

DEByE – Data Extraction By Example

Alberto H. F. Laender Berthier Ribeiro-Neto

Altigran S. da Silva

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

Abstract

In this paper we present *DEByE (Data Extraction By Example)*, an approach to extracting data from Web sources, based on a small set of examples specified by the user. The novelty is in the fact that the user specifies examples according to a structure of his liking and that this structure is described at example specification time. For the specification of the examples, the user interacts with a tool we developed which adopts nested tables as its visual paradigm. Nested tables are simple, intuitive, and allow shielding the user from technical details (such as HTML tags, formatting operators, and learning automata) related to the extraction problem. The examples provided by the user are then used to generate patterns which allow extracting data from new documents. For the extraction, DEByE adopts a new bottom-up procedure we proposed which is very effective with various Web sources, as demonstrated by our experiments.

Keywords: data extraction, wrapper generation, web data management

1 Introduction

The spreading of modern digital libraries and the popularization of the Web have made a huge volume of textual information available to a very large audience. In fact, just the amount of Web data in text format is estimated to be in the order of one terabyte [8]. Additionally, a large and ever increasing audience has demonstrated great interest in accessing information. This leads to a great necessity of new and efficient tools to manage, retrieve, and filter information in the Web.

The need for new Web data management tools is further stressed by the fact that finding the desired information in the large text databases present in the Web is not a trivial task. In fact, it is well known that many Web users find it difficult (and frequently impossible) to locate the information of their interest. One main reason is that users might be interested in semistructured data (which exists in many Web pages) that is not recognized by traditional Web interfaces and search engines. New tools for solving this problem are in great demand.

It is worth noticing that the spreading of XML (as a standard format for Web data publishing and interchanging) does not provide a trivial solution to the problem of retrieving existing semistructured Web data. The reasons are multi-fold: (1) the volume of HTML pages currently available is yet enormous and it is increasing; (2) XML is meant to be deployed in business-to-business scenarios, which suggest that most users will continue to access data in HTML format; (3) since it is necessary to translate from HTML to XML, new tools for managing semistructured data can greatly simplify the task; and (4) in many cases, this new technology can also be used for dealing with non-HTML data sources (e-mail, business reports, scientific documents, program output, etc.). Thus, new solutions to the problem of retrieving (or extracting) semistructured Web data continue to be of great interest.

In this work in particular, we are mostly interested in studying the problem of how to gain access to semistructured data which is present in Web pages but is not readily available. To appreciate the relevance of this problem, consider a Web site of a bookstore such as “Murder by the Book” (a bookstore specialized in mystery books) [39]. Figure 1 presents a snapshot of a page of this site, which also includes data on thousands of other books. For each book, the site exhibits data on authors, titles, publication, year, and price. These data are *attributes* or features of a book and are readily recognizable when we glance through the Web page. Such features define an

implicit structure associated with each book listed. This structure is said to be *implicit* because it has not been declared explicitly as done when we specify the schema of a database. Further, since this structure might vary from one book to another (for instance, information on the number of pages might be available for one book but not for another), we say that the actual data is *semistructured* [2, 11, 24].

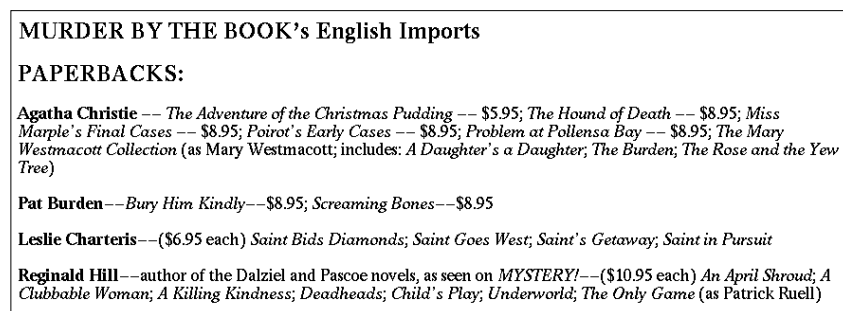


Figure 1: Excerpt of a page from the Murder by the Book Web site.

Consider a user who is interested in mystery books which cost between US\$5.00 and US\$7.00. Since the data on book prices might appear mixed in the text, this user might need to browse through the pages of the site to find books within his cost constraints. If the number of pages is large, this browsing task becomes tedious and time consuming. To avoid this problem, we could extract the data from the Web pages automatically and store them in a data repository (e.g., a database) for later querying. This would allow easily retrieving books with a price between US\$5.00 and US\$7.00.

The automatic recognition and extraction of semistructured data constitutes a current and important research topic concerning the Web [4, 6, 19, 20, 22, 23, 26, 30, 33, 36, 43, 46, 47]. More formally, the problem of extracting data from Web sources can be stated as follows. Given a Web page S containing a set of implicit objects, determine a mapping W that populates a data repository R with the objects in S . The mapping W must also be capable of recognizing and extracting data from any other page *similar* to S . We use the term *similar* in a very empirical sense, meaning pages provided by a same site or Web service, such as pages of a same Web bookstore. In the recent literature, any software that executes the mapping W has been termed a *wrapper*.

An important point concerning the generation of such a mapping is defining what an object is. For instance, in the page excerpt of Figure 1, one may be interested in extracting only names of authors, while others may be interested in titles and prices of books. Further, a third user may be interested in extracting entire author’s entries present in the page, that is, names along with lists of the titles of their books, and the corresponding prices. Notice that, in this last case, we are dealing with complex objects which present a nested structure. Further, we must expect situations in which distinct objects associated with the same semantics present distinct structures. That is, structural variations such as the absence of an attribute or out of order attributes are expected to occur. In Figure 1, for instance, the entries corresponding to the authors “Pat Burden” and “Leslie Charteris” present noticeable structural variations.

In this paper, we present a new approach for generating wrappers that implement the mapping W , which we call *DEByE (Data Extraction By Example)*¹. The motivation is to let the user or database designer specify a target structure for the data to be extracted. Contrary to most approaches in the literature, DEByE does not require the user (or database designer) to describe the inherent structure of a whole Web page. Instead, the user pick pieces of data of interest from the source page and insert these data into a structure of his liking. The structure the user devises might be rather distinct from the inherent structure of the whole Web page. This approach has some advantages: (a) the user is shielded from many of the details required to describe the inherent structure of a whole Web page, (b) the user can map the data into a new structure of his preference even before the data has been extracted, (c) the step of providing examples for the extraction procedure is simpler and more intuitive than other approaches described in the literature. Our approach innovates by combining an intuitive paradigm for the specification of the target data with a suitable strategy for recognizing and extracting objects with nesting levels and structural variations, features typical of data found in Web pages.

The idea of using examples for generating wrappers is not new and has been applied in recent works in the area of machine learning [30, 33, 43]. However, these approaches rely on the knowledge of the structure of the page containing the data to be extracted. To illustrate, in the work described in [33] six distinct kinds of page structure are identified and specific classes of wrappers are

¹This name is an homage to Moshé Zloof, creator of QBE [52], who suggested the paradigm we use to specify the data to be extracted from Web pages.

proposed do deal with each one of them. However, only the generation of wrappers that deal with pages containing flat tuples with a fixed structure are actually addressed. In [30] a generalization of this work is proposed that can deal with possible variations in the tuples (e.g., changes in the order of the attributes or missing attributes), but the limitation of dealing with flat structures remains. The strategy proposed in [43] allows the extraction of nested objects, but it also relies on a previous description of the structure of the target page, which is separately encoded. In DEByE, each example provided by the user suggests a possible structure for the objects being extracted (which, frequently, constitutes a small portion of the structure of the whole page). This adds flexibility to the extraction process and provides a natural and efficient way of handling nested objects and structural variations.

Based on our data extraction approach, we implemented a tool that is used in our experiments. To allow a convenient specification of examples, the DEByE tool represents the structure of the data through nested tables [37, 48]. Nested tables are interesting because they are simple, intuitive, and are expressive enough to represent the semistructured data normally present in common Web pages [34]. The examples provided by the user are used to generate extraction patterns which allow extracting new data from new Web pages. For this, we use techniques from information retrieval to build an extraction strategy which is very effective with various Web sources, as demonstrated through experimentation.

In this work, we describe the DEByE extraction process as a whole, since the interaction of the user with the GUI until the instantiation of the extracted objects. We run a thorough round of experimentation (which includes 15 distinct Web sources) and provide a complete analysis of our results. A comparison with other approaches in the literature is also included. This complements the work we did in [34, 46, 47].

The paper is organized as follows. In Section 2 we introduce the DEByE approach. Section 3 discusses our strategy for recognizing and extracting semistructured data in new Web pages. In Section 4, the main features of the DEByE tool are discussed. Section 5 shows the results of the experiments we have performed using the tool. In Section 6 we discuss related works and compare our approach with them. Finally, Section 7 presents our conclusions.

2 The DEByE Approach

For illustrative purposes, we consider the excerpt of a page from the Murder by the Book Bookstore [39] Web site shown in Figure 1.

At first, we notice that there is an inherent structure to the text on this page. In fact, we are able to identify four distinct pieces of data which refer to four authors and some of their books. We consider each one of these pieces of data as a distinct *object* that appears implicitly in the text. For each of these objects, we are able to distinguish an author name and a corresponding list of books. Further, for the books in a list, we can identify book titles and prices. As it can be noticed, there is an inherent and implicit structure associated with the objects in this Web page which can be perceived by a user either entirely or at least partially. Such structure has not been declared anywhere but is clearly identifiable. Texts or pages which present such a type of inherent structure and whose data is restricted to a specific domain are said to be *data rich* and *narrow in ontological breadth* [21, 22, 23]. Such pages constitute the target of our study and are referred to simply as data rich pages.

The implicit objects in Figure 1 have a multi-level structure and, because of that, are said to be *complex objects*. To illustrate, Figure 2 displays a hierarchical structure for the object corresponding to data on the books by *Agatha Christie*. As we can see, the object Author is composed of two other objects that are called component objects or simply *components* [34]. The first component (Name) is an attribute (or *atom*) object. The second component (Book) is a list object, or simply a *list*. This list is composed of several identical tuple objects, or simply *tuples*, each one composed of two attributes (Title and Price)

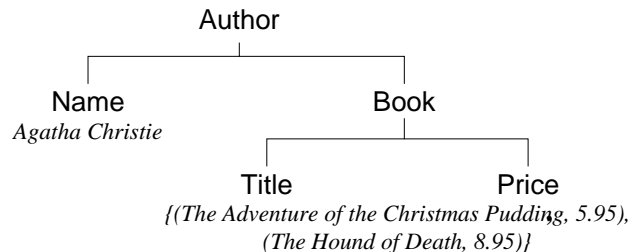


Figure 2: Hierarchical structure for one of the implicit objects in Figure 1.

Because the structure of the implicit objects (i.e., the focus is on the objects of interest to the user, not on the structure of the page) might not be easily recognizable at first, some type

of procedure for facilitating the task of specifying this structure should be provided. In the DEByE approach, this is done indirectly through the specification of an *example object*. This is accomplished by cutting pieces of data from a sample page and inserting these pieces into a nested table. Nested tables are adopted as the DEByE interface paradigm because they are intuitive and simple to specify. For instance, Figure 3 illustrates a nested table (built using the graphical interface described in Section 4) that represents the complex object in Figure 2 (only one book title is shown). While assembling the example, the user concentrates on arranging the nested table such that it reflects his interpretation of the data. Nested tables provide an intuitive metaphor that can be effectively used by various people for properly specifying their interpretation of the structural information implicit on a data rich Web page. Furthermore, as demonstrated by our experiments [34], such metaphor is so convenient that it very frequently leads to the same representation of the implicit structural information whenever the users have the same view (or interpretation) of the data.

It is worth noting that representing semistructured data using formats such as XML [9] or OEM [45] provides much more expressiveness and flexibility than using nested tables. Indeed, intuitively, every nested table can be represented in terms of complex objects, although the contrary is not necessarily true. Thus, nested tables are not as powerful as XML or OEM. For this reason, in our work we extend the usual notion of nested tables to allow representing variations in the structure of the data (details can be found in [34]).

