

# Representing Web Data as Complex Objects\*

Alberto H.F. Laender, Berthier Ribeiro-Neto,  
Altigran S. da Silva<sup>†</sup>, and Elaine S. Silva

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

**Abstract.** The popularization of the Web has made a huge volume of data available for a large audience. In a large number of Web sites, such as bookstores, electronic catalogs, travel agencies, etc., the pages constitute documents which are composed of pieces of data whose overall structure can be easily recognized. Such pages are called data-rich and can be seen as collections of complex objects. In this paper, we show how such objects can be represented by nested tables, which are simple, intuitive, and quite convenient for expressing their implicit structure. The assumption is that, for most sites of interest, only few examples are required to reveal the structure of the objects. To corroborate our assumption, we describe a data extraction tool that adopts this approach and present results of some experiments carried out with this tool.

## 1 Introduction

Despite the huge volume of data made available through the Word Wide Web, manipulating such data effectively is not a simple problem. In fact, traditional database operations, such as querying, view generation, and data integration, are usually very difficult to carry out with the data available on the Web.

One of the main reasons for such a difficulty is the lack of knowledge on the structure of textual data in general. In fact, the structure associated with this type of data is usually left undeclared. However, such structure is usually present in some inherent form. For example, Web sites such as bookstores, electronic catalogs, and travel agencies include pages which are composed of pieces of data whose overall structure can be easily recognized. Such structure has not been declared anywhere but is clearly identifiable. Such pages are said to be *data rich* and *narrow in ontological breadth* [6]. Usually, the data found in them do not have a rigid structure and therefore is said to be *semistructured* [2]. Such pages are the main target of our study and, for convenience, are referred to simply as data rich pages.

---

\*This work is supported by Project SIAM (grant MCT/FINEP/PRONEX 76.97.1016.00) and by individual research grants from CNPq and CAPES.

<sup>†</sup>On leave from the University of Amazonas, Brazil.

In this paper, we propose a new approach for revealing the structure of data present in data rich Web pages. The general idea is to let the user assemble example objects taken from a Web page into a nested table according to her/his perspective and then to automatically analyze the structure of the resulting table to determine the structure of the assembled object. Nested tables [7, 10, 17] are interesting because they are simple, intuitive, and are expressive enough to represent the semistructured data normally present in common Web pages. Using this idea, we developed a tool for semistructured data extraction called *DEByE (Data Extraction By Example)* [8, 15, 16], whose graphical user interface (GUI) allows assembling an example object using simple “cut-and-paste” operations. The structure of the example object can then be used to identify new objects in the page or in other similar pages. In case of the structure of the objects vary, more than one example must be provided. The complexity and the “size” of the structural information required depend on the number of objects having distinct structures and on how distinct these structures are. We expect habitual Web users to be able to identify noticeable structural differences between objects and provide more example objects whenever necessary. We notice, however, that our aim in this paper is to present our approach for revealing the structure of Web data. Therefore, we do not address details of the extraction strategy of our tool.

Several data models have been proposed to represent semistructured data [3, 4, 14]. 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 [14]. 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 [3] 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 means of a set of trees whose leaf nodes have the actual instances associated with them. The model presented in [4] also represents data as a directed labeled graph in which each node corresponds to an object. But unlike the other two, this model is deterministic in the sense that the edges emanating from any node (that describes data) must be distinctly labeled. However to properly use these data models, we need to previously know the structure of the objects.

Many different approaches have been proposed in the literature to discover structural information from implicit objects found in data rich Web pages [6, 11, 12, 18]. The work in [11] presents a method to derive hierarchies of types and to assign objects to the derived types in a given OEM database. In [12] the authors discuss the trade-offs between having a large, but very precise, structural description of the data and having a more compact one, but that is possibly imperfect. The approach proposed in [18] aims at determining structural patterns matching the majority of objects in a OEM database, and using this information to determine a typical structure for the objects composing it. The work in [6] presents a method for the discovery of one-level structures (records) in Web pages based on HTML formatting tags. More recently, some proposals have been put forward

to address the problem of finding structural constraints in semistructured data [13].

