Available online at www.sciencedirect.com
SCIENCE DIRECT®
Data & Knowledge Engineering 54 (2005) 3–28
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

Automating the extraction of data from HTML tables with unknown structure

David W. Embley [a,*], Cui Tao [a], Stephen W. Liddle [b]

[a] *Department of Computer Science, Brigham Young University, Provo, UT 84602, USA*
[b] *Information Systems Group and Rollins eBusiness Center, Brigham Young University, Provo, UT 84602, USA*

Available online 20 November 2004

Abstract

Data on the Web in HTML tables is mostly structured, but we usually do not know the structure in advance. Thus, we cannot directly query for data of interest. We propose a solution to this problem based on document-independent extraction ontologies. Our solution entails elements of table understanding, data integration, and wrapper creation. Table understanding allows us to find tables of interest within a Web page, recognize attributes and values within the table, pair attributes with values, and form records. Data-integration techniques allow us to match source records with a target schema. Ontologically specified wrappers allow us to extract data from source records into a target schema. Experimental results show that we can successfully locate data of interest in tables and map the data from source HTML tables with unknown structure to a given target database schema. We can thus "directly" query source data with unknown structure through a known target schema.

Keywords: Document-independent extraction ontologies; Table understanding; Data integration; Wrapper creation

* Corresponding author. Tel.: +1 801 422 6470; fax: +1 801 422 0169.
E-mail addresses: embley@cs.byu.edu (D.W. Embley), ctao@cs.byu.edu (C. Tao), liddle@byu.edu (S.W. Liddle).

0169-023X/$ - see front matter
doi:10.1016/j.datak.2004.10.004

1. Introduction

The schema-mapping problem for heterogeneous data integration is hard and is worthy of study [22]. The problem is to find a *semantic correspondence* between one or more *source schemas* and a *target schema* [6]. In its simplest form the semantic correspondence is a set of *mapping elements*, each of which binds an attribute in a source schema to an attribute in a target schema or binds a relationship among attributes in a source schema to a relationship among attributes in a target schema. Such simplicity, however, is rarely sufficient, and researchers thus use queries over source schemas to form attributes and relationships among attributes to bind with target attributes and attribute relationships [23,2]. Furthermore, as we shall see in this paper, we may also need queries beyond those normally defined for database systems. Thus, we more generally define the *semantic correspondence for a target attribute* as any named or unnamed set of values that is constructed from source elements, and we define the *semantic correspondence for a target n-ary relationship* among attributes as any named or unnamed set of n-tuples over constructed value sets. Sets of values for target attributes may be obtained from source elements in any way, e.g. directly taken from already present source values, computed over source values, constructed by concatenation or decomposition from source values, or directly taken or manufactured from source attribute names, from strings in table headers or footers, or from free text surrounding tables.

We limit our discussion here to HTML tables found on the Web. [1] We consider Web pages containing HTML tables of interest for a given application domain to be our sources. We also include pages linked from within these HTML tables. Our target is a simple relational schema.

As a running example, we use car advertisements, which are plentiful on the Web and which often present their information in tables. Suppose, for example, that we are interested in viewing and querying Web car-ads through the target database in Fig. 1, whose schema is

$$\{Car, Year, Make, Model, Mileage, Price, PhoneNr\}$$
$$\{Car, Feature\}.$$

Figs. 2 [3], 3 [3], and 4 [1] show some potential source tables. The data in the tables in Fig. 1 is a small part of the data that can be extracted from Figs. 2–4.

1.1. HTML tables—location problems

It is easy for a human to locate the table of interest in Fig. 2. Algorithmically finding the table of interest on an Web page, however, is often non-trivial, even when the system can tell that the page is of interest for the given application [10]. Fig. 2, for example, presents several challenges for table location.

- *Multiple panes*. The page in Fig. 2 has three panes (HTML frames), but we are only interested in the one starting with *Pre-Owned Inventory*.