Recognizing the structure of the data solves a part of the problem. Another critically important part is the issue of how to automatically extract new data from new Web pages to populate a given nested table. For this, we again take advantage of information provided indirectly by the user while specifying an example object, as follows. To assemble an example, the user must first mark and cut pieces of data from a sample Web page. Each time he marks a piece of data, he also indicates implicitly a *syntactic context* in which that piece of data appears in the page. Using techniques from the area of information retrieval, we demonstrate that such context can be characterized without human intervention and then used effectively to guide an automatic data extraction procedure. Because of this characteristic, there is no need for encoding either implicitly in the extraction procedure or separately information on the structure of the target pages as required by other approaches in the literature [4, 30, 33, 36, 43].

The combination of nested tables with the automatic identification of a syntactic context associated with the data to extract provides high interactivity. By high interactivity, we mean that the user or database designer can specify example objects, examine the results of the extraction process, and specify new example objects if necessary in a matter of minutes. This interactive cycle can be done quickly because the user does not have to be concerned with the whole structure of the text source. Most important, this degree of interactivity shields the user from details of the extraction process and allows him to focus on the task of specifying examples that reflect his interpretation of the data.

As we shall see later, a couple of examples is usually sufficient to describe the inherent structure of the objects on a typical data rich Web page and to generate extraction patterns required to process hundreds of new (but similar in structure) pages. Most important, the whole approach is intuitive and can be easily grasped by a user, as demonstrated by the experiments we carried out [34].

Author			
	Name	Book	
1	Agatha Ch...	Title	Price
		1	The Adve... 5.95
2	a	2	3
		1	a a

Figure 3: A nested table corresponding to the hierarchical structure in Figure 2.

Figure 5 presents an overview of the whole DEByE approach. The two modules called *Graphical User Interface (GUI)* and *Extractor* compose what we call the DEByE tool. These two modules communicate through an intermediary data structure called *Object Extraction Patterns (oe-patterns)*. The *GUI* module provides the user with a Java interface that he uses to assemble the example objects. To assist the user with this task, the interface displays sample pages from the target Web source and allows the user to cut and paste data from these sample pages. The assembled objects are then used to generate *oe-patterns* for extracting new objects. The *Extractor* module takes these patterns and applies them to new pages from the target Web source. This

matching strategy allows identifying new objects which are then extracted and provided as output of the DEByE tool. The set of *Extracted Objects* is coded in an XML-based format which can be easily converted to other format or inserted into (nested) tables for later querying. To illustrate, Figure 4 shows an object described according to such a format. The attributes *ipos* and *fpos* indicate the initial and the final positions for the occurrence of the piece of data in the source text.

```

<TUPLÉ type="Author">
  <ATOM type="Name">
    <VALUE ipos="2058" fpos="2073">Agatha Christie</VALUE>
  </ATOM>
  <LIST type="Book">
    <TUPLÉ type="Book">
      <ATOM type="Title">
        <VALUE ipos="2084" fpos="2122">The Adventure of the Christmas Pudding</VALUE>
      </ATOM>
      <ATOM type="Price">
        <VALUE ipos="2131" fpos="2135">5.95</VALUE>
      </ATOM>
    </TUPLÉ>
    <TUPLÉ type="Book">
      <ATOM type="Title">
        <VALUE ipos="2257" fpos="2280">The Hound of Death</VALUE>
      </ATOM>
      <ATOM type="Price">
        <VALUE ipos="2289" fpos="2293">8.95</VALUE>
      </ATOM>
    </TUPLÉ>
  </LIST>
</TUPLÉ>

```

Figure 4: Example of an extracted object described according to our XML-based format.

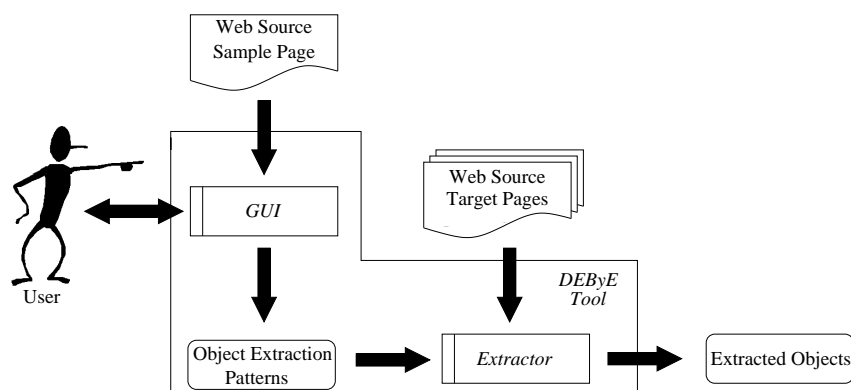


Figure 5: Modules of the DEByE tool and their role in the data extraction process.

3 Data Extraction

In DEByE, the examples of objects specified by the user are used to generate *Object Extraction Patterns (oe-patterns)*. These oe-patterns combine structural and textual information that are used to recognize and extract new objects.

In this section, we discuss the generation of oe-patterns and present two extraction strategies called *top-down* and *bottom-up*. The extraction approaches we propose are entirely based on the assumption that the user is able to provide examples of objects to extract. The discussion on how the user can specify such example objects is postponed until Section 4. We begin by introducing some necessary notation and terminology.

3.1 Notation and Terminology

For representing complex objects we use the notion of an *object type*. An object of a specific object type is called an *instance* of that type. Definition 1 presents the syntax and the semantics of object types we adopt in this paper.

Definition 1 We define an **object** as a pair $o = \langle \tau, v \rangle$, such that $v \in \text{dom}(\tau)$, where τ is the **type** of o , v is the **value** of o , and $\text{dom}(\tau)$ is the **domain** of τ . Further, o is said to be an **instance** of τ . Types are defined as follows:

- If $\text{dom}(\tau) = \{a_1, a_2, a_3, \dots, a_n\}$ ($n \geq 1$) is a set of atomic values, then τ is an **atomic type** or **a-type**. Instances of a-types are termed **atoms**. The pair $\langle \tau, a_i \rangle$ is referred to as an Attribute-Value Pair (AVP).
- If $\tau_1, \tau_2, \dots, \tau_n$ ($n \geq 2$) are types, then $\tau = (\tau_1, \tau_2, \dots, \tau_n)$ is a **tuple type** or **t-type**. In this case, $\text{dom}(\tau) = \text{dom}(\tau_1) \times \text{dom}(\tau_2) \times \dots \times \text{dom}(\tau_n)$. Instances of t-types are termed **tuples**.
- If τ_0 is a type, then $\{\tau_0\}$ is a **list type** or **l-type**. In this case, $\text{dom}(\{\tau_0\}) = \{i \rightarrow \text{dom}(\tau_0) \mid i \in \{1, \dots, |\text{dom}(\tau_0)|\}\}$. Instances of l-types are termed **lists**.
- If $\tau_1, \tau_2, \dots, \tau_n$ ($n \geq 2$) are types, then $\tau = [\tau_1, \tau_2, \dots, \tau_n]$ is a **variant type** or **v-type**. In this case, $\text{dom}(\tau) = \text{dom}(\tau_1) \cup \text{dom}(\tau_2) \cup \dots \cup \text{dom}(\tau_n)$. Instances of v-types are termed **variants**.

Informally, a-types, t-types and l-types capture the usual notions of atoms, tuples and lists, respectively. To capture the semistructured nature of typical Web data, we adopt the notion of a v-type [35]. Instances of a v-type are objects of any type from a list of types called the *alternatives* of the variant type.

3.2 Object Extraction Patterns

Object Extraction Patterns (oe-patterns) are patterns used by the *Extractor* module as illustrated in Figure 5. The oe-patterns describe the hierarchical structure of the example objects (specified by the user). A typical hierarchical structure is shown in Figure 2. The nodes at the bottom of the hierarchy are used to match AVPs and are called *Attribute-Value Pair patterns (avp-patterns)*. Before discussing oe-patterns, we first define avp-patterns.

Attribute-Value Pair Patterns

Definition 2 An Attribute-Value Pair Pattern (avp-pattern) is a pair $\langle \tau, \rho \rangle$, where τ is an atomic type (i.e., an a-type) and ρ is a text pattern. The text pattern ρ is used to match string values, in the domain of the a-type τ , as they occur in a given text (i.e., in an HTML page).

Each string a_i that matches ρ is used to compose an AVP described as $\langle \tau, a_i \rangle$. To each such AVP we associate a *local syntactic context* that can be derived from the strings surrounding the AVP value in the text. For this, we use the concept of a *passage* (or *window*) and techniques from information retrieval [8, 15, 31], as follows. The tokens surrounding the AVP value constitute a passage that can be used as its *local context*. For instance, Figure 7 illustrates the value 10.95 of type Price (which occurs in a page of the Murder by the Book Web site as illustrated in Figure 6) and a passage that can be used as its local context. Using this context information, we build an avp-pattern which can be later used to identify other values for the type Price. Figure 8a illustrates a possible representation of this avp-pattern. We refer to this avp-pattern as $\langle \tau, s_{pre} * s_{suf} \rangle$, where the symbol $*$ is used to match any sequence of characters (representing a value selected by the user to assemble an example), the symbol s_{pre} refers to a string that is a prefix for $*$, and the symbol s_{suf} refers to a string that is a suffix for $*$.

The avp-pattern in Figure 8a is too specific and will most likely not match other values expected for Price. The key problem is that this pattern includes too much information about the local

```

Malcolm Hamer--GOLF MYSTERIES!--
A Deadly Lie--$10.95; Dead on Line--$13.95;
Shadows on the Green--$10.95; Sudden Death--$9.95;

```

Figure 6: A sample text.

value

```

A Deadly Lie--$ 10.95 ; Dead on Line

```

Figure 7: Example of an AVP context.

context in which the value 10.95 appeared. Thus, to be able to effectively use this avp-pattern for recognizing and extracting new values for the type Price in other Web pages, we must transform this pattern into a new pattern that is more general in the sense that it contains less contextual information. Figure 8b shows a variation of the avp-pattern for the value 10.95, generated by reducing the length of the prefix s_{pre} and the length of suffix s_{suf} . This new avp-pattern can now be used to effectively match other values for Price in this page. In general, the generation of avp-patterns can be accomplished as follows.

Given an AVP selected by the user, we determine a passage surrounding this AVP value in the text. Initially, we adopt symmetric passages composed of W text tokens to the right and W text tokens to the left of the AVP value. Due to heuristics applied to token identification (for instance, a run of spaces is considered a single token), an avp-pattern might result asymmetric as illustrated in Figure 8b. We start with a small pattern composed solely of the symbol *, of a token to its right and of a token to its left. These tokens can be character strings or special symbols.

s_{pre} s_{suf}

```

A Deadly Lie--$ * ; Dead on Line

```

(a)

s_{pre} s_{suf}

```

--$ * ;

```

(b)

Figure 8: Examples of avp-patterns.

For instance, in Figure 8b the initial avp-pattern would be composed of the symbol * surrounded by the prefix "--\$" and the suffix ";". Notice that the prefix and the suffix are very useful, are present in the page, and can be recognized automatically once the user marks 10.95 as a value of his interest. We then parse the sample page (which is displayed on the user screen) looking for matches to the avp-pattern just defined, and count the number of matches. This count is compared with an estimate, provided by the user, for the number of price values in the example page (see Section 4). If the number of matches counted exceeds the number of price values provided by the user, we add additional terms to the pattern, increasing its width W and the amount of contextual information attached to it. This process is repeated automatically until we have a good definition for the local context of the AVP in consideration (we stop when the number of matches is smaller than the number of price values identified by the user). Notice that all the user has to provide is a single number that indicates the number of instances (values) of Price he sees in the sample page. This number does not need to be the exact number of occurrences of Price. In most cases, a rough approximation of this number is sufficient to adjust the width W of the avp-pattern. Such information is simple to provide and presents little inconvenience to the user.

The resulting avp-patterns are then used to compose an *Object Extraction Pattern (oe-pattern)*, as we discuss in the immediately following.

Oe-patterns

Oe-patterns are essentially trees containing information on the structure of the objects and on their associated AVPs. The sub-trees of an oe-pattern are themselves oe-patterns, modeling the structure of component objects. At the bottom of the hierarchy lie the avp-patterns, used to identify atomic components (AVPs).