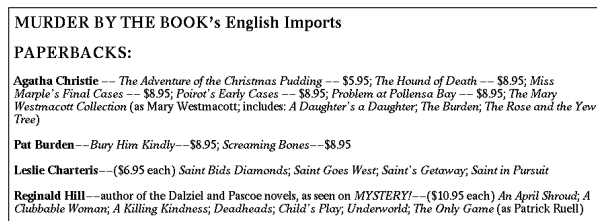
Our approach is remarkably distinct from previous works because it relies on the user’s perception of the object’s structure. That is, instead of trying to derive structural (semantic) matches from the formatting (syntax), we induce the user to inform the structure as she/he perceives it. We believe that in those situations in which the user is available to provide the information (for example, when browsing a Web site), this approach works better than alternative methods based on “blind” heuristics [11, 12, 18]. Furthermore, our approach is not tied to any specific formatting system (e.g. HTML, XML, LaTeX, etc.). Instead, it takes advantage of any type of markups surrounding the data of interest.

The remainder of the paper is organized as follows. Section 2 presents the modeling concepts adopted to describe the structure of data present in data rich Web pages. In Section 3 we discuss the notion of nested tables and its correspondence to complex objects types. The *DEByE* tool is covered in Section 4. Section 5 presents the results of an experiment carried out with the tool. Our conclusions are presented in Section 6.

## 2 Complex Objects in Web Pages

In this section, we discuss the data modeling concepts that we adopt to describe the structure of data present in data rich Web pages. These modeling concepts rely on the assumption that such pages can be seen as collections of *complex objects* which have an inherent implicit structure. In many cases, these objects are composed of sub-objects, that also have themselves an implicit structure yielding a hierarchy of objects.

Consider, for instance, the excerpt of a page from the Murder by the Book bookstore Web site (<http://www.murderbythebook.com>) shown in Fig. 1. There



**Fig. 1.** Excerpt of a page from the Murder by the Book bookstore Web site.

is an inherent structure to the text on this page. For instance, we are able to identify distinct portions of data on books by four authors. Each one of these portions can be regarded as a distinct implicit *object*. For each of these objects, we can distinguish an author name and a corresponding list of books. For the books in this list, we can identify information on book titles and prices. Thus,

there is an inherent structure associated with the objects implicitly present in the Web page of Fig. 1. Such structure has not been declared anywhere but is clearly identifiable. The implicit objects have a multi-level structure and, because of that, are said to be *complex objects*.

In what follows, we present the data modeling concepts that we use for the definition of complex object types with arbitrarily deep nesting levels. They follow the ideas described in [1] and [9], and can be seen as extensions to the relational data model.

A set of similar complex objects is described by the notion of an *object type*. An object of a specific type is called an *instance* of the type. We consider the following object types:

- *atomic type*: an object of an atomic type can only assume atomic values;
- *tuple type*: an object of a tuple type is an aggregation of other objects called its *components*;
- *list type*: an object of a list type is an ordered set of objects of the same type, called its *elements*.

To each type  $\tau$  is associated a *domain*, denoted by  $dom(\tau)$ .

We now describe more precisely the syntax and the semantics of these object types.

**Definition 1** *If  $\tau$  is an atomic type or a-type,  $dom(\tau)$  is given by the enumeration of its elements, i.e.,  $dom(\tau) = \{e_1, e_2, e_3, \dots, e_m\}$  ( $m \geq 1$ ). We assume that for every a-type  $\tau$ ,  $\lambda \in dom(\tau)$ , where  $\lambda$  denotes a null value. We also refer to a-types as **attributes**.*

Alternatively, and for the sake of simplicity, we allow for atomic types domains to be defined as the domains of usual programming languages basic types such as *int*, *float* or *string*.

**Definition 2** *Let  $\tau_1, \tau_2, \dots, \tau_n$  ( $n \geq 2$ ) be types. The notation  $\tau : (\tau_1, \tau_2, \dots, \tau_n)$  will be used to define a **tuple type** or **t-type**. The domain of a t-type  $\tau$  is defined as  $dom(\tau) = dom(\tau_1) \times dom(\tau_2) \times \dots \times dom(\tau_n)$ .*

**Definition 3** *Let  $\tau$  be a type. The notation  $\{\tau\}$  will be used to define a **list type** or **l-type**. The domain of an l-type  $\{\tau\}$  is defined as  $dom(\{\tau\}) \subseteq \{i \rightarrow dom(\tau) \mid i \in \{1, \dots, |dom(\tau)|\}\}$ .*

When dealing with data rich Web pages, semantically similar objects can present variations in their structures. To capture this semistructured [2] nature of such objects, we adopt the notion of a *variant type* [9]. Objects of a variant type are objects of any type from a list of types called the *alternatives* of the variant type. The syntax and the semantics of a variant type are defined in what follows.

