of trading flexibility in favor of simplicity, since we are not interested in dealing with arbitrary XML documents.

An issue not discussed in this paper is how to automatically obtain the Web pages from where data will be extracted. Although this may be seen as a problem that is orthogonal to

what we have discussed here, we have also designed and implemented a tool for assisting the user in the tasks of generating agents for collecting Web pages containing data of interest, possibly produced as results of form submission (i.e., dynamic pages).

Another major issue on Web data management not addressed in this paper is Web data integration. Although the DEByE data model provides a suitable modeling framework for properly “merging” the objects to be integrated, the problem of establishing identity between objects remains open. This is one of the main research directions we are pursuing at the moment within the DEByE approach.

References

- [1] ABITEBOUL, S., QUASS, D., MCHUGH, J., WIDOM, J., AND WIENER, J. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries* 1, 1 (April 1997), 68–88.
- [2] BONIFATI, A., AND CERI, S. Comparative analysis of five XML query languages. *SIGMOD Record* 29, 1 (2000), 68–79.
- [3] BUNEMAN, P. Semistructured data. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Tucson, AZ, 1997), pp. 117–121.
- [4] COLBY, L. S. A recursive algebra and query optimization for nested relations. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Portland, Oregon, 1989), pp. 273–283.
- [5] CRESCENZI, V., MECCA, G., AND MERIALDO, P. RoadRunner: Towards automatic data extraction from large Web sites. In *Proceedings of the 26th International Conference on Very Large Data Bases* (Rome, Italy, 2001), pp. 109–118.
- [6] DEUTSCH, A., FERNANDEZ, M. F., AND SUCIU, D. Storing semistructured data with STORED. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Philadelphia, Pennsylvania, 1999), pp. 431–442.
- [7] EMBLEY, D. W., CAMPBELL, D. M., JIANG, Y. S., LIDDLE, S. W., KAI NG, Y., QUASS, D., AND SMITH, R. D. Conceptual-model-based data extraction from multiple-record Web pages. *Data and Knowledge Engineering* 31, 3 (1999), 227–251.
- [8] EVANGELISTA-FILHA, I. M. R., LAENDER, A. H. F., AND SILVA, A. S. Querying semistructured data by example: The QSByE interface. In *Proceedings of the International Workshop on Information Integration on the Web* (Rio de Janeiro, Brazil, 2001), pp. 156–163.
- [9] FLORESCU, D., AND KOSSMAN, D. Storing and querying XML data using a RDBMS. *IEEE Data Engineering Bulletin* 2, 3 (1999), 10–17.
- [10] FLORESCU, D., LEVY, A. Y., AND MENDELZON, A. O. Database techniques for the World-Wide Web: A survey. *SIGMOD Record* 27, 3 (1998), 59–74.

- [11] GOLGHER, P. B., DA SILVA, A. S., LAENDER, A. H. F., AND RIBEIRO-NETO, B. A. Bootstrapping for Example-Based Data Extraction. In *Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management* (Atlanta, GA, 2001), pp. 371–378.
- [12] LAENDER, A. H. F., RIBEIRO-NETO, B., AND DA SILVA., A. S. DEByE – Data Extraction by Example. *Data and Knowledge Engineering* 40, 2 (2002), 121–154.
- [13] LAENDER, A. H. F., RIBEIRO-NETO, B., DA SILVA, A. S., AND TEIXEIRA., J. S. A Brief Survey of Web Data Extraction Tools. *SIGMOD Record* 2, 31 (2002).
- [14] MAGALHÃES, K. V., LAENDER, A. H. F., AND DA SILVA, A. S. Storing semistructured data in relational databases. In *Proceedings of the 8th Symposium on String Processing and Information Retrieval* (Laguna de San Rafael, Chile, 2001), pp. 143–152.