Definition 3 An oe-pattern o_e is a tree whose internal nodes are of the form $\langle \tau, o \rangle$, where τ is the type of the node and o is another oe-pattern (i.e., a "sub" oe-pattern). The leaves of an oe-pattern are avp-patterns, as previously defined. The function $leaves(o_e)$ returns the set of nodes in the leaves of o_e , while the function $root(o_e)$ returns the root node of o_e . Let $n = \langle \tau, o \rangle$ be a node of the oe-pattern o_e . We define $type(n) = \tau$ as a function that returns the type of n . Further, if n is an internal node, $children(n)$ gives the set of child nodes of n .

For pages with non-homogeneous objects (such as the pages of the Murder by the Book site in Figure 1), the user has to specify more than one example object. For instance, in Figure 1, the user has to specify a second example object, besides the example object illustrated in Figure 2, to indicate that, in the case of the author *Leslie Charteris*, the price of all her books is the same. In this case, a distinct oe-pattern is generated for each of the two example objects provided by the user.

3.3 Extraction Strategies

In this section, we discuss two distinct strategies for extracting new objects from new Web pages, given an oe-pattern. These strategies are labeled *top-down* and *bottom-up*. The top-down strategy was first presented in [47], while a first version of the bottom-up strategy was presented in [46].

Top-down Extraction Strategy

Given a small set of example objects, a simple strategy for extracting new objects from new pages consists in: (a) assembling an oe-pattern for each example object and (b) using these oe-patterns to directly recognize new objects. This can be accomplished roughly by the algorithm in Figure 9 (consider, for a moment, that the examples provided by the user are flat, i.e., they have a single level hierarchical structure) which is described here in high level. This algorithm assembles an

```

TD-Extraction
begin
  Let  $G$  be a set of Web pages given as input;
  combine all the avp-patterns to form an oe-pattern for whole objects;
  foreach page  $g$  in  $G$  do
    use the oe-pattern to recognize and extract new objects;
  end
  output the newly extracted objects;
end

```

Figure 9: The top-down algorithm.

oe-pattern and uses this pattern to identify new objects in new pages. This procedure is repeated if more than one example object is provided (in this case, more than one oe-pattern is assembled). We say that the algorithm is *top-down* because it recognizes whole objects which are then broken

down into their components.

Despite its simplicity, this top-down extraction strategy works well with pages that are well structured (i.e., that are data rich and present none or little variation in their structure). For pages with variable structure (which are quite common in the Web), a distinct *bottom-up* extraction strategy is more appropriate, as we now discuss.

Bottom-up Extraction Strategy

The main feature of our bottom-up extraction strategy is that it recognizes and extracts atomic components (i.e, AVPs that lie at the bottom of the hierarchical structure of a complex object), prior to the recognition of the object itself. The extracted AVPs are then used to assemble the object through a bottom-up composition operation. Before discussing the bottom-up algorithm, we introduce some necessary notation.

Definition 4 *We extend our early notation and represent objects by a triple $o = \langle \tau, v, l \rangle$, where τ is the type of the object, v is its value, and the parameter l is called the index of the object. The index provides the relative location of the object within the source page from where it originates. For any given object $o = \langle \tau, v, l \rangle$, we define the function *type* as returning the type of o , that is, $\text{type}(o) = \tau$. Similarly, we define the indexing function ℓ as returning its index, that is, $\ell(o) = l$.*

Another important notion for our purposes is the sequencing of objects in the source page. This leads to the definition of a *sequence*, as follows.

Definition 5 *Let O be a set of objects. We define a sequence in O as any ordered subset $S = \langle o_1, o_2, \dots, o_n \rangle$ ($n > 1$) of O such that: (1) $i < j$ iff $\ell(o_i) < \ell(o_j)$ and (2) there is no $o \in O$ such that $\ell(o_i) < \ell(o) < \ell(o_{i+1})$ ($i < n$).*

In other words, sequences are simple occurrences of consecutive objects in O .

Our bottom-up extraction strategy is considerably more complex than the top-down strategy, as illustrated in Figure 10. The algorithm takes as input an oe-pattern o_e and a Web page g . Instead of using the oe-pattern o_e for recognizing whole objects, the algorithm uses the avp-patterns of o_e (i.e., the leaves of o_e) to match strings in the source page g . For each avp-pattern $\langle \tau, \rho \rangle$ the algorithm obtains all strings within the current page g that match ρ . Each matching string s is used to compose an atomic object $\langle \tau, s, l \rangle$, where the index l is the position of the

Bottom-Up Extraction

Input

A Web page g ;
An oe-pattern o_e ;

Let R be a global variable that stores the set of objects being assembled;

begin

Extraction Phase

foreach avp-pattern $\langle \tau, \rho \rangle \in \text{leaves}(o_e)$, **do**
 foreach string s in page g that matches ρ **do**
 Let l be the location of the string s in page g ;
 $R \leftarrow R \cup \{\langle \tau, s, l \rangle\}$; (S₁)
 end
end

Assembling Phase

Assemble-Objects(o_e);

end

Assemble-Objects

Input

oe-pattern o_e ;

begin

Let $r = \text{root}(o_e)$

foreach $o_c \in \text{children}(r)$
 if $\text{type}(o_c)$ is not an a-type (S₂)
 Assemble-Objects(o_c);
 end
end

Tuple Assembling

if $\text{type}(r)$ is a t-type of the form $(\tau_1, \tau_2, \dots, \tau_m)$, $m \geq 2$ (S₃)
 foreach sequence $S = \langle o_1, o_2, \dots, o_n \rangle$ ($0 < n \leq m$) in R such that,
 for every $o_i, o_j \in S$,
 - $\{\text{type}(o_i), \text{type}(o_j)\} \subseteq \{\tau_1, \tau_2, \dots, \tau_m\}$ and
 - $\text{type}(o_i) \neq \text{type}(o_j)$

do
 $R \leftarrow R - S$;
 $R \leftarrow R \cup \{\langle \text{type}(r), S, \ell(o_1) \rangle\}$; (S₄)

end

end

List Assembling

if $\text{type}(r)$ is a l-type of the form $\{\tau_0\}$ (S₅)
 foreach sequence $S = \langle o_1, o_2, \dots, o_n \rangle$ ($n > 0$) in R such that,
 for every $o \in S$, $\text{type}(o) = \tau_0$;

do
 $R \leftarrow R - S$;
 $R \leftarrow R \cup \{\langle \text{type}(r), S, \ell(o_1) \rangle\}$; (S₆)

end

end

end

Figure 10: The bottom-up algorithm.

string s in the page g . We call this step the **Extraction Phase** of the algorithm. At the end of the **Extraction Phase**, a set R of triples corresponding to the extracted AVPs is obtained. These triples are then used to compose new objects in the **Assembling Phase** of the algorithm.

The core of the **Assembling Phase** is the recursive procedure **Assemble-Objects**. It receives an oe-pattern o_e and operates on the global repository R (which, at the beginning of the **Assembling Phase** contains only atomic objects). For each non-atomic component o_c of o_e (i.e., for each $o_c \notin \text{leaves}(o_e)$), a recursive call is made until the leaves of o_e (i.e., its avp-patterns) are reached (step S_2). From there, the procedure **Assemble-Objects** operates bottom-up.

The innermost component for which a call to **Assemble-Objects** was made is either a t-type or an l-type. If it is a t-type, sequences of objects in R that match the components of the t-type are identified (step S_3), removed from R , and reinserted as a tuple (step S_4). If instead it is an l-type, sequences of objects in R that match the members of this l-type are identified (step S_5), removed from R , and reinserted as a list (step S_6).

The structure of the whole object being composed is implicitly stored in the recursion stack. As we finish a recursive call, we move up in the structure of the object being composed. At the upper level, the complex component to assemble next is again either a t-type (in which case steps S_3 and S_4 are executed) or an l-type (in which case steps S_5 and S_6 are executed). The procedure operates in depth first traversal mode.

When a tuple is being assembled (step S_3), sequences must be composed of adjacent objects of distinct types. As soon as a repeated type occurs, the assembling of the current tuple ends and the assembling of a new tuple begins. The assembling of a tuple can also finish with the occurrence of an object whose type is distinct from the types expected for that tuple. The objects in a sequence can appear in any order. Further, some of the types expected for composing the tuple might be missing. When a list is being composed (step S_5), all objects in the sequence must be of the same type. The assembling of the list stops when an object of a type distinct from the expected type is found.

We now provide an example of how the bottom-up strategy works with objects that have a non-flat structure. The assembling steps for this example are illustrated in Figure 11. In this figure, circles represent objects extracted or being assembled. Consider the instances of type **Author** whose structure is illustrated in Figure 2. Assembling such instances requires three assembling

steps. In the first step, **Book** instances are assembled from AVPs of types **Title** and **Price** (obtained in the extraction phase). In the second step, **Book** instances related to a same instance of **Author** are collected together in a list (referred to as **{Book}**). In the third step, each one of those lists are combined with an instance of **Name** (previously extracted), to assemble **Author** instances. Notice that this order corresponds to a bottom-up traversal of the hierarchical structure of **Book** instances.

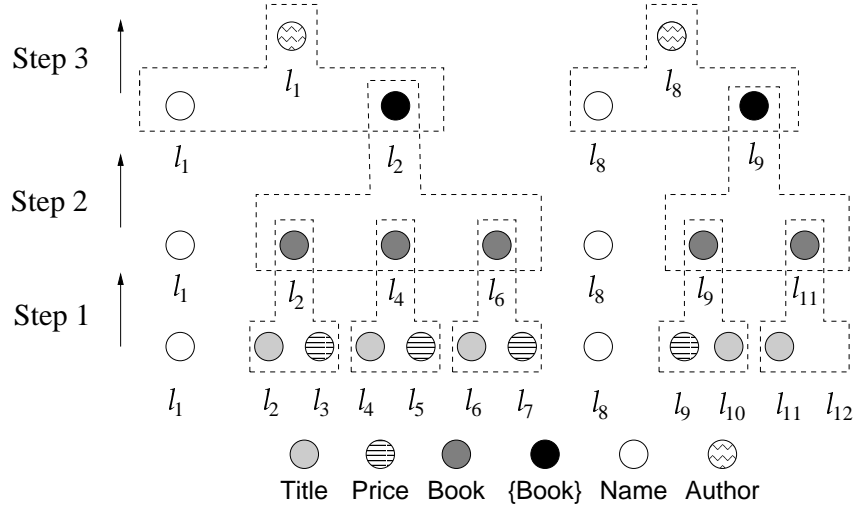


Figure 11: Execution of the bottom-up algorithm.

The lowest row of circles represents a set containing only AVPs. A label l_i is associated with each one of these AVPs. This label indicates the position in the source page of the string associated with the AVP. Also, if $j > i$ then $l_j > l_i$ (i.e., the list is ordered by the position l_i of the string in the text).

Contiguous pairs of Title and Price values are combined to form Book instances. Each of these instances is labeled with the smaller position value (also called *lowest component*). Notice that the Book instance labeled l_9 is built with components that appear in inverted order (relative to the order for previous objects). 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 first step is represented by the second row in Figure 11.

As for the second step, runs of Book instances are grouped into lists of Book instances (indicated as **{Book}**). The third step simply combines these Book lists with Name instances (in our example, l_1 is combined with l_2 and l_8 is combined with l_9), as illustrated on the top of Figure 11.

On the Basic Assumptions of Our Bottom-up Algorithm

The Assembling Phase procedure is based on two fundamental assumptions. First, that AVPs can be correctly identified and extracted from a text (page), i.e., each avp-pattern determines a set of instances of an atomic type τ . Second, that the presence of any component of an instance indicates the existence of such an instance. Therefore, if many of the AVPs correspond to incorrect strings (false positives), the assembling phase may form spurious objects. Further, if some values are not captured (false negatives), the assembling phase may create wrong complex objects containing, for instance, atomic objects that belong to other objects.

The problem of detecting false positives and false negatives in Web data extraction is, indeed, common to many approaches proposed on the literature. For instance, in [33] the author describes a *corroboration algorithm* that uses simple domain-specific heuristics to verify the values extracted. In [30] the authors propose the use of “negative examples” to make their extraction rules more effective.