**Definition 4** *Let  $\tau_1, \tau_2, \dots, \tau_n$  ( $n \geq 2$ ) be types. The notation  $\tau : [\tau_1, \tau_2, \dots, \tau_n]$  will be used to define a **variant type** or **v-type**. The domain of a v-type  $\tau$  is defined as  $dom(\tau) = dom(\tau_1) \cup dom(\tau_2) \cup \dots \cup dom(\tau_n)$ .*

An example of a complex object type composed of several sub-types is:

$$\text{Author}:[(\text{Name},\{\text{Book}:(\text{Title},\text{Price})\}), \\ (\text{Name},\text{Collection}:(\text{UnitPrice},\{\text{BookTitle}\}))] \quad (1)$$

This v-type, *Author*, has as alternatives two distinct t-types (we omit their names). The first alternative is composed by an a-type *Name* and by an l-type defined over a t-type *Book*. The second alternative is similar to the first one, but its second component is a t-type *Collection*, whose internal components differ from the t-type *Book* composing the first alternative. This v-type can be used to describe the structure of the data found in the sample page of Fig. 1. Two instances of *Author* present in this page are described in Fig. 2.

```
[Author:((Name,Agatha Christie ),
  {Book:((Title,The Adventure of the Christmas Pudding),(Price,5.95));
  Book:((Title,The Hound of Death),(Price,8.95));
  Book:((Title,Miss Marple's Final Cases),(Price,8.95));
  Book:((Title,Poirot's Early Cases),(Price,8.95));
  Book:((Title,Problem at Pollensa Bay),(Price,8.95))})]
```

```
[Author:((Name,Leslie Charteris),
  Collection: ((UnitPrice,6.95 ),
    {(BookTitle,Saint Bids Diamonds);
    (BookTitle,Saint Goes West);
    (BookTitle,Saint in Pursuit)})))]
```

**Fig. 2.** Two possible instances of *Author* found in the sample page.

### 3 Representing Complex Objects Through Nested Tables

The types defined in Section 2 are powerful enough to describe arbitrarily complex object types. At the down side, describing such types may not be a trivial task. As a result, users might find it difficult to describe the structure of objects of their interest in such a way.

Motivated by this difficulty, we propose in this section the adoption of nested tables [7, 10, 17] as a paradigm for facilitating the description of complex objects by the user. This paradigm is the base of the tool presented in Section 4.

Regarding the correspondence between nested tables and complex objects, as we shall see, every table scheme can indeed be defined in terms of complex objects, although the contrary is not necessarily true. Thus, nested tables are not as powerful as complex object types. However, they provide a simple, intuitive, and expressive enough paradigm to describe the structure of complex objects normally found in the Web. For this matter, we extend the usual notion of nested tables to allow for variations in their structure.

Intuitively, a table can be defined as an l-type (list type) object defined over a t-type (tuple type) object. For “flat” tables (i.e., pure relational tables)

each t-type object component is an a-type (attribute type) object. Multi-valued attributes are handled by defining t-type object components as l-type objects defined over a-type objects. To represent nested tables, one can recursively define t-type object components as l-types over t-type objects.

Consider the type defined in (1). We can define an l-type

$$\{\text{Author}:(\text{Name},\{\text{Book}:(\text{Title},\text{Price})\})\} \quad (2)$$

over it to “store” data on many author entries having a structure similar to this. This type corresponds to a table named **Author**, where the values of one of its columns, **Book**, are themselves tables. Fig. 3a illustrates such a table containing two instances of this type.