[1] The problems encountered in HTML tables are more than sufficient for this investigation. Table extraction within the broader context of images of paper tables and other types of electronic tables [19] is also possible.

Car	Year	Make	Model	Mileage	Price	PhoneNr
0001	1999	Pontiac	Firebird	32,833		405-936-8666
0002	2000	Acura	RL 3.5	36,657	$23,988	405-936-8666
0003	2002	Honda	Accord EX	13.875	$21,988	405-936-8666
...						
0101	1992	ACURA	legend		$9500	
0102	2000	AUDI	A4		$34,500	
0103	1985	BMW	325e		$2700.00	

Car	Feature
0001	Blue
0001	...
...	...
...	
0003	White
0003	Air Conditioning
0003	Driver Side Air Bag
...	
0101	Auto
0101	AM/FM
...	

Fig. 1. Sample tables for target schema.

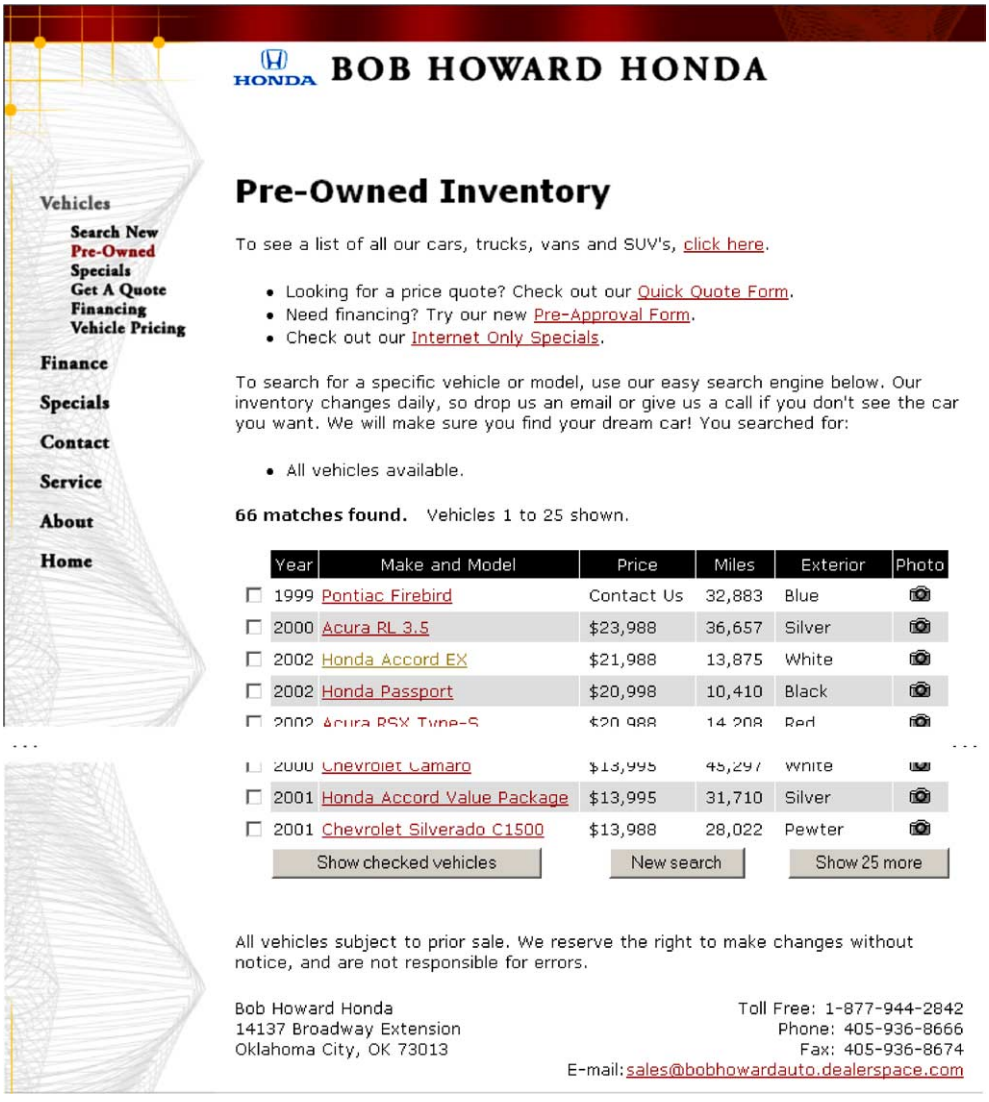


Fig. 2. Web page with table from [3].