In DEByE, many of the problems caused by imperfect avp-patterns can be alleviated by the features of the interface. That is, the user can provide new examples, change the estimated number of occurrences of instances in the source page, and mark some attributes as being mandatory in object instances. Obviously, there are cases for which this will not work at all. Further, it is frequently possible to build counter-examples that can break any heuristic one can devise. This is also the case of all the other approaches proposed on the literature. Thus, experimentation is a must to verify the spectrum of application of any semistructured data extraction algorithm. In the case of DEByE, we consider 15 distinct sources of Web data, including 3 of the more complex data sources in the RISE repository [40]. Our experimental results demonstrate that DEByE is an effective data extraction tool, which presents advantages (such as easiness of use, quick prototyping, and coverage of a variety of data sources with variations in structure) when compared to other approaches in the literature.

Top-Down versus Bottom-up

The top-down strategy recognizes objects in their entirety. Thus, recognition of partial objects (i.e., objects that are missing a component) is not done. Further, objects that contain components out of order are also not recognized. To recognize partial objects, the top-down strategy

depends on a potentially large set of example object patterns. To illustrate, consider that the user specifies as example an object with two levels and three atomic components labeled o_a , o_b , and o_c . Retrieval of all possible partial matching objects would require seven distinct object patterns (one pattern for the complete object, three patterns to indicate the absence of a single component, and three patterns to indicate the absence of two components). For large example objects, the number of cases might be exponential in the number of atomic components. This makes the top-down extraction procedure very inefficient in time (because each new page has to be processed independently for each example pattern) and far less useful in practice.

The bottom-up strategy is more flexible than the top-down strategy because it assembles complex objects through a composition of simpler object components. Thus, this strategy is specially suitable for cases where missing components or components out of order are expected [46]. Because of this characteristic, we implemented the extractor module of the DEByE tool using the bottom-up strategy.

4 The DEByE Tool

As shown in Figure 5, the DEByE tool comprises two separate modules called *Graphical User Interface (GUI)* and *Extractor*, which we now discuss in more detail.

The *GUI* module allows the user to specify the example objects. At the beginning, the user is presented with the screen in Figure 12. The screen includes three separate windows: the *Source Window*, the *Table Window*, and the *Pattern Builder Window*. The *Source Window* displays a Web page from a given Web source. In this case, the page is from the Murder by the Book Web site that we have been using throughout our discussion. The *Table Window* is used to assemble example objects. The *Pattern Builder Window* is only used at the end of a session to build the actual object extraction patterns.

To assemble an example, the user marks pieces of data from the page in the *Source Window* and copies them into the columns of the *Table 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 12). These operations allow the user to build a (nested) table that embeds the structure of the example object.

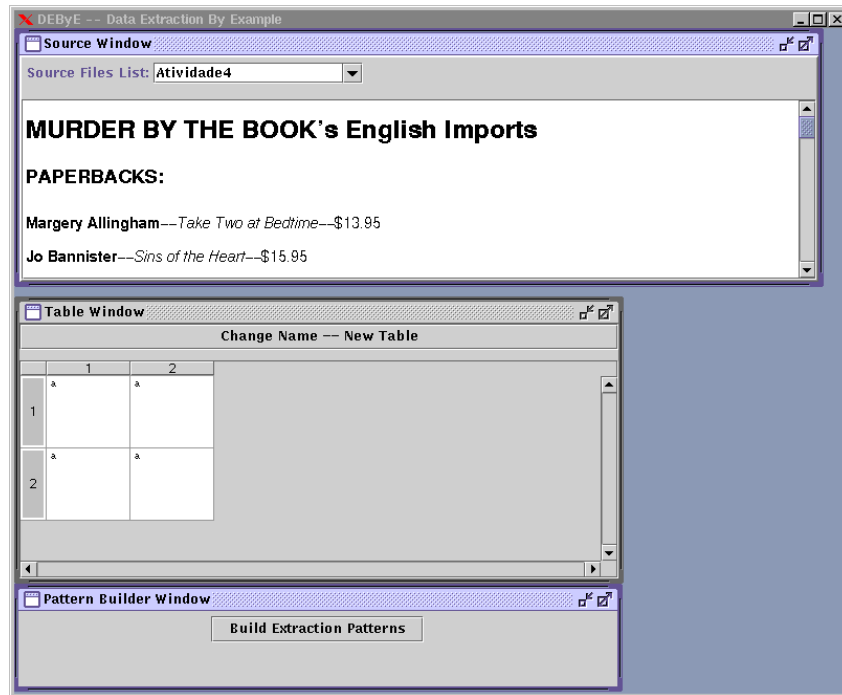


Figure 12: Main screen provided by the *DEByE GUI* module.

In Figure 13, the user has advanced in his example specification task. Having observed that *Agatha Christie* has written many books, he decided to compose the attributes Title and Price into a nested table, called *Book*, which can be used to hold various books written by *Agatha Christie*. This means that the example object being specified is now a two-level hierarchical object as illustrated in Figure 2. Also, to be able to extract a larger fraction of the objects from this site, the user needs to specify a second example reflecting objects with a different structure. In this case, the second example corresponds to the data on the author *Leslie Charteris*, whose books have a same price.

Once the user has specified his example objects, he activates the *Pattern Builder Window* to build the corresponding object extraction patterns to be used by the *Extractor* module. The *Extractor* module receives oe-patterns from the *GUI* module and matches them against new incoming Web pages. For this, it uses the bottom-up extraction strategy discussed in Section 3.

The DEByE extractor outputs the extracted data in an XML-based format that is used to represent semantic constructs such as tuples, lists, and variants. The tags are properly nested, with the atomic objects (AVPs) placed inside the innermost ones, following the structure of the

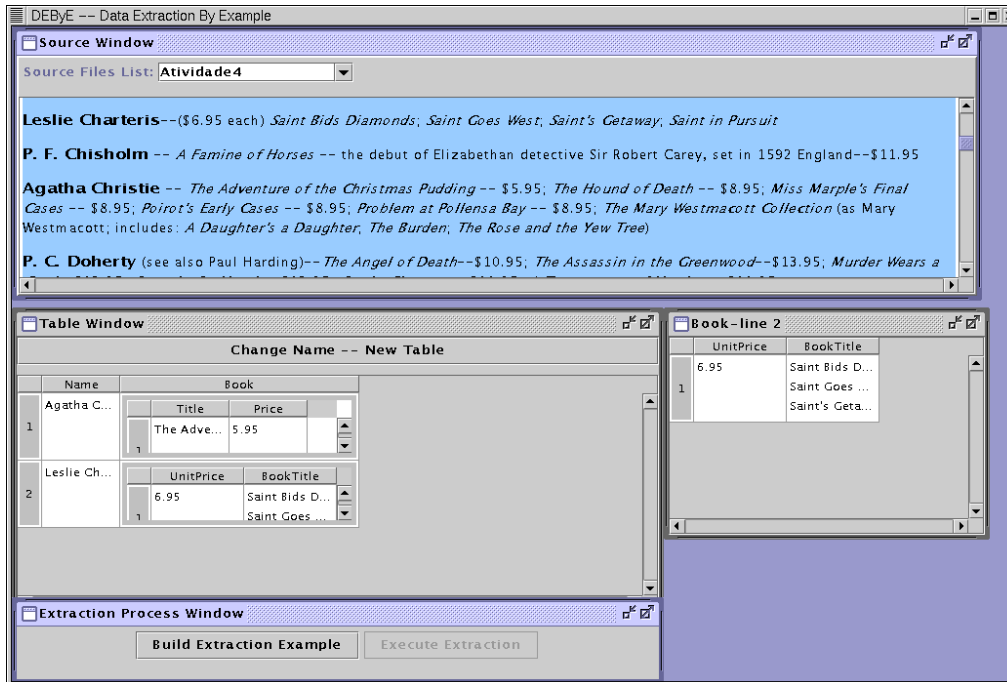


Figure 13: Specification of two example objects presenting structural variations.

objects extracted. As an example, Figure 14 illustrates an excerpt from the output of the *Extractor* module when it is fed with the examples specified in Figure 13 (the actual output was edited to fit the page). We notice that each object encompasses its structure according to its corresponding oe-pattern.

5 Experimental Results

In this section, we present the results of experiments we carried out using the DEByE tool. For 15 Web data sources, including 3 of the most complex data sources in the RISE repository [40], we fed the DEByE *Extractor* with oe-patterns generated using the DEByE *GUI*. The goal is to demonstrate the features of our bottom-up extraction strategy, as implemented in the DEByE tool, and the effectiveness of the whole approach.

We analyzed various sample pages from each of the 15 sources used in our experiments and manually identified and counted the implicit objects in each of them. These objects were then used to verify the precision of our extraction procedure. Since the identification of implicit objects is a laborious job, we could not always use in our experiments all the objects present in all of the

```

<TUPLE type="Author">
  <ATOM type="Name">
    <VALUE ipos="1777" fpos="1793">Leslie Charteris</VALUE>
  </ATOM>
  <LIST type="Book">
    <TUPLE type="Book">
      <ATOM type="UnitPrice">
        <VALUE ipos="1802" fpos="1806">6.95</VALUE>
      </ATOM>
      <LIST type="Title">
        <ATOM type="Title">
          <VALUE ipos="1815" fpos="1834">Saint Bids Diamonds</VALUE>
        </ATOM>
        ...
        <ATOM type="Title">
          <VALUE ipos="1891" fpos="1907">Saint in Pursuit</VALUE>
        </ATOM>
      </LIST>
    </TUPLE>
  </LIST>
</TUPLE>
...
<TUPLE type="Author">
  <ATOM type="Name">
    <VALUE ipos="2058" fpos="2073">Agatha Christie</VALUE>
  </ATOM>
  <LIST type="Book">
    <TUPLE type="Book">
      <ATOM type="Title">
        <VALUE ipos="2084" fpos="2122">The Adventure of the Christmas Pudding</VALUE>
      </ATOM>
      <ATOM type="Price">
        <VALUE ipos="2131" fpos="2135">5.95</VALUE>
      </ATOM>
    </TUPLE>
    ...
    <TUPLE type="Book">
      <ATOM type="Title">
        <VALUE ipos="2257" fpos="2280">Problem at Pollensa Bay</VALUE>
      </ATOM>
      <ATOM type="Price">
        <VALUE ipos="2289" fpos="2293">8.95</VALUE>
      </ATOM>
    </TUPLE>
  </LIST>
</TUPLE>

```

Figure 14: Excerpt of an output of the DEByE *Extractor*.

sample pages (i.e., sometimes we used only a subset of all sample pages). Despite this limitation, we managed to always use at least a few hundred objects in each case.

5.1 Experiments with Representative Web Sources

We first performed extraction experiments with 12 data rich Web sources – the focus of our interest. From each of these sources, we collected a subset of pages which we used to test DEByE. The majority of the sources are listed in the top positions in the list of the 100Hot Web site² [1] in their respective categories. The selected sites are the best ranked data rich Web sites in their categories. In the immediate following, we briefly describe the main features of each Web data source used. To describe the structure of the objects found in the Web sources, we use the notation introduced in Section 3.1.

Placebo	Without You I'm Nothing	\$16.97	\$11.88
Portishead	Pnyc	\$16.97	\$11.88
Louis Prima	Collectors Series	\$11.97	\$8.38
Queen	Greatest Hits I & II	\$29.97	\$20.98
R.E.M.	Up	\$16.97	\$11.88

(a) CDNow

Date	Location	Job Title	Company
Jun 7	US-TX-Austin	Software Developer - Web	ESG Consulting
Jun 7	US-CA-San Ramon	Web Interface Engineer - 99 Stock	IMS Net Corporation
Jun 7	US-FL-Ft. Lauderdale	GREAT VC++/INTERNET OPPORTUNITY !!!	Maxim Group
Jun 7	US-IN-Indianapolis	System Engineer	Wang Global
Jun 7	US-New Jersey	Systems Analyst II	A-R-C (Alternative Resources)
Jun 6	US-CA-San Jose	Senior Network Engineer	Infonet, Inc.
Jun 6	US-WI-Milwaukee	System Engineer	Wang Global
Jun 6	US-Illinois	Web Developer	IMI Systems
Jun 6	US-IL-Chicago	System Engineer	Wang Global
Jun 6	US-CA-San Francisco	Web Developer For Large Financial Site	IMI Systems

(b) Monster

Figure 15: Excerpts of pages from CD Now and Monster Web Sites.