Author													
Name	Book												
Agatha Christie	<table border="1"> <thead> <tr> <th>Title</th> <th>Price</th> </tr> </thead> <tbody> <tr> <td><i>The Adventure of the ...</i></td> <td>5.95</td> </tr> <tr> <td><i>The Hound of Death</i></td> <td>8.95</td> </tr> <tr> <td><i>Miss Marple's Final Cases</i></td> <td>8.95</td> </tr> <tr> <td><i>Poirot's Early Cases</i></td> <td>8.95</td> </tr> <tr> <td><i>Problem at Pollensa Bay</i></td> <td>8.95</td> </tr> </tbody> </table>	Title	Price	<i>The Adventure of the ...</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
	Title	Price											
	<i>The Adventure of the ...</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												
Pat Burden	<table border="1"> <thead> <tr> <th>Title</th> <th>Price</th> </tr> </thead> <tbody> <tr> <td><i>Bury Him Kindly</i></td> <td>8.95</td> </tr> <tr> <td><i>Screaming Bones</i></td> <td>8.95</td> </tr> </tbody> </table>	Title	Price	<i>Bury Him Kindly</i>	8.95	<i>Screaming Bones</i>	8.95						
	Title	Price											
	<i>Bury Him Kindly</i>	8.95											
<i>Screaming Bones</i>	8.95												

(a)

Author													
Name	Book												
Agatha Christie	<table border="1"> <thead> <tr> <th>Title</th> <th>Price</th> </tr> </thead> <tbody> <tr> <td><i>The Adventure of the ...</i></td> <td>5.95</td> </tr> <tr> <td><i>The Hound of Death</i></td> <td>8.95</td> </tr> <tr> <td><i>Miss Marple's Final Cases</i></td> <td>8.95</td> </tr> <tr> <td><i>Poirot's Early Cases</i></td> <td>8.95</td> </tr> <tr> <td><i>Problem at Pollensa Bay</i></td> <td>8.95</td> </tr> </tbody> </table>	Title	Price	<i>The Adventure of the ...</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
	Title	Price											
	<i>The Adventure of the ...</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												
Leslie Charteris	<table border="1"> <thead> <tr> <th>UnitPrice</th> <th>Title</th> </tr> </thead> <tbody> <tr> <td>6.95</td> <td><i>Saint Bids Diamonds</i></td> </tr> <tr> <td></td> <td><i>Saint Goes West</i></td> </tr> <tr> <td></td> <td><i>Saint in Pursuit</i></td> </tr> </tbody> </table>	UnitPrice	Title	6.95	<i>Saint Bids Diamonds</i>		<i>Saint Goes West</i>		<i>Saint in Pursuit</i>				
	UnitPrice	Title											
	6.95	<i>Saint Bids Diamonds</i>											
	<i>Saint Goes West</i>												
	<i>Saint in Pursuit</i>												

(b)

**Fig. 3.** Nested tables representing instances of **Author** according to (2) and (3).

A table where rows may have many alternative structures can be defined by using an l-type defined over a v-type, provided that each alternative t-type of this v-type has the same components. The components of the alternative t-type, however, may have different structures. For example, the table illustrated in Fig. 3b represents an instance of the following l-type:

$$\{\text{Author}:[(\text{Name},\{\text{Book}:(\text{Title},\text{Price})\}), (\text{Name},\{\text{Book}:(\text{UnitPrice},\{\text{Book Title})\})\})]\} \quad (3)$$

This l-type was built over a modified version of the v-type defined in (1), in which the second component of the second alternative of **Author** is now a list over a different t-type **Book**.

From our discussion so far, it should be clear that not all complex object types defined in Section 2 can be represented by nested tables. This is the case of the v-type defined in (1). To precisely characterize this we define the notion of a *table scheme*, as follows.

**Definition 5** *An l-type defined over a v-type  $\{\tau : [\tau_1, \tau_2, \dots, \tau_n]\}$  ( $n > 0$ ) is a **table scheme** only if all of its alternative types  $\tau_i$  ( $0 < i \leq n$ ) are of the form*

$\tau_i = (\gamma_1, \gamma_2, \dots, \gamma_m)$  ( $m > 0$ ), where each  $\gamma_j$  ( $j \geq 0$ ) is an *a-type*, an *l-type* defined over an *a-type* or a table scheme.

Given the definition of a table scheme, we can now define a table as follows.

**Definition 6** A *table* is defined by using the notation  $\mathbb{T}\{\tau : [\tau_1, \tau_2, \dots, \tau_n]\}$ , where  $\mathbb{T}$  is the table name and  $\{\tau : [\tau_1, \tau_2, \dots, \tau_n]\}$  is its scheme. Further, we define an *instance* of a table  $\mathbb{T}$  as a subset of  $\text{dom}(\tau)$ .