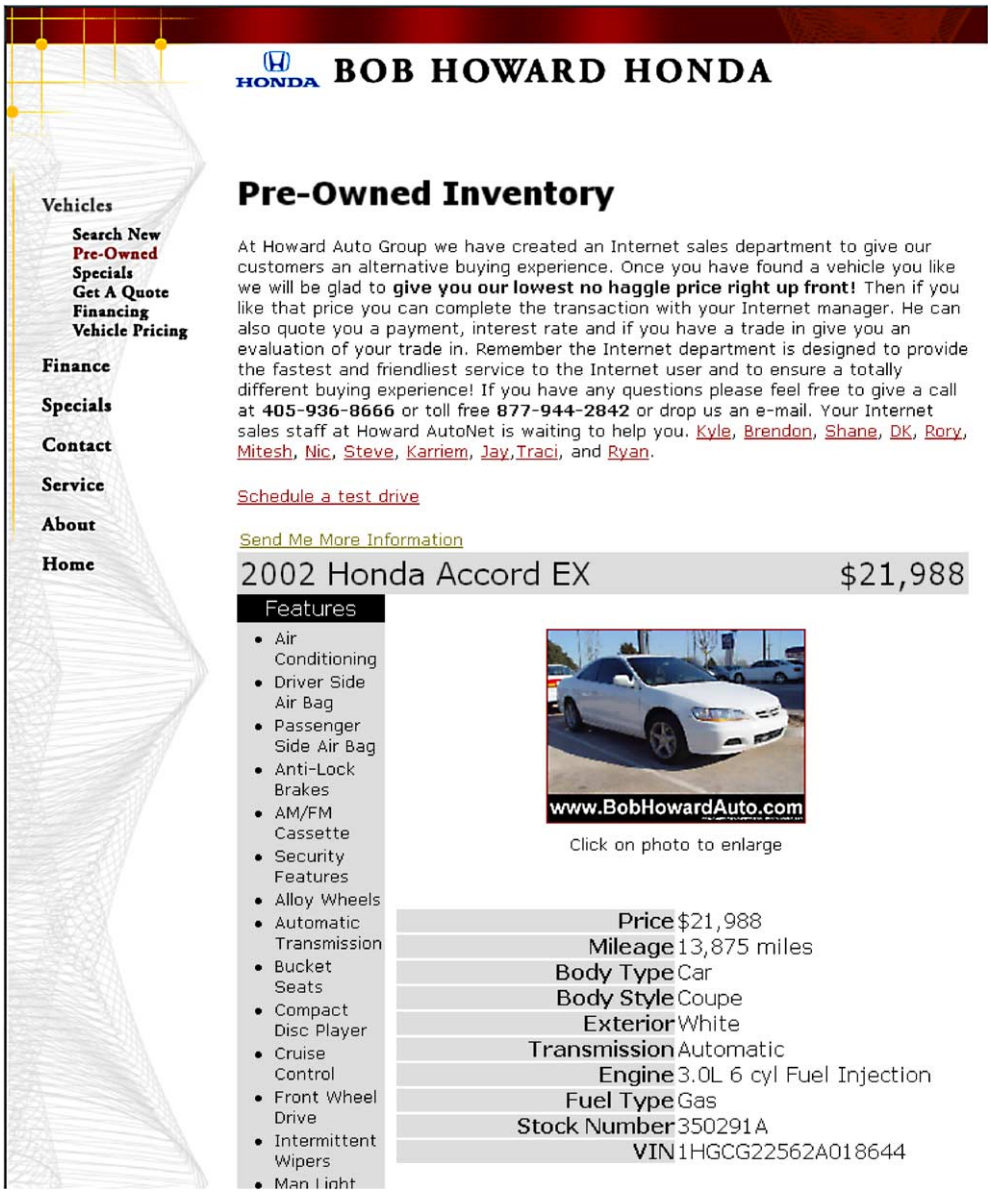


Fig. 3. Linked Page with additional information [3].