CDNow. We collected the “30% discount” pages of the site. In these pages, the structure of the CD entries are very regular with no components missing (see Figure 15(a)). We modeled the CD entries in these pages as instances of the type $CD=(ArtistName, Title, Price, Discount)$. All 219 CD

²This Web site maintains a list of the most popular Web sites in many categories such as shopping, jobs, entertainment, etc. Whenever we refer to the popularity of a Web site in the 100Hot ranking, we are considering its position by the time the experiments were carried out.

<p>The Internet for Dummies In-Stock: Ships within 24 hours. John R. Levine, Margaret Levine Young, Carol Baroudi / Paperback / Date Published: February 1999 Retail Price: \$19.99 Our Price: \$15.99, You Save \$4.00 (20%) ▶ Buy this book or read more about it</p> <p>The Internet: The Rough Guide In-Stock: Ships 2-3 days. Angus J. Kennedy, Rough Guides (Editor) / Paperback / Date Published: November 1998 Retail Price: \$8.95 Our Price: \$7.16, You Save \$1.79 (20%) ▶ Buy this book or read more about it</p> <p>The Internet for Dummies: Quick Reference In-Stock: Ships within 24 hours. John R. Levine, Margaret Levine Young, Arnold Reinhold / Paperback / Date Published: January 1999 Retail Price: \$12.99 Our Price: \$10.39, You Save \$2.60 (20%) ▶ Buy this book or read more about it</p>

(a) Barnes & Noble

<p>Title 1995 Human-Computer Interaction Laboratory Video Reports Author(s) Catherine Plaisant (Edited by) Document ID ncstrlumcp/CS-TR-3532 Institution University of Maryland, College Park</p> <p>Title Abstract Semantics of Synchronous Languages: The Example Esterel Author(s) Manfred Broy Document ID ncstrl.tu.munich_cs/TUM-19706 Institution Technical University of Munich</p> <p>Title Adaptive Scheduling with Client Resources to Improve WWW Server Scalability Author(s) Daniel Andresen and Tao Yang Document ID ncstrlucsb_cs/TRCS96-27 Institution University of California, Santa Barbara. Computer Science.</p>

(b) NCSTRL

<p>Sam Lord's Castle - Barbados \$333- (5 nts.) Stay at this legendary landmark castle on one of the world's most beautiful beaches! Dive into 3 pools, soak in the whirlpool, play tennis on lighted courts, sail, snorkel, waterski, and work out in the fitness room. All on 72 landscaped acres.</p> <p>3-Night Southern Caribbean Cruise \$349- Now you can explore the faraway islands of the Southern Caribbean on a three-night cruise -- and save a minimum of \$140 off the brochure rate! Aboard the <i>Nordic Empress</i>, you will depart San Juan and visit St. Thomas and St. Maarten.</p> <p>Wyndham Aruba Beach Resort & Casino \$366- (5 nts.) Couples, singles and families alike will love this resort located on Aruba's most exclusive beach. Enjoy swimming, water sports, tennis and nearby championship golf. And the Kids Klub is fun for all ages.</p>

(c) Travelocity

Figure 16: Excerpts of pages from Barnes & Noble, NCSTRL, and Travelocity Web Sites.

entries in the source pages were recognized as instances of the CD type and extracted with just one example provided. CDNow [16] is the 5th most popular site under the category *music* in the 100Hot ranking and is the best ranked CD store in this list.

Monster. Monster.com [38] is, according to the 100Hot ranking, the most popular site in the category *job*. We did a keyword search using “WWW” as an argument and used the first 500 job offer entries returned in our experiments. We modeled the entries in these pages as instances of the object type $Job=(Date,Location,Title,Company)$. With just one example provided, we were able to retrieve all 500 entries as instances of this type. An excerpt of one of these pages is presented in Figure 15(b).

Barnes & Noble. For this electronic bookstore, the second most popular site in the *book* category of 100Hot, we collected the set of pages returned by using the keyword “Internet”. We used the first 200 entries returned in our experiments. An excerpt of one of these pages is shown in Figure 16(a). We modeled the book entries as objects of the type $Book=(Title,Authors,Price)$. The

Brazil

Geography

[\[Top of Page\]](#)

Location: Eastern South America, bordering the Atlantic Ocean

Geographic coordinates: 10 00 S, 55 00 W

Map references: South America

Area:
total: 8,511,965 sq km
land: 8,456,510 sq km
water: 55,455 sq km
note: includes Arquipelago de Fernando de Noronha, Atol das Rocas, Ilha da Trindade, Ilhas Martin Vaz, and Penedos de Sao Pedro e Sao Paulo

Area-comparative: slightly smaller than the US

(a) CIA Factbook

BRAND	PRODUCT	DESCRIPTION	PRICE	DOMESTIC Ship/Handling	DATE/HR	DEALER	ST	PART#
Generic	Slimnote 2B, Bare Bones notebook, CPU, RAM & Hard Drive optional, carrying case	13.3inch TFT (Supports Pentium® II-233, 266 & 300 CPU), Max 128MB SDRAM, 24K CDROM, 1.44M FDD	\$ 1081	No Data	1/28/99 9:17:49 PM CDT	Clifford Technology, Inc. 888-807-7253 714-526-0771 -- P.O. accepted Online Ordering	CA	SYS-SNF2B
Sager	Np6600, Bare Bones Notebook CPU, RAM & Hard Drive optional	12.1inch TFT (Supports Pentium® II-233, 266 and 300 CPU), Max 128MB SDRAM, 24K CDROM, 1.44M FDD, carrying case	\$ 1119	No Data	1/28/99 9:08:22 PM CDT	Comtrade 800-969-2123 626-961-6688	CA	-

(b) Price Watch

[Internet Publishing Kit](#)
 Hardcover / Published 1995
 Our Price: \$149.95 (*Special Order*)

[Internet Publishing with Microsoft Word 7.0; With CDROM With CDROM](#)
 Connie Dunn / Paperback / Published 1998
 Our Price: \$29.95 (*Not Yet Published*)
[Read more about this title...](#)

[Internet Quick Reference Guide](#)
 Glenn Davis / Paperback / Published 1997
 Our Price: \$9.60 - *You Save: \$2.40 (20%) (Back Ordered)*
[Read more about this title...](#)

(c) Amazon

Figure 17: Excerpts of pages from CIA Factbook, Price Watch, and Amazon Web sites.

entries present some variations in their structure (there are missing components in some of them), which required us to provide two separate examples. As a result, we were able to retrieve all the 200 book entries as instances of Book.

NCSTRL. The NCSTRL (Networked Computer Science Technical Reference Library) [44] is a very popular repository of computer science technical reports from many institutions all over the world. We issued a query using “WWW” as a keyword and retrieved a page with 177 entries. Some entries are shown in the excerpt of the page presented in Figure 16(b). The entries of these pages were modeled as instances of the object type Report=(Title,AuthorName,Institution). One single example was provided. Despite the poor HTML formatting exhibited by the page, we were able to recognize all 177 entries as instances of Report.

Travelocity. We collected the page on vacation packages from the Travelocity Web site [50], ranked as the most popular site in the *travel* category of 100Hot. The page on vacations packages contained 162 entries. We modeled the objects as instances of the type `Package=(Place,NoOfNights,Price)`. One single example was provided. We intentionally ignored a free text description of each vacation package, present in each entry. For some of the entries, the information on the number of nights was missing. With this single example provided, all 162 entries were recognized as instances of `Package` and extracted. Figure 16(c) shows an excerpt of the Travelocity page.

CIA Factbook. This Web site [17] provides a detailed profile of 266 political entities, most of them countries. Every political entity has a profile, in the form of an HTML page, including information on geography, people, economy, government, etc., organized in the form of labeled fields. Figure 17(a) shows an excerpt of the page corresponding to the profile of Brazil. As there are hundreds of fields in each profile, we chose just a few of them which were modeled as instances of the object type `Country=(Name,Location,Coordinates,Area:(Total,Land,Water),Population,NationalCapital)`. We considered the first 50 profiles (in the order they appear in the main site's page) in our experiments. With one single example given, we obtained 36 complete instances (72%), 13 incomplete instances (26%) (with one attribute missing in comparison with the profile), and 1 incorrect instance (2%) (which included an incorrect attribute instance). We then provided one additional example and obtained as a result 43 complete instances (86%), 6 incomplete instances (12%), and 1 remaining incorrect instance (2%). With two more examples (thus using a total of 4 examples), we got 49 complete instances (98%), but the incorrect instance remained. This instance corresponds to the profile of *Antarctica*, which has some features quite distinct from those found in the profile of the other countries.

Price Watch. The Price Watch Web site [51] provides information on prices of computers and computer parts from several vendors. It occupies the 11th place in the category *hardware* of 100Hot, but it is the best ranked site that is not a Web site of a specific vendor. We collected a set of pages on Pentium II 266 Mhz notebooks. The resulting pages are formatted as HTML tables with one row for each item. Figure 17(b) shows an excerpt of one of these pages. There were a total of 113 items. Instances of the type `Item=(Brand,Model,Description,Price,Date,Dealer=(Name,Phone),State,PartNo)` were provided as examples of the objects found in this site. With one example, we retrieved 57

CNN interactive weather **worldforecasts**

Belo Horizonte, Brazil Forecast 1 p.m. (local), Jun. 15

FOUR-DAY FORECAST

Day	Icon	High	Low
Tue	partly cloudy	23 C 74 F	13 C 57 F
Wed	cloudy	23 C 73 F	13 C 57 F
Thu	partly cloudy	24 C 75 F	15 C 60 F
Fri	partly cloudy	25 C 77 F	15 C 58 F

CURRENT CONDITIONS
cloudy
Temp: 20 C, 68 F
Rel. Humidity: 72%
Wind: ESE at 9 kph (6 mph)
Sunrise: 6:26 a.m.
Sunset: 5:23 p.m.

RELATED LINKS

- World News America
- Travel Companion
- Think World of you
- CNN World Market
- CNN World Sport
- CNN e-Exchange

WEATHER MAPS

SATELLITE FORECAST

Four-day forecasts for 7,200 cities worldwide

(a) CNN World Whether

I. VLDB 1975: Framingham, Massachusetts

Douglas S. Kerr (Ed.): Proceedings of the International Conference on Very Large Data Bases, September 22-24, 1975, Framingham, Massachusetts, USA. ACM

```

$proceedings(DBLP:conf/vldb/75,
  editor   = {Douglas S. Kerr},
  title    = {Proceedings of the International Conference on Very Large Data
    Bases, September 22-24, 1975, Framingham, Massachusetts, USA},
  publisher = {ACM},
  year     = {1975},
  bibsource = {DBLP, 'http://dblp.uni-trier.de'}
)

```

Data Description Models I

- Moshé M. Zloof:
Query-by-Example: the Invocation and Definition of Tables and Forms. 1-24,
Electronic Edition
- Michael Hammer, Dennis McLeod:
Semantic Integrity in a Relational Data Base System. 25-47,
Electronic Edition
- Kapali P. Eswaran, Donald D. Chamberlin:
Functional Specifications of Subsystem for Database Integrity. 48-68,
Electronic Edition

(b) VLDB page at DB&LP

Figure 18: Excerpts of pages from CNN World Whether Web site and VLDB at DB&LP.

complete items (50%) and 56 incomplete items (50%) (with one or two attributes missing in comparison with the item in the page). With two examples, we retrieved 90 complete items (80%) and 23 incomplete items (20%) (with just one attribute missing in comparison with the item in the page).