One way of defining complex object types that can always be derived from the structure of a nested table is to use pre-defined operations for manipulating table schemes. These operations are such that, applying any sequence of them, we will always have as a result a complex object type that satisfies Definition 5.

There are six operations that can be used for table scheme manipulation: *add-to-v-type*, *add-to-t-type*, *group-components*, *remove-from-v-type*, *remove-from-t-type* and *ungroup-components*. These operations always modify a given table scheme by adding or removing types that compose it. The *add-to-v-type* operation adds a new t-type to the list of t-types that compose a given v-type, i.e., it adds a new alternative t-type to the v-type, observing the conditions of Definition 5. The *add-to-t-type* operation adds a new component to each t-type that composes a given v-type, i.e., it adds a new column in the table. The *group-components* operation adds a nested table scheme as a new component of every t-type  $\tau$  that composes a given table scheme. Fig. 4(b) shows the result of applying the *group-components* operation to the table in Fig. 4(a). The *remove-from-v-*

Author		
Name	Title	Price
Agatha Christie	The Hound of Death	8.95
Leslie Charteris	Saint Bids Diamonds	6.95

(a)

Author		
Name	Book	
	Title	Price
Agatha Christie	The Hound of Death	8.95
Leslie Charteris	Saint Bids Diamonds	6.95

(b)

**Fig. 4.** Table `Author` before (a) and after (b) the *group-components* operation.

*type*, *remove-from-t-type*, and *ungroup-components* operations are, respectively, symmetric to the *add-to-v-type*, *add-to-t-type* and *group-components* operations. A more precise discussion of these operations are out of the scope of this paper and can be found in [5].

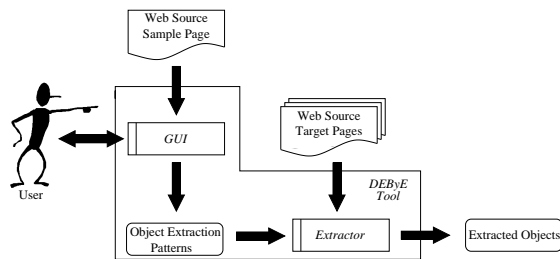
## 4 The DEByE Tool

The graphical user interface (GUI) of our semistructured data extraction tool, *DEByE (Data Extraction By Example)\** [8, 16], adopts the nested table paradigm,

\* The name of our tool is an homage to Moshé Zloof, creator of the QBE (Query By Example) language, who suggested us the example-based approach we use to specify the data to be extracted.

as discussed in Section 3 to describe the structure of complex objects found in data rich Web pages. By using this GUI, users can easily assemble example of complex objects using, so that their implicit structure, as perceived by them, can be determined. For such data rich pages, it is expected that just a couple of different examples are required to determine the structure of most objects in the page or even in other similar pages (e.g., pages of a same site).

Once this structural information is available, the tool adds to it information on the textual context of the example objects to build data structures called *Object Extraction (OE) patterns*. The OE patterns can then be fed to an *Extractor* module that uses them to identify and to extract new objects whose structure and textual context are similar to the example objects provided. Fig. 5 presents an overview of the major modules which compose the *DEByE* tool and their role in the data extraction process. The details of the *Extractor* module are out of



**Fig. 5.** Modules of the DEByE tool and their role in a data extraction process.

the scope of this paper and can be found in [5, 8, 15]. In this section we focus our discussion on the use of the *DEByE* GUI to assemble proper example objects.

Fig. 6 presents a snapshot of the main screen provided by the GUI to the user. The screen includes three main windows: the *Source Window*, the *Table Window*, and the *Pattern Builder Window*. The *Source Window* displays a Web page selected by the user. In this case, the page is from the Murder by the Book bookstore Web site which we have been using throughout our discussion. The *Table Window* is used to assemble the nested table that describes the structure of the example objects. The *Pattern Builder Window* is used by the user to trigger the generation of the OE patterns for the example object given.

Formally, in the *Table Window* each row corresponds to an alternative t-type that composes a v-type which is defined inside an l-type that corresponds to the outermost table, in a way similar to the l-type defined in (3). The names of the types composing these t-types appear as the heading of the table columns. This forces every row to have the same attributes, according to our definition of table scheme (Definition 5). The same is true for each nested table.