- *Tables for layout*. In the pane of interest, the first ⟨*table*⟩ tag encountered has two lines: the first for the text above the table, and the second for the table and the footer text below the table.
- *Table rows not in table*. The last two lines of the table are not actually part of the table. The last line contains the contact information, and the next-to-last line contains the buttons and the claim of non-responsibility.

Make	Model	Year	Colour	Price	Auto	Air Cond.	AM/FM	CD
ACURA	legend	1992	grey	$9500	Yes	No	Yes	No
AUDI	A4	2000	Blue	$34,500	Yes	Yes	Yes	Yes
BMW	325e	1985	black	$2700.00	No	No	Yes	No
CHEVROLET	Cavalier Z24	1997	Black	$11,995.00	No	Yes	Yes	No

Fig. 4. Table from [1].

- *Tables displayed piecemeal*. The table in Fig. 2 displays 25 rows per page. To obtain the rest of the table rows, we need to have the system simulate a click on *Show 25 more*.
- *Tables spanning multiple pages*. We obtain the page in Fig. 3 by clicking on *Honda Accord EX* in the table in Fig. 2. Clicking on all makes and models gives us similar pages. Each page has a column of attribute–value pairs that starts with *Price* and ends with *VIN*. The collection of these columns from each page constitutes a large table whose attributes are all the same, *Price ... VIN*, and whose values are the value columns from each linked page.
- *No ⟨Table⟩ Tag*. Each linked page similar to the one in Fig. 3 also has a single-column table headed by *Features*. The source, however, does not tag this table with a ⟨*table*⟩ tag, but rather with a ⟨*ul*⟩ tag, making it an HTML list.

In general there are even more challenges for locating tables. We have listed here only the challenges that appear in Figs. 2 and 3. We list other challenges in the discussion of our implementation status and future work in Section 6.

1.2. HTML tables—extraction problems

Not only is it easy for a human to find the tables of interest in Figs. 2 and 3, it is also easy for a human to parse the table and determine its meaning, independent of any particular view. Even with the constraint imposed of needing to match a source table with respect to a fixed target view, such as the one in Fig. 1, semantic matching is mostly straightforward for a human. It is easy to see that *Year* in the source table in Fig. 2 as well as *Year* in the source table in Fig. 4 map to *Year* in the target table in Fig. 1. It is also easy to see that although *Make* and *Model* in Fig. 4 match directly with *Make* and *Model* in Fig. 1, we need to split *Make and Model* in Fig. 2 to match *Make* and *Model* in Fig. 1. It is not as easy, however, to see that both *Exterior* in Fig. 2 and *Colour* in Fig. 4 map to *Feature* in Fig. 1, and it is a little harder to see that we should map the attributes *Auto*, *Air Cond.*, *AM/FM*, and *CD* in Fig. 4 as values for *Feature* in Fig. 1, but only for "Yes" values.

Algorithmically sorting out these semantic matches is significantly harder. We encounter the following list the challenges when trying to match source HTML tables in Figs. 2–4 with the target schema in Fig. 1. We list other challenges in the discussion of our implementation status and future work in Section 6.

- *Merged attributes/values*. *Make* and *Model* are separate attributes in Fig. 1 but are merged as one attribute in Fig. 2.