Amazon. For this electronic bookstore, the most popular site [5] in the *book* category according to 100Hot, we collected the pages satisfying the title search whose argument is the word “Internet”. An excerpt of one of these pages is presented in Figure 17(c). As a rule, book entries present in the returned pages were very poorly formatted, what is particularly true for the information on authors. Interestingly, the first pages resulting from this query present book entries with very regular structure, but the entries in the last pages are increasingly less regular and poorly formatted. This is an unexpected finding, since the pages are supposed to be ordered by the title of the books present in it. The query returned a total of 4421 book entries in 89 pages. We analyzed 5 book entries in each page, to take into account the degradation in the structure of the entries. Thus a total of 445 entries were analyzed (our sample would be far more regular in structure if we had used the 445 best ranked entries). The examples were provided as instances of the object type $Book=(Title,Authors,Price)$ (as in the Barnes & Noble experiment). With one single example taken from the first page, 205 instances (46%) were completely retrieved and 240 (54%) were retrieved with missing attributes. Using an additional example, we obtained 289 complete instances (65%) and 156 incomplete (35%) instances. Increasing the number of examples to 3, 4,

Jeffrey D. Ullman

List of publications from the [DBLP Bibliography Server](#)

[Homepage](#)

		1999
199	EE	Ramana Yerneni, Chen Li, Jeffrey D. Ullman, Hector Garcia-Molina: Optimizing Large Join Queries in Mediation Systems. <i>ICDT 1999</i> : 348–364
198		Jeffrey D. Ullman: Some Advances in Data-Mining Techniques (Abstract). <i>NGITS 1999</i> : 1
197	EE	Ramana Yerneni, Chen Li, Hector Garcia-Molina, Jeffrey D. Ullman: Computing Capabilities of Mediators. <i>SIGMOD Conference 1999</i> : 443–454
196		Alon Y. Levy, Anand Rajaraman, Jeffrey D. Ullman: Answering Queries Using Limited External Query Processors. <i>ICSS 58(1)</i> : 69–82 (1999)
		1998
195	EE	Shalom Tsur, Jeffrey D. Ullman, Serge Abiteboul, Chris Clifton, Rajeev Motwani, Svetlozar Nestorov, Arnon Rosenthal: Query Flocks: A Generalization of Association-Rule Mining. <i>SIGMOD Conference 1998</i> : 1–12
194	EE	Chen Li, Ramana Yerneni, Vasilis Vassalos, Hector Garcia-Molina, Yannis Papakonstantinou, Jeffrey D. Ullman, Murty Valiveti: Capability Based Mediation in TSIMMIS. <i>SIGMOD Conference 1998</i> : 564–566
193	EE	Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, Jeffrey D. Ullman: Computing Iceberg Queries Efficiently. <i>VLDB 1998</i> : 299–310

(a) Author page at DB&LP

<p>Agatha Christie — <i>The Adventure of the Christmas Pudding</i> — \$5.95; <i>The Hound of Death</i> — \$8.95; <i>Miss Marple's Final Cases</i> — \$8.95; <i>Poirot's Early Cases</i> — \$8.95; <i>Problem at Pollensa Bay</i> — \$8.95; <i>The Mary Westmacott Collection</i> (as Mary Westmacott; includes: <i>A Daughter's a Daughter</i>; <i>The Burden</i>; <i>The Rose and the Yew Tree</i>)</p> <p>Dorothy Dunnnett — <i>Niccolo Rising</i> — \$15.95; <i>Race of Scorpions</i> — \$15.95; <i>Scales of Gold</i> — \$15.95; <i>The Spring of the Ram</i> — \$15.95</p> <p><i>Missing Person</i> (1993) — \$11.95; <i>No Fixed Abode</i> (1994) — \$11.95; <i>Identity Unknown</i> (1995) — \$13.95</p> <p>Dick Francis — <i>The Sport of Queens</i> (autobiography) — \$10.95</p>

(b) Murder by The Book

Figure 19: Excerpts of pages from (a) an Author Page at DB&LP and (b) a page from Murder by the Book Web Site.

and 5, the fraction of instances retrieved in their entirety also increased to 74%, 93%, and 96%, respectively.

CNN World Forecast. We collected weather forecast pages from the CNN Web site [18], one of the three most popular sites in the 100Hot *news* category, relative to 100 cities (i.e., we obtained 100 object instances). Figure 18(a) shows an excerpt of a city's page. We modeled the objects as instances of the object type $\text{Forecast}=(\text{City},\{\text{WeekDay}=(\text{Day},\text{Temp}=(\text{High}=(\text{HC},\text{HF})),\text{Low}=(\text{LC},\text{LF}))\})$ which reflects our interest in the maximum and minimum temperatures (in Celsius and Fahrenheit) for each day of a given week. With just one example, we were able to recognize and correctly extract the 100 instances of this type.

VLDB Pages at DB&LP. We collected from the DB&LP Web site the pages with the contents of the VLDB proceedings from 1975 to 1983. There was one page per proceedings, except for the proceedings of the 1978's conference that had two pages. Examining just the page corresponding to the 1975 proceedings, illustrated in Figure 18(b), we specified a single example with the following type: `Proceedings=(Year,Place,{Section=(SecTitle,{Article=(ArtTitle,{AuthorName},Pages)}}))`. The example for this object type includes one instance of `Proceedings`, two instances of `Section`, and three instances of `Article`. We then used this single example to retrieve object instances from all the 9 source pages and obtained the following results. From 461 instances of `Article` manually identified, 424 (92%) were retrieved with all their attributes and 37 (8%) were retrieved with one or two attributes missing (values of `AuthorName` were totally retrieved). From 144 instances of `Section` manually identified, 134 (93%) were retrieved with all components and 10 (7%) were retrieved with missing components. Note that this takes into account the components missing in instances of `Article` that compose a `Section` instance. From the 10 instances of `Proceedings`, 4 were completely retrieved and 6 were retrieved with missing components. Examining these results, we notice that most of the problems were found in objects related to the page of the 1976 proceedings and to the second page of the 1978 proceedings. This occurred because the 1976 proceedings were not organized in sections, thus having a structure very distinct from the example given, and because the second page of 1978 proceedings, although having the same structure, had very different formatting features for the article entries.

Author Page at DB&LP. For this experiment, we took Professor Jeffrey Ullman's page at DB&LP. An excerpt of this page is present in Figure 19(a). To capture the variations in the implicit structure of the objects in this page, we used two type definitions. The first is a type defined as `Article=(Title,{AuthorName},Pages,Journal,Number)`, while the second is a variation of this type defined as `Article=(Title,{AuthorName},Pages,Proceedings)`. We provided 5 examples of article entries, using these two types. As a result, we were able to correctly extract 554 instances of `AuthorNames` (100%) and 196 complete instances of `Article` (98%). The remaining 3 `Article` instances (less than 2%) were retrieved with a single missing attribute.

Murder by the Book. *Murder by The Book* [39] is a bookstore specialized in mystery books. Its pages are very poorly formatted (for instance, the data on book prices is formatted in many dif-

ferent ways). For that reason, we chose this site as a particular case of interest. As in the previous experiment, for this site there was the need of modeling variations for the type `Author`. For this, we provided example objects that were either instances of the type `Author=(Name,{Book=(Title,Price)})`, or instances of the type `Author=(Name,{Book=(UnitPrice,{BookTitle})})`. We then extracted object instances from a page (the “English Imports” page) containing 147 books entries grouped into 49 author entries. With respect to `Book` instances, we extracted 129 of them completely (89%) and 16 (11%) with missing attributes (`Price`), when 5 examples were provided. Regarding the 49 `Author` instances, we extracted 40 of them (82%) completely and 9 (18%) with some missing component, when 2 examples were provided.

Analysis of our Results

Table 1 summarizes our results. The percentage figures for the number of objects retrieved are relative to the total number of objects identified manually in the sources pages. The number of examples used in the extraction is determined by trial and error. When the number of examples is insufficient, the results of the extraction process include many objects with missing attribute values. It is then necessary to provide more examples to obtain more complete (and precise) results. The number of examples is increased until the quality of the results improves. In Table 1, we only show the points at which the results were improved considerably. For instance, for the *Murder by the Book* site, we only show results when 2 and 5 examples are provided. As we can see, the DEByE *Extractor* was able to recognize and retrieve most objects in the given pages.

In Table 2 we present the time (in seconds) spent generating oe-patterns, extracting AVPs (i.e., atomic values), and assembling complex objects using the extracted AVPs. These times refer to the extraction carried out using the number of examples shown in Table 1. The column `Size` refers to the size of the sample Web page used to generate the avp-patterns and the column `Total Size` refers to the total size (in bytes) of the pages from where the complex objects were extracted. The column `Total Time` indicates the total time spent in each case. As we can observe, the whole extraction procedure can be completed in a matter of seconds, even for complex sites.

Although we have limited our experiments to a few hundred objects of each type, it is worth noting that almost all of the Web sites considered contain a large amount of pages which have the same overall structure and formatting features. Thus, many more objects could have been

Web Source	Object Type	Total	Examples	Objects Retrieved	
				Complete	Incomplete
CDNow	CD	219	1	219 (100%)	–
Monster	Job	500	1	500 (100%)	–
Barnes & Noble	Book	200	2	200 (100%)	–
NCSTRL	Report	177	1	177 (100%)	–
Travelocity	Package	162	1	162 (100%)	–
CIA Factbook	Country	50	1	36 (72%)	13 (26%)
			2	43 (86%)	6 (12%)
			4	49 (98%)	–
Price Watch	Item	113	1	57 (50%)	56 (50%)
			2	90 (80%)	23 (20%)
Amazon	Book	445	1	205 (46%)	240 (54%)
			2	289 (65%)	156 (35%)
			3	327 (74%)	118 (26%)
			4	417 (93%)	28 (7%)
			5	429 (96%)	16 (4%)
CNN	City	100	1	100 (100%)	–
	Weekday	400	1	400 (100%)	–
VLDB Pages at DB&LP	Proceedings	10	1	4 (40%)	6 (60%)
	Section	144	2	134 (93%)	10 (7%)
	Article	461	3	424 (92%)	37 (8%)
Author Page at DB&LP	Article	196	5	199 (98%)	–
	Author	554	12	554 (100%)	–
Murder by the Book	Author	49	2	40 (82%)	9 (18%)
	Book	147	5	130 (89%)	16 (11%)

Table 1: Number of objects retrieved by the DEByE *Extractor* for various Web sources.

retrieved from these sites. For instance, for the Amazon pages, we recall that the number of objects actually extracted (4421) was equal to the total number of book entries retrieved by the query, although we have analyzed only 445 book entries.

As a final observation, the fact that our extraction strategy was very effective with data available in popular Web sites (such as the ones we used in our experiments) confirms our hypothesis on how data is usually made available on the Web. In fact, the pages of many very popular and interesting Web sources present an inherent implicit structure that can be recognized and modeled accurately. Furthermore, data contained in these pages are usually surrounded by contextual information which allows their effective recognition and extraction.

5.2 Experiments with Sources from the RISE Repository

RISE (Repository of Online Information Sources Used in Information Extraction Tasks) [40] is a repository of online information sources that were used for data extraction experimentation by various authors, most of them from the machine learning community. Among these, we distinguish the research on the WIEN [32], Stalker [42] and SoftMealy [29] systems. Our purpose here is to

Web Source	oe-pattern Generation		Object Extraction			Total Time (sec.)
	Time (sec.)	Size	Time (sec.)		Total Size	
			AVP Extraction	Assembling		
CDNow	0.223192	31654	0.395016	0.095113	301892	0.713321
Monster	0.683688	39521	0.397513	0.233642	363686	1.314843
Barnes & Noble	0.526398	51565	2.782157	0.100536	1342861	3.409091
NCSTRL	3.200000	160531	0.221553	0.094289	160531	3.515842
Travelocity	0.311210	76067	0.345100	0.048700	76067	0.705010
CIA Factbook	3.004563	42045	0.364568	0.051170	4288460	3.420301
Price Watch	1.520407	17460	0.325296	0.149700	151488	1.995403
Amazon	1.917623	33551	7.242873	1.497996	2613915	10.658492
CNN	0.523155	32127	0.165441	0.107794	727400	0.796390
VLDB Pages at DB&LP	0.942920	18173	0.702879	0.573590	532403	2.219389
Authors Page at DB&LP	3.601458	73814	0.220965	0.229656	73814	4.052079
Murder by The Book	0.338172	10853	0.041062	0.058324	10853	0.437558

Table 2: Time spent generating oe-Patterns, extracting AVPs, and assembling complex objects in the extraction experiments.

provide a preliminary comparison of DEByE with these three other systems, using data sources from the RISE repository.