To assemble an example object, the user must mark pieces of data in the text in the *Source Window* and copy them into the columns of the *Table Window*. This is done separately for each piece of data. To allow the user to specify the (nested) table which embeds the structure of her/his example object, the tool

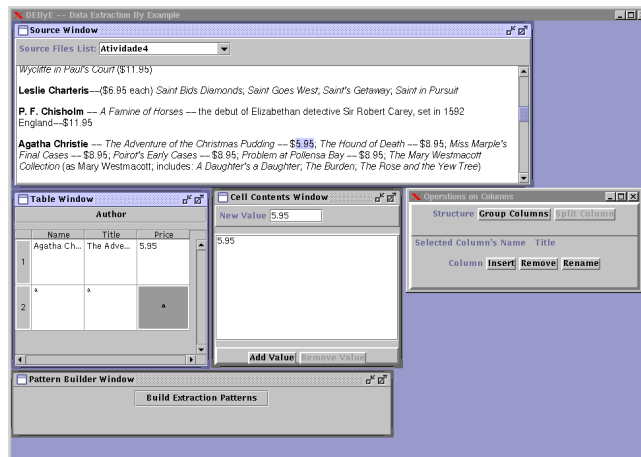


Fig. 6. DEByE screen with a partially specified example.

provides the column operations *insert*, *remove*, *group*, and *split*, which compose the *Operations on Columns* window as illustrated in Fig. 6. These operations correspond, respectively, to the operations *add-to-t-type*, *remove-from-t-type*, *group-components*, and *ungroup-components* described in Section 3. An additional *rename* column operation is provided for convenience to allow the user to rename a column. For instance, in Fig. 6, the user has executed the *insert* column operation to add a new column (Price) and the *rename* column operation to assign the labels Name, Title, and Price to the three columns. The other two operations are discussed in the immediately following.

In Fig. 7, the user has advanced in her/his example specification task. Having observed that *Agatha Christie* has written many books, she/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*. To create this nested table, the user applies the *group* operation on the attributes Title and Price, and the *rename* operation to assign the label *Book* to it.

For pages with non-homogeneous objects (as the pages of the Murder by the Book site), the user might have to specify more than one example object to cover a larger fraction of the (implicit) objects in the page. For instance, in Fig. 7 the user specified a second example object to indicate that, in the case of the author *Leslie Charteris*, the price of all her books is the same. This is indicated by assembling the book titles written by *Leslie Charteris* in a list. This is done by an *insert* row operation, which corresponds to the operation *add-to-v-type* and results in the inclusion of a new row into the (nested) table. Likewise, there is also a *remove* row operation, which corresponds to the *remove-from-v-type* operation. We stress that each row in a (nested) table corresponds in fact to an alternative t-type of a v-type.

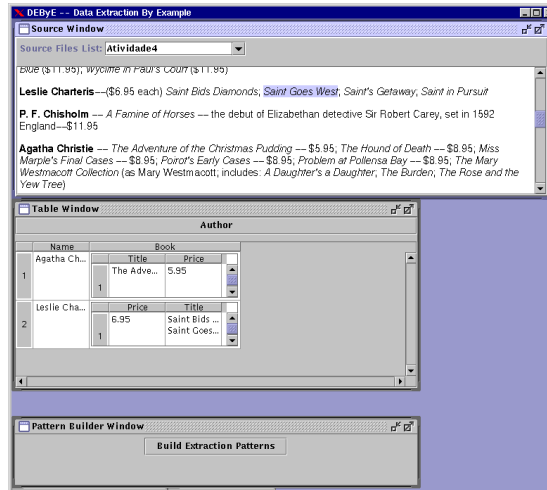


Fig. 7. Specification of two distinct example objects for a sample page.

## 5 Experimental Results

In this section, we present the results of some experiments we carried out using the DEByE tool. The goal was to investigate how useful is the DEByE GUI in assisting the user with the specification of example objects. In the experiment sixteen undergraduate Computer Science students used the DEByE GUI to specify example objects for three Web sites: CDNow (<http://www.cdnw.com>), ACM TODS from DB&LP (<http://www.informatik.uni-trier.de/~ley/db>), and Murder by the Book (<http://www.murderbythebook.com>). Fig. 8 illustrates excerpts of pages from these three sites.