- *Subsets*. *Exterior* in Fig. 2 and *Colour* in Fig. 4 contain colors. Colors in the target are a special kind of *Feature* and thus the sets of colors in Figs. 2 and 4 are subsets of the feature values we want for Fig. 1. Indeed, these are proper subsets since there are also many other feature values in Figs. 2–4.
- *Synonyms*. *Mileage* in Fig. 1 and *Miles* in Fig. 2 have the same meaning, but the attribute names are not the same.
- *Extra information*. The tables in Fig. 1 make no request for photographs, which are present in Fig. 2.
- *Linked information*. The values for the attribute *Make and Model* are linked to further information. Clicking on *Honda Accord EX* in Fig. 2 yields the information in Fig. 3.
- *List table*. A one-dimensional table and a list are similar in appearance. *Features* in Fig. 3 is a list, but could just as easily have been formatted as a table. Although it is a list, we nevertheless wish to match *Features* in Fig. 3 with *Feature* in Fig. 1.
- *Position of attributes*. The linked subtable in Fig. 3 has its attributes in the left column, rather than in the top row.
- *Missing information*. The schema in Fig. 1 expects a phone number, but none of the tables in Figs. 2, 3, or 4 contains a phone number.
- *Externally factored data*. Although no phone number appears in the tables in Figs. 2 or 3, phone numbers do appear in the footer text of the table in Fig. 2 and in the text above the tables in Fig. 3. A value, such as a dealer phone number, that applies to all records in a table is often factored out, external to the table, and displayed only once.
- *Duplicate data*. The price for the *Honda Accord EX* in Figs. 2 and 3 appears three times, once under *Price* in Fig. 2, once as the value for the *Price* attribute in the (vertical) table row in Fig. 3, and once at the top of the layout table in Fig. 3. (Luckily, the values are all the same.) Other values also appear more than once. The number of miles, in fact, appears with two different attributes, once with *Miles* and once with *Mileage*.
- *Unexpected multiple values*. The schema in Fig. 1 expects at most one contact phone number for each vehicle, but there may be several as Figs. 2 and 3 show.
- *Attribute as value*. In Fig. 4, the features *Auto*, *Air Cond.*, *AM/FM*, and *CD* are all attributes rather than values. Here, we must understand that *Yes* and *No* are not the values; rather they indicate whether the values *Auto*, *Air Cond.*, *AM/FM*, and *CD* should be included as *Feature* values in the tables in Fig. 1.

1.3. HTML tables—location and extraction solutions

Rather than directly try to locate tables in ways similar to those reported in [19] or directly try to find mappings from source schemas to target schemas as suggested in [23,6,22,2], our contribution in this paper is to argue for a different approach, show that it works, and explain why it may be superior. The approach requires aspects of table understanding [19], but it especially relies on extraction ontologies [8]. Our approach includes five steps.

1. *Locate table of interest*. Based mostly on recognizing expected attribute names and values specified in our ontology, we find the tables of interest. We determine, for example, that the table containing a row for each vehicle in Fig. 2 begins with the row of attributes *Year*,

Make and Model, ... and ends with the row containing the *2001 Chevrolet Silverado C1500*. We also determine that the table continues on additional pages (*Show 25 more*), and that each vehicle has subtables, *Features* which is formatted as a list, and the table of attribute–value pairs starting with *Price $21,988* that spans every page linked to the top-level table.

2. *Form attribute–value pairs*. Given individual strings recognized by our ontology as potential attributes and values, we use table understanding techniques to form attribute–value pairs. We determine, for example, that ⟨*Year*: 1999⟩ and ⟨*Exterior*: *Blue*⟩ are two of the attribute–value pairs for the first record in Fig. 2.
3. *Adjust attribute–value pairs*. We convert, for example, the recognized attribute–value pair ⟨*CD*: *Yes*⟩ in row two of Fig. 4 to *CD*, meaning that this car has a CD player, and the the pair ⟨*CD*: *No*⟩ in rows one, three, and four to null strings, meaning that these cars do not have CD players.
4. *Analyze extraction patterns*. Given the table layout, links to subtables, and location of text surrounding tables, we look for specific patterns of interest. The extraction ontology recognizes, for example, that *32,833* in the first row in Fig. 2 should be extracted as the *Mileage* for the first car in Fig. 1 and that the first part of the value *Pontiac Firebird* should be extracted as the *Make* while the second part should be extracted as the *Model*.
5. *Infer mappings*. Given the recognized extraction patterns, the system can infer a general mapping from source to target. Based on the extraction examples above, the system would know, for example, that the *Miles* values in Fig. 2 map to *Mileage* in the target (Fig. 1), that the first part of the *Make and Model* strings map to *Make*, and that the second part maps to *Model*.