In Table 3 we present experimental results obtained by running DEByE on three of the most complex data sources in RISE: Okra, BigBook, and IAF. For comparison, we also present published results obtained for these three sources by the WIEN, the Stalker and the SoftMealy systems. We first observe that, contrary to the other systems, WIEN takes whole pages as examples. Thus, when considering the number of examples used by the WIEN system, we take the number of pages given as examples and multiply them by the average number of objects in each page. This is what is shown in Table 3. We further observe that these three sources include only pages containing flat (i.e., a single level) objects, contrary to most sources we use in Section 5.1. As a result, DEByE was as effective as any of the other three systems for these three data sources, while frequently using a smaller number of examples and always requiring less effort on the user part for the specification of the examples. Further, DEByE is conceptually simpler than the other three systems and requires a less complex implementation. Because of that, it tends to be faster on conventional Web data sources.

Web Source	DEByE		WIEN		Stalker		SoftMealy	
	Ex.	Retrieved	Ex.	Retrieved	Ex.	Retrieved	Ex.	Retrieved
OKRA	2	100%	46	100%	1	97%	1	100%
BigBook	1	99%	274	100%	8	97%	6	100%
IAF	3	99%	–	–	10	85% – 100%	1	99%

Table 3: Results of experiments with the DEByE *Extractor* for RISE Web sources.

While the results in Table 3 shed some light on the relative performance of DEByE, they do not allow a full and direct comparison between DEByE and the three other systems. There are several

reasons. First, WIEN, Stalker and SoftMealy are all based on machine learning techniques, while DEByE is based on the identification of context through passage analysis (a well known information retrieval technique). Second, the experimental protocol used by the three other systems was much more exhaustive than ours. While in WIEN, Stalker and SoftMealy the examples were randomly chosen in several trials (30 for WIEN, 500 for Stalker and 300 for SoftMealy), in DEByE the users choose the examples at their will. In fact, a key point in the DEByE approach is to extract objects according to the users' preferences. The fact that most often the users are able to specify a useful example in the first trial indicates the validity of the DEByE approach. Third, the goals of the experimentation performed were distinct in the case of each system. In the WIEN experiments, the goal was to obtain 100% accuracy in the extraction. Thus, the number of examples was increased until this accuracy level was reached. In Stalker, the authors stopped the experiments after reaching 97% of accuracy or after 10 examples were given. SoftMealy, as DEByE, reached almost perfect accuracy with few examples, so that no limits needed to be imposed on the accuracy and in the number of examples given. However, in the experiments with SoftMealy, the authors used three alternative extraction strategies, while in DEByE the same algorithm was used for the three sources. This is an important point. In DEByE, there is no knowledge base or set of heuristics to guide the extraction procedure. All the adaptations in the extraction strategy are automatically generated by the same extraction algorithm, using solely the evidences presented in the user specified examples. Fourth, in DEByE we used only the sets of pages available from RISE (which were also used in the experiments with WIEN and Stalker) while in the experiments with SoftMealy the authors used an extended set with many more pages for each source.

Examining Table 3, we observe that the number of examples required by DEByE for each source does not vary much, as it occurs with the other systems. This may be explained by the fact that, in DEByE, we only generate patterns for extracting single atomic values that are assembled afterwards, according to the structure of the examples provided. As discussed in Section 3.3, the assembling of objects in DEByE does not rely on the ordering of the component objects. As a result, there is no need of generating alternative patterns for capturing distinct ordering of components.

These preliminary results with the RISE repository suggest that DEByE is as effective as the known alternatives based on wrapper induction.

6 Comparison with Related Work

Recent efforts in modeling and extracting semistructured data from Web sources are discussed in [11, 24, 41]. These efforts involve data models, wrapper induction, NLP techniques and ontology-based extraction approaches, as we now briefly review.

Several data models have been proposed to represent semistructured data [12, 13, 45]. These models are, in general, based on labeled directed graphs and aim at capturing the irregular structure inherent to such data. OEM (Object Exchange Model) is an object-based model adopted by the TSIMMIS project [45]. An OEM object can be of type either atomic or complex. The value of an OEM object of type complex is a set of object references to its components and these references can be cyclic. The data model proposed in [12] for the UnQL query language is quite similar to OEM. The difference is that the UnQL data model lacks the notion of an object, describing data by a means of a set of trees whose leaf nodes have the actual instances associated with them. The model presented in [13] also represents data as a directed labeled graph in which each node corresponds to an object. However, the edges emanating from any node (that describes data) are labeled distinctly. In our work, we adopt a simple data model in which complex objects are represented as nested tables. Our model is based on the ideas described in [3, 35] and can be seen as an extension of the relational model.

Wrappers constitute the most common approach to extracting data from Web sources [6, 7, 26, 28]. Originally, wrappers were manually written using some specific language [7, 27]. Although such languages provide an effective approach for wrapper generation, they have some drawbacks. First, they require a previous knowledge of the structure of the pages in the Web source. Second, they are sensitive to changes on the structure of the Web pages. Due to such limitations, efforts have been made to automate the wrapper generation process [6, 28, 43]. In particular, approaches based on examples (such as DEByE) have proved to be very efficient for the task of wrapper generation. This is because they do not require any previous knowledge of the target Web pages. Additionally, structural changes in those pages can be accommodated by simply providing new examples.

Many works in the literature propose the use of machine learning techniques to induce wrappers automatically or semi-automatically [19, 30, 33, 43]. In general, these approaches consist of using

training examples to generate automata that recognize instances in contexts similar to the ones of the given examples. The ShopBot comparison-shopping agent [19] generates wrappers to extract data from on-line stores and also to compare them. Its strong assumptions on the type of the pages and on the type of data appearing in these pages allow it to generate wrappers automatically, but compromise the generalization of its approach. The approach proposed in [33] and adopted in the WIEN system relies, like ours, on examples from the source to be wrapped. The main drawbacks of this pioneer work are: (1) it does not deal with missing or out of order components of the objects and (2) although it identifies the need for the extraction of complex objects present in a nested structured, the solution provided is computationally intractable and has not been implemented in the WIEN system. These two very important problems are addressed in SoftMealy [30] and Stalker [43]. Both systems also generate wrappers generalizing given examples through machine learning techniques and are very effective in wrapping several types of Web pages. The main problem with SoftMealy is that the absence of a component or out of order components must be represented beforehand by an example in order to allow proper recognition and extraction. Stalker [43] can deal with such variations in a much more flexible way, since each object component is extracted independently (as in our bottom-up algorithm).

A common feature to all the approaches above is that the extraction process relies on an implicit knowledge or on a separate description of the structure of the source document (e.g., Web page). In WIEN, pages are assumed to conform to one out of six pre-defined kinds of structure (for instance, they should have a head, then a body with a set of tuples, and then a tail) that must be flat. This might generate extraction difficulties when dealing with typical variations found in Web pages. SoftMealy allows for the occurrence of structural variations, but the limitation regarding flat objects remains. In Stalker, the extraction of nested objects is possible but the approach relies on a separate description of the structure source page. In DEByE, there is no need of specifying (or having knowledge of) the structure of the source page. Indeed, the examples provided by the users describe a structure for the objects to be extracted. This adds flexibility to the extraction process. The user focusses on the target structure of his interest and does not have to concern himself with the structure implicit in the Web source page. This is an important advantage of the DEByE approach.

Besides wrapper induction, there are other approaches for learning extraction patterns that are

more suitable for “less” structured data sources such as newspaper classifieds, seminar announcements, and job postings. These data sources present grammatical elements that are common to telegraphic style of writing. In general, these data extraction approaches use techniques typical of Natural Language Processing (i.e., semantic class, part-of-speech tagging, etc.), sometimes combined with the recognition of syntactical elements (e.g., delimiters). This is the case of the Rapier [14] and the SRV [25] systems. The WHISK system [49] goes beyond and addresses a large spectrum of types of documents ranging from rigidly formatted ones to free text. For formatted text, this system has a behavior that is closer to wrapper induction systems like WIEN [33].

An ontology-based approach to extracting data from Web sources is presented in [21, 22, 23]. The approach uses a semantic data model to provide an ontology that describes the data of interest including relationships, lexical appearances, and context keywords. By parsing this ontology, a relational database schema and a constant/keyword recognizer are automatically generated, which are then used to extract the data that will populate the database. Prior to the application of the ontology, the approach requires the application of an automatic procedure to extract chunks of text containing data “items” (or records) of interest [20, 21]. To work properly, the approach requires a careful construction of an ontology, a task that must be done manually by an expert in the domain of the ontology. On the positive side, if the ontology is representative enough, the extraction is fully automated. Furthermore, wrappers generated according to such an approach are inherently resilient (i.e., they continue to work properly even if the formatting features of the source pages change) and adaptable (i.e., they will work for pages from many distinct sources belonging to a same application domain). Indeed, these features are unique of this approach.

Although this approach also requires the user to provide a conceptual description of the data to be extracted, it is radically distinct from ours with respect to the extraction strategy adopted. While we rely on the textual context surrounding the data of interest, the ontology-based approach relies mainly on the expected contents of the pages, according to what was anticipated by the pre-specified ontology. As a consequence, while our approach fails with Web sources where data presents little context information in their surroundings, the ontology-based approach fails when faced with domains where ontologies are hard to be created. This is the case of on-line bookstores, CD stores and bibliographical reference sites. Also, while the ontology-based approach requires a specialist to build the ontology using a notation specially designed [21, 23], the DEByE approach

provides a visual metaphor that is more suitable for the casual Web user.

NoDoSE [4] is another tool that, like ours, adopts an user-driven approach for data extraction. This tool provides a graphical interface which the user uses to decompose a given document (e.g., a Web page) into a hierarchy that describes its structure. Additional documents of the same type are then provided to the tool and automatically parsed. If tuning is required (which frequently is the case), the user must inspect the results (using the interface), modify the hierarchy that describes the document, and use it to parse the pages again. The process is completed when all of the documents have been successfully parsed. NoDoSE requires the user to specify all the structure of the whole document (i.e., the set of pages provided as input) in a top-down fashion which, in some cases, might be hard to do. Despite these drawbacks, the approach is effective for a large class of textual documents once the parsing is concluded successfully. The most noticeable distinction between NoDoSE and DEByE is the way examples are provided by the user to generate the extraction patterns. While in NoDoSE the user must decompose the whole document by marking regions in the entire document body, in DEByE the user marks only atomic values which he organizes according to his perception of the implicit structure of the objects to extract. Besides, in DEByE the assembling of objects is supported by a quite intuitive and simple metaphor, namely the nested tables. As a result, the user is completely shielded from the specific formatting features of the source page he is dealing with.

Another interesting approach for user-driven Web data extraction is the XWRAP [36] system. In this tool, the user is presented with a syntax tree that describes the HTML structure of the page he is interested in. The nodes of this tree correspond to HTML tags (e.g., <TABLE>, <TR>, etc.). By browsing this tree, the user selects the portions of the page he is interested in. For each of these portions, the tool applies a special set of pre-defined extraction rules. For instance, there are rules for tables, lists, etc. The extracted data is outputted in XML, but the tags used in the final document are also derived from the source page, under the assumption that the page contains text that can be used as meta-data. The major drawback is the explicit use of the HTML syntax and structure. This is a remarkable distinction from DEByE, in which the page formatting features are completely transparent to the user.

The work described in [10], as ours, also uses information on the local context of a value to identify new values in new Web pages. However, it is entirely focused on the recognition of

instances of a pair of attributes only. Thus, there is no notion of complex objects, of specification of the structure, and of a more general extraction strategy.

7 Conclusions

In this work, we described an example-based approach to automatically extracting semistructured data, which we call DEByE. The novelty of our work lies on two facts. First, that the user specifies examples according to a structure of his liking (instead of being bounded to the structural information implicitly encoded for the source page). This is important because, for instance, the user might be interested in only a subset of the information encoded in the page and, as result, might be unwilling to specify the structure of the whole page. Second, that a new, simpler, and faster bottom-up extraction procedure, built using techniques from information retrieval, can take advantage of the user provided examples to recognize and extract new data with great efficacy.