According to an increasing order of complexity, the users were first presented with a page from CDNow, followed by a page from DB&LP and, at the end, with a page from Murder by the Book. In each case, each user was asked to specify a set of example objects (in a number of her/his choosing). Each set of example objects yielded a separate set of OE patterns. We notice that the (implicit) objects in the CDNow pages are flat (i.e., their structure is given by a one-level hierarchy), while the (implicit) objects in the DB&LP TODS and Murder by the Book pages can be interpreted as a two-level hierarchical structure. Further, the objects in the DB&LP TODS pages are homogeneous, while the objects in the Murder by the Book pages present structural variations.

To evaluate the user example objects specified by our 16 users, we compared each example object with all others and separated them in 3 groups. The first group is called *complete* and is composed of objects that have all a same structure and whose structure is the most complex (among all the example objects specified), according to the perception of the users. The second group is called *partial* and is composed of objects that have a structure which is only a part of the structure of the objects in the first group. The third group is called *incorrect*

Placebo <a href="#">Without You I'm Nothing</a>	\$16.97	<b>\$11.88</b>
Portishead <a href="#">Pnyc</a>	\$16.97	<b>\$11.88</b>
Louis Prima <a href="#">Collectors Series</a>	\$11.97	<b>\$8.38</b>
Queen <a href="#">Greatest Hits I &amp; II</a>	\$29.97	<b>\$20.98</b>
R.E.M. <a href="#">Up</a>	\$16.97	<b>\$11.88</b>

(a) CDNow

<p>Volume 19, Number 1, March 1994</p> <ul style="list-style-type: none"> <li>• Won Kim: Charter and Scope. 1–2</li> <li>• Martin S. Oliver, Sebastian H. von Solms: A Taxonomy for Secure Object-Oriented Databases. 3–46, <i>Electronic Edition</i> (<a href="#">link</a>)</li> <li>• Patrick Tendick, Norman S. Matloff: A Modified Random Perturbation Method for Database Security. 47–63, <i>Electronic Edition</i> (<a href="#">link</a>)</li> <li>• James Clifford, Albert Crocker: On Completeness of Historical Relational Query Languages. 64–116, <i>Electronic Edition</i> (<a href="#">link</a>)</li> <li>• Kenneth Salem, Hector Garcia-Molina, Jeannie Shands: <b>Altruistic Locking</b>. 117–165, <i>Electronic Edition</i> (<a href="#">link</a>)</li> </ul>
--

(b) DB&LP TODS

<p><b>Agatha Christie</b> — <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><b>Dorothy Dunnnett</b> — <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><b>Dick Francis</b> — <i>The Sport of Queens</i> (autobiography) — \$10.95</p>
---

(c) Murder by the Book

Fig. 8. Excerpts of pages from three Web sites used in our experiments.

and is composed of all the objects that have a structure which does not seem to make sense. The results of our 16 user sessions are summarized in Table 1.

Web Site	User Specified Example Objects			Percentage of Objects Extracted
	Complete	Partial	Incorrect	
CDNow	15	-	1	100%
DB&LP ACM TODS	16	-	-	92%
Murder by the Book	9	3	4	89%

Table 1. Summary of the results of our experiments.

As can be observed, the users were very effective in providing examples for the CDNow and DB&LP Web sites. Surprisingly, one user miss-interpreted the CDNow page which is the least complex one. Not surprisingly, the users found it more difficult to deal with the Murder by the Book Web site. Even though, the majority of the users was able to successfully specify the example objects properly. Table 1 also presents the percentage of objects extracted from a set of pages of each Web site when we submitted complete examples to the DEByE *Extractor* module. As we can see, the tool was very effective and extracted a high percentage of the existing objects in these pages. The results of a larger set of extraction experiments on pages of several popular Web sites are reported in [8, 15].

## 6 Conclusions

In this paper, we proposed a new approach for revealing the structure of data present in data rich Web pages. This approach relies on the assumption that such pages can be seen as collections of similar complex objects and that the user can give her/his interpretation of the data structure by providing examples of such objects.

To be able to create abstractions for collections of similar objects, we defined a set of object types, including a special type, called *v-type*, which allows us to capture the semistructured nature of the data involved. The types defined are powerful enough to describe very complex data structures. Here, however, we discussed the use of nested tables as a form of representation for such structures. Nested tables are not as powerful as the defined object types, but they have the advantage of being simple, intuitive, and quite convenient for expressing multi-level hierarchical structures [10]. We also showed that every table scheme corresponds to a complex object type, although the contrary is not necessarily true.