We present the details of our contribution in the remainder of the paper as follows. Section 2 describes extraction ontologies. Section 3 explains how we locate tables and preprocess them into a standard form. Section 4 provides the details about how we form and adjust attribute–value pairs and then analyze them and infer mappings. In Section 5, we report the results of experiments we conducted involving car advertisements and cell-phone sales, which we found on the Web. Section 6 reports on the status of our implementation, saying what we have implemented, what we plan to implement, and what is potentially of interest but outside the scope of the current project. We summarize in Section 7.

2. Extraction ontologies

An extraction ontology is a conceptual-model instance that serves as a wrapper for a narrow domain of interest such as car-ads [8]. The conceptual-model instance includes objects, relationships, constraints over these objects and relationships, descriptions of strings for lexical objects, and keywords denoting the presence of objects and relationships among objects. When we apply an extraction ontology to a Web page, the ontology identifies the objects and relationships and associates them with named object sets and relationship sets in the ontology's conceptual-model instance and thus wraps the recognized strings on a page and makes them "understandable" in terms of the schema implicitly specified in the conceptual-model instance. The hard part of writing a wrapper for extraction is to make it robust so that it works for all sites, including sites not in

```
 1.  Car [-> object];
 2.  Car [0:1] has Year [1:*];
 3.  Car [0:1] has Make [1:*];
 4.  Car [0:1] has Model [1:*];
 5.  Car [0:1] has Mileage [1:*];
 6.  Car [0:*] has Feature [1:*];
 7.  Car [0:1] has Price [1:*];
 8.  PhoneNr [1:*] is for Car [0:1];
 9.  Year matches [4]
10.      constant {extract "\d{2}";
11.                context "\b'[4-9]\d\b";
12.                substitute "^" -> "19"; },
13.                ...
14.  Mileage matches [8]
15.      ...
16.      keyword "\bmiles\b", "\bmi\.", "\bmi\b",
17.              "\bmileage\b", "\bodometer\b";
18.  ...
```

Fig. 5. Car-Ads extraction ontology (partial).

existence at the time the wrapper is written and sites that change their layout and content after the wrapper is written. Wrappers based on extraction ontologies are robust. [2] Robust wrappers are critical to our approach: without them, we may have to create (by hand or at best semiautomatically) a wrapper for every new table encountered; with them, the approach can be fully automatic.

An extraction ontology consists of two components: (1) an *object/relationship-model instance* that describes sets of objects, sets of relationships among objects, and constraints over object and relationship sets, and (2) for each object set, a *data frame* that defines the potential contents of the object set. A data frame for an object set defines the lexical appearance of constant objects for the object set and establishes appropriate keywords that are likely to appear in a document when objects in the object set are mentioned. Fig. 5 shows part of our car-ads application ontology, including object and relationship sets and cardinality constraints (Lines 1–8) and a few lines of the data frames (Lines 9–18).