We argued that convenience in the specification of the examples can be achieved through a user interface that adopts nested tables as its fundamental paradigm. Nested tables are simple, intuitive, and can be used effectively by various users with distinct Web sources. We also argued that the bottom-up extraction procedure is quite effective, because it is able to recognize new data even when such data is incomplete or appears out of order. Through experimentation with our extraction tool, we verified our assumptions and confirmed our arguments. Using 15 distinct types of Web sources, including three data sources from the RISE repository, we demonstrated that a few examples are enough to allow the recognition and extraction of almost the totality of the objects present in the Web sources we considered.

Our approach is also quite distinct with regard to the role the user plays in the whole extraction process. In DEByE, we allow the user to declare his perception of the structure of the objects in the pages of interest. That is, instead of trying to derive structural information from the formatting of the text, we let the user inform the structure as he perceives it. We believe that this approach presents advantages when compared with alternative methods that require the user to declare the structure of the whole page, as it was conceived.

Acknowledgments

We acknowledge the contributions of Elaine Spinola Silva and Karine Gomes Chaves, who implemented the first version of the DEByE interface, Paulo Golgher, who helped us with the experimentation, and Prof. Alberto Mendelzon from Toronto University, who provided us with useful comments on a first implementation of DEByE. We would also like to thank the anonymous reviewers for their comments on the first version of this paper. The work described in this paper was partially supported by Project SIAM (MCT/CNPq/PRONEX grant number 00418.00/00) and by the authors' individual grants from CNPq and CAPES.

References

- [1] 100 HOT. 100 Hot Web Site. <http://www.100hot.com/>.
- [2] ABITEBOUL, S., BUNEMAN, P., AND SUCIU, D. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, 1999.
- [3] ABITEBOUL, S., HULL, R., AND VIANU, V. *Foundations of Databases*. Addison-Wesley, Reading, Massachusetts, 1995.
- [4] ADELBERG, B. NoDoSE - A Tool for Semi-Automatically Extracting Structured and Semistructured Data from Text Documents. In *Proceedings of the ACM SIGMOD Conference on Management of Data* (Seattle, Washington, 1998).
- [5] AMAZON.COM, INC. Amazon.com Bookstore Web Site. <http://www.amazon.com>.
- [6] ASHISH, N., AND KNOBLOCK, C. Wrapper Generation for Semi-structured Internet Sources. *ACM SIGMOD Record* 26, 4 (1997), 8–15.
- [7] ATZENI, P., AND MECCA, G. Cut & Paste. In *Proceedings of the ACM Symposium on Principles of Database Systems* (Tucson, Arizona, 1997), pp. 144–153.
- [8] BAEZA-YATES, R., AND RIBEIRO-NETO, B. *Modern Information Retrieval*. Addison-Wesley, Harlow, England, 1999.

- [9] BRAY, T., PAOLI, J., AND SPERBERG-MCQUEEN, C. M. Extensible Markup Language (XML) 1.0.
<http://www.w3.org/TR/REC-xml>.
- [10] BRIN, S. Extracting Patterns and Relations from the World Wide Web. In *Proceedings of the World Wide Web and Databases, International Workshop - WebDB'98* (Valencia, Spain, 1999), pp. 172–183.
- [11] BUNEMAN, P. Semistructured Data. In *Proceedings of the Sixteenth ACM SIGMOD Symposium on Principles of Database Systems* (Tucson, Arizona, 1997), pp. 117–121.
- [12] BUNEMAN, P., DAVIDSON, S., HILLEBRAND, G., AND SUCIU, D. A Query Language and Optimization Techniques for Unstructured Data. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1996), 505–516.
- [13] BUNEMAN, P., DEUTSCH, A., AND TAN, W. A Deterministic Model for Semistructured Data. In *Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats* (Jerusalem, Israel, 1999).
- [14] CALIFF, M. E., AND MOONEY, R. J. Relational Learning of Pattern-Match Rules for Information Extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI'99)* (Orlando, Florida, 1999), pp. 328–334.
- [15] CALLAN, J. P. Passage-Level Evidence in Document Retrieval. In *Proceedings of the ACM SIGIR Conference on Information Retrieval* (Dublin, Ireland, 1994), pp. 302–309.
- [16] CDNOW, INC. CDNow: The Internet's Number One Music Store.
<http://www.cdnow.com>.
- [17] CIA. The CIA World Factbook Web Site.
<http://www.odci.gov/cia/publications/factbook/country.html>.
- [18] CNN. CNN Web Site. <http://cnn.com/>.

- [19] DOORENBOS, R., ETZIONI, O., AND WELD, D. S. A Scalable Comparison-Shopping Agent for the World-Wide Web. In *Proceedings of the First International Conference on Autonomous Agents* (Marina del Rey, California, 1998), pp. 39–48.
- [20] EMBLEY, D., JIANG, Y., AND NG, Y.-K. Record-Boundary Discovery in Web Document. In *Proceedings of the ACM SIGMOD Conference on Management of Data* (Philadelphia, Pennsylvania, 1999), pp. 271–274.
- [21] EMBLEY, D. W., CAMPBELL, D. M., JIANG, Y. S., LIDDLE, S. W., LONSDALE, D. W., NG, Y.-K., QUASS, D., AND SMITH, R. D. Conceptual-model-based data extraction from multiple-record web pages. *Data & Knowledge Engineering* 31, 3 (1999), 227–251.
- [22] EMBLEY, D. W., CAMPBELL, D. M., JIANG, Y. S., LIDDLE, S. W., NG, Y.-K., QUASS, D., AND SMITH, R. D. A Conceptual-Modeling Approach to Extracting Data from the Web. In *Conceptual Modeling - ER'98* (Berlin, 1998), T. W. Li, S. Ram, and M. Lee, Eds., Springer Verlag, pp. 78–91.
- [23] EMBLEY, D. W., CAMPBELL, D. M., LIDDLE, S. W., AND SMITH, R. D. Ontology-Based Extraction and Structuring of Information from Data-Rich Unstructured Documents. In *Proceedings of the International Conference on Information and Knowledge Management* (Bethesda, Maryland, 1998), pp. 52–59.
- [24] FLORESCU, D., LEVY, A., AND MENDELZON, A. Database Techniques for the World-Wide Web: a Survey. *SIGMOD Record* 27, 3 (1998), 59–74.
- [25] FREITAG, D. Machine learning for information extraction in informal domains. *Machine Learning* 39, 2-3 (2000), 169–202.
- [26] GRUSER, J., RASCHID, L., VIDAL, M. E., AND BRIGHT, L. Wrapper Generation for Web Accessible Data Sources. In *Proceedings of the Third International Conference on Cooperative Information Systems* (New York City, New York, 1998), pp. 14–23.
- [27] HAMMER, J., GARCIA-MOLINA, H., NESTOROV, S., YERNENI, R., BREUNIG, M., AND VASSALOS, V. Template-Based Wrappers in the TSIMMIS Experience. In *Proceedings of the ACM SIGMOD Conference on Management of Data* (Tucson, Arizona, 1997), pp. 532–535.

- [28] HAMMER, J., MCHUGH, J., AND GARCIA-MOLINA, H. Semistructured Data: The TSIMMIS Experience. In *Proceedings of the First East-European Workshop on Advances in Databases and Information Systems* (San Petersburg, Russia, 1997).
- [29] HSU, C.-N., AND CHIEN-CHI. Finite-state transducers for semi-structured text mining. In *Proceedings of IJCAI-99 Workshop on Text Mining: Foundations, Techniques and Applications* (Stockholm, Sweden, 1999).
- [30] HSU, C.-N., AND DUNG, M.-T. Generating Finite-State Transducer for Semi-Strucutred Data Extraction from the Web. *Information Systems* 23, 8 (1998), 521–538.
- [31] KASZKIEL, M., AND ZOBEL, J. Passage Retrieval Revisited. In *Proceedings of the ACM SIGIR Conference on Information Retrieval* (Philadelphia, Pennsylvania, 1997), pp. 178–185.
- [32] KUSHMERICK, N. *Wrapper Induction for Information Extraction*. PhD thesis, Department of Computer Science and Engineering, University of Washington, 1997.
- [33] KUSHMERICK, N. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence* 118, 1-2 (2000), 15–68.
- [34] LAENDER, A. H. F., RIBEIRO-NETO, B., DA SILVA, A. S., AND SILVA, E. S. Representing Web Data as Complex Objects. In *Electronic Commerce and Web Technologies*, K. Bauknecht, S. K. Mandria, and G. Pernul, Eds. Springer, Berlin, 2000, pp. 216–228.
- [35] LIBKIN, L. A Relational Algebra for Complex Objects Based on Partial Information. In *Proceedings of the Third Symposium on Mathematical Fundamentals of Database and Knowledge Systems* (Rostock, Germany, 1991), pp. 29–43.
- [36] LIU, L., PU, C., AND HAN, W. XWRAP: An XML-enabled Wrapper Construction System for Web Information Sources. In *Proceeding of the 16th International Conference on Data Engineering* (San Diego, USA, 2000), pp. 611–621.
- [37] LUO, D., AND YAO, S. B. Form Operation By Example: A Language For Office Information Processing. In *Proceedings of the 1981 ACM SIGMOD International Conference of Management of Data* (Ann Arbor, MI, 1981), pp. 212–223.

- [38] MONSTER WEB SITE. Monster.com Web Site. <http://monster.com/>.
- [39] MURDER BY THE BOOK. Murder by the Book Web Site. <http://www.neosoft.com/~mrdbrbybk>, 1999.
- [40] MUSLEA, I. A repository of online information sources used in information extraction tasks. <http://www.isi.edu/muslea/RISE/index.html>, University of Southern California, Information Sciences Institute, 1998.
- [41] MUSLEA, I. Extraction patterns for information extraction tasks: A survey. In *Proceedings of The AAAI-99 Workshop on Machine Learning for Information Extraction* (Orlando, Florida, 1999), pp. 1–6.
- [42] MUSLEA, I., MINTON, S., AND KNOBLOCK, C. An Hierarchical Approach to Wrapper Induction. In *Proceedings of the Third Annual Conference on Autonomous Agents* (Seattle, WA, 1999), pp. 190–197.
- [43] MUSLEA, I., MINTON, S., AND KNOBLOCK, C. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems* 4, 1/2 (2001), 93–114.
- [44] NCSTRL. Networked Computer Science Technical Reference Library Web Site. <http://www.ncstrl.org/>.
- [45] PAPAKONSTANTINOY, Y., GARCIA-MOLINA, H., AND WIDOM, J. Object Exchange Across Heterogeneous Information Sources. In *Proceedings of the Eleventh International Conference on Data Engineering* (Taipei, Taiwan, 1995).
- [46] RIBEIRO-NETO, B., LAENDER, A. H. F., AND DA SILVA, A. S. Extracting Semi-Structured Data Through Examples. In *Proceedings of the Eighth ACM International Conference on Information and Knowledge Management - CIKM'99* (Kansas City, MO, 1999), pp. 94–101.
- [47] RIBEIRO-NETO, B., LAENDER, A. H. F., AND DA SILVA, A. S. Top-down Extraction of Semi-Structured Data. In *Proceedings of the 6th. SPIRE String Processing and Information Retrieval Symposium - SPIRE'99* (Cancun, Mexico, 1999), pp. 184–191.

- [48] SHU, N. C., HOUSEL, B. C., AND LUM, V. Y. Convert: A high level translation definition language for data conversion. *Communications of the ACM* 18, 10 (1975), 557–567.
- [49] SODERLAND, S. Learning information extraction rules for semi-structured and free text. *Machine Learning* 34, 1-3 (1999), 233–272.
- [50] THE SABRE GROUP, INC. Travelocity Vacations Web Page.
<http://www2.travelocity.com/vacations/>.
- [51] WATCH, P. Price Wacth Web Site. <http://www.pricewatch.com/>.
- [52] ZLOOF, M. M. Query-by-Example: A Data Base Language. *IBM Systems Journal* 16, 4 (1977), 324–343.