Based on the above ideas, we implemented a semistructured data extraction tool, called *DEByE*, whose GUI allows the assembling of example objects into a nested table by using simple “cut-and-paste” operations. The structure of the example object can then be used to identify new objects in the page or in other similar pages. As we discuss in [8, 15], this approach is very effective and usually only a couple of examples is sufficient to extract a high percentage of the objects existing in the pages.

The approach we presented is remarkably distinct from previous works because it relies on the user’s perception of the object’s structure. That is, instead of trying to derive structural (semantic) matches from the formatting (syntax), we induce the user to inform the structure as she/he perceives it. We believe that in those situations in which the user is available to provide the information (for example, when browsing a Web site), this approach works better than alternative methods based on “blind” heuristics. Furthermore, our approach is not tied to any specific formatting system (e.g. HTML, XML, LaTeX, etc.). Instead, it takes advantage of any type of markups surrounding the data of interest.

## References

- [1] ABITEBOUL, S., HULL, R., AND VIANU, V. *Foundations of Databases*. Addison-Wesley, Reading, Massachusetts, 1995.
- [2] BUNEMAN, P. Semistructured Data. In *Proceedings of the Sixteenth ACM SIGMOD Symposium on Principles of Database Systems* (Tucson, Arizona, 1997), pp. 117–121.
- [3] BUNEMAN, P., DAVIDSON, S., HILLEBRAND, G., AND SUCIU, D. A Query Language and Optimization Techniques for Unstructured Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Quebec, Canada, 1996), pp. 505–516.

- [4] BUNEMAN, P., DEUTSCH, A., AND TAN, W. A Deterministic Model for Semistructured Data. In *Proceedings of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats* (Jerusalem, Israel, 1999).
- [5] DA SILVA, A. S. *Example-based Extraction and Integration of Semi-Structured Data*. Ph.D. Thesis Proposal, Department of Computer Science, Federal University of Minas Gerais, Belo Horizonte, Brazil, 2000. In preparation.
- [6] EMBLEY, D. W., CAMPBELL, D. M., JIANG, Y. S., LIDDLE, S. W., NG, Y.-K., QUASS, D., AND SMITH, R. D. Conceptual-model-based data extraction. *Data & Knowledge Engineering* 31, 3 (1999), 227–251.
- [7] JAESCHKE, G., AND SCHEK, H.-J. Remarks on the algebra of non first normal form relations. In *Proceedings of the ACM Symposium on Principles of Database Systems* (Los Angeles, California, 1982), ACM, pp. 124–138.
- [8] LAENDER, A. H. F., RIBEIRO-NETO, B., AND DA SILVA, A. S. DEByE – Data Extraction By Example. Technical Report, Department of Computer Science, Federal University of Minas Gerais, Belo Horizonte, Brazil, 2000.
- [9] 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.
- [10] LORENTZOS, N. A., AND DONDIS, K. A. Query by Example for Nested Tables. In *Proceedings of the 9th International Conference in Database and Experts Systems Applications* (Vienna, Austria, 1998), pp. 716–725.
- [11] NESTOROV, S., ABITEBOUL, S., AND MOTWANI, R. Inferring Structure in Semistructured Data. *SIGMOD Record* 26, 4 (1997), 39–43.
- [12] NESTOROV, S., ABITEBOUL, S., AND MOTWANI, R. Extracting Schema from Semistructured Data. In *Proceedings of the ACM SIGMOD Conference on Management of Data* (Seattle, Washington, 1998), pp. 256–306.
- [13] P. BUNEMAN AND W. FAN AND S. WEINSTEIN. Interaction between Path and Type Constraints. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)* (Philadelphia, Pennsylvania, 1999), pp. 56–67.
- [14] 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).
- [15] 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, Missouri, 1999), pp. 94–101.
- [16] SILVA, E. S. Example-Based Semi-Structured Data Extraction. Master's Thesis, Department of Computer Science, Federal University of Minas Gerais, Belo Horizonte, Brazil, 1999. In Portuguese.
- [17] VAN GUCHT, D., AND FISCHER, P. C. Multilevel nested relational structures. *Journal of Computer and System Sciences* 36, 1 (1988), 77–105.
- [18] WANG, K., AND LIU, H. Schema Discovery for Semistructured Data. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97)* (Newport Beach, California, 1997), pp. 271–274.