An object set in an application ontology represents a set of objects which may either be lexical or non-lexical. Data frames with declarations for constants that can potentially populate the object set represent lexical object sets, and data frames without constant declarations represent non-lexical object sets. *Year* (Line 9) and *Mileage* (Line 14) are lexical object sets whose character representations have a maximum length of 4 characters and 8 characters respectively. *Make*,

[2] Page-specific, handwritten wrappers (e.g. the early wrappers produced for TSIMMIS [4]) are not robust. Machine-learning-based wrappers (e.g. [16,26]) are not robust since new and changed pages must be annotated and learned. Wrappers that automatically infer regular expressions for Web pages (e.g. [5]) are robust in the sense that the regular-expression generator only needs to be rerun for new and changed pages; however, high page layout regularity is required, an assumption that often fails, but which we intend to consider in our future work with tables. Extraction ontologies (e.g. [8]) are robust because they are based on conceptual-model specifications of a domain of interest, not on page layout. Although they are hand-crafted, as ontologies typically are, our experience shows that an expert can create a reasonably good extraction ontology for a narrow domain of interest such as car-ads in a few dozen hours.

Model, *Price*, *Feature*, and *PhoneNr* are the remaining lexical object sets in our car-ads application; *Car* is the only non-lexical object set.

We describe the constant lexical objects and the keywords for an object set by regular expressions using Perl-like syntax. [3] When applied to a textual document, the `extract` clause (e.g. Line 10) in a data frame causes a string matching a regular expression to be extracted, but only if the `context` clause (e.g. Line 11) also matches the string and its surrounding characters. A `substitute` clause (e.g. Line 12) lets us alter the extracted string before we store it in an intermediate file. (For example, the *Year* data frame treats a year written "'95" as the constant "1995".) We also store the string's position in the document and its associated object-set name in the intermediate file. One of the non-lexical object sets must be designated as the *object set of interest—Car* for the car-ads ontology, as indicated by the notation "[→object]" in Line 1.

We denote a relationship set by a name that includes its object-set names (e.g. *Car has Year* in Line 2 and *PhoneNr is for Car* in Line 8). The *min:max* pairs in the relationship-set name are *participation constraints*. *Min* designates the minimum number of times an object in the object set can participate in the relationship set and *max* designates the maximum number of times an object can participate, with $*$ designating an unknown maximum number of times. The participation constraint on *Car* for *Car has Feature* in Line 6, for example, specifies that a car need not have any listed features and that there is no specified maximum for the number of features listed for a car.

In our initial work with semistructured and unstructured Web pages [8], a data-extraction ontology allowed us to recognize data values and context keywords for a particular application, organize data into records of interest, and fill object and relationship sets with data according to ontologically specified constraints. In our current work with tables, nested subtables in linked pages, and surrounding semistructured and unstructured text, we use extraction ontologies in much the same way. Recognized context keywords tend to be attributes; sometimes recognized values are also attributes. For tables, geometric layout gives us the clues we need to decide which recognized strings are attributes and which are values. This knowledge, plus the ontological domain knowledge about which attributes and values belong to which object sets, establishes the basis for determining record groupings and semantic correspondences for target attributes and relationships. Our system's ability to extract attributes and values and to pair them together constitutes the fundamental basis for enabling it to recognize tables containing data of interest and to discover mapping rules that can transform the contents of source tables to a target schema.

3. Table location

As a starting place, we assume that we have a Web page with a table whose rows correspond one-to-one with the fundamental object of interest in our application's data-extraction ontology. [4] We do not assume, however, that we have identified the table on Web page nor that we have, in hand, any linked pages of interest. The Web page in Fig. 2 is an example of such a starting Web page. The rows in the table of cars in Fig. 2 correspond one-to-one with *Car*, the object of interest declared in Fig. 5.

[3] Thus, for example, "\b" indicates a word boundary, "\d" indicates a numeric digit, and so forth.

[4] As stated earlier, we can use previous work [10] to make this determination.

Our table-location task is to find the fundamental table of interest in the top-level page and the tables of interest in linked pages. To resolve the table-location problem, we face all the problems mentioned in the introduction, i.e., *Multiple Panes*, *Tables for Layout*, *Table Rows Not in Table*, *Tables Displayed Piecemeal*, *Tables Spanning Multiple Pages*, and *No ⟨Table⟩ Tag*. We also face other problems we have encountered, including some that our system handles, such as folded tables and factored rows, and some that we report as future challenges in Section 6.

Our system resolves the problems of finding the main table of interest by using the following heuristics.

- *Potential table*. The main table must be embedded in an HTML ⟨*Table*⟩.
- *Table size*. The main table must have at least three rows and at least three columns.
- *Grid layout*. We can count the number of data cells in each row in the table. Letting N be the number of rows in the table that have the most common number of data cells and M be the number of rows in the table, the ratio N/M must exceed 2/3. This ensures that the vast majority of the rows extend across the width of the table and thus that the table, at least roughly, has the expected geometry of a table.
- *Attributes*. Based on the keywords and the object-set names for the various object sets in our extraction ontology, we have a reasonable idea about what some of the attribute names for a table should be. Thus, we look for a row near the top from which we have been able to extract 60% of the data entries as attributes—these attributes, of course, must be distinct. (Note that we do not depend on the table creator to mark the attributes with a ⟨*th*⟩ tag.) If we cannot find an attribute row in the table, we try columns, preferably leftmost columns. If we find an attribute column, we can transpose the table so that the attributes are in rows.
- *Value density*. Based on the values expected for the various lexical object sets, we find all ontology-recognized strings. If the ratio of the number of characters in recognized strings to the total number of characters in strings within the table exceeds 10%, we have some reasonable evidence that the table is of interest for the application. (Although 10% may seem low, previous experiments with density [9] show that the density test should fail only for extremely low percentages, usually below 1%.)
- *Folded tables*. Sometimes tables have so many columns that table designers fold them for viewing on a single page or in a single window either by placing the second half of the columns below the first half of the columns or by making two (or more) rows of attributes at the top that associate with pairs (triples,...) of values in the columns below. Thus, if more than one attribute row appears, we compare the attribute rows. If they are not the same, we treat the table as a folded table; otherwise we remove the duplicate attribute rows.
- *Factored-value rows*. We consider as possible factored values those values in each table row where the row has less than half the cells filled and the cells that are filled are adjacent left-most fields. (Many car-ads tables, for example, group cars by year and display the year in a row by itself above all the cars listed for that year.) We add factored values that are above the attribute row to all subsequent rows, and we add factored values that are below the attribute row to all subsequent rows until the next row of factored values. We eliminate rows that do not satisfy these factoring criteria—presumable these are not value table rows—for example, the row of buttons at the bottom of the table in Fig. 2.

- "*More*"*-link pages*. We can make use of linked components of the top-level table to help determine with certainty which strings are attributes and which are values by observing that the attributes remain the same across pages while the values change. For the page in Fig. 2, for example, all subsequent pages linked by "*Show 25 more*" have identical attributes on the top row of the table, namely

  $Year, Make$ and $Model, Price, Miles, Exterior, Photo$.

  Indeed, in this way, we are likely to be able to identify attributes, such as *Photo*, even when they are not in our application ontology.

For tables in linked pages, table detection is different. We use the following heuristics for these tables.

- *Table size*. We do not expect subtables to be as large as top-level tables. Thus we only require at least two rows or two columns.
- *Attributes*. This is the same as for top-level tables.
- *Attribute–value pair table*. To locate table components that contain attribute–value pairs, we look for a pair of columns where the strings in the first column have been extracted mostly as attributes and the strings in the second column have been extracted mostly as values. The table component in Fig. 3 is an example—the left column starting with *Price* contains many strings our extraction ontology recognizes as attributes, and the right column starting with $21,988 contains many strings our extraction ontology recognizes as values. Sometimes these types of tables are folded, so we must consider several pairs of columns side by side. As for other attribute tests, we use 60% as our threshold. We also check for row pairs in the same way to locate table components formatted with the attributes above the values, rather than to the left.
- *Single-attribute table*. To find lists like the *Features* list in Fig. 3, we look for a ⟨*ul*⟩ or an ⟨*ol*⟩ tag or for a ⟨*table*⟩ tag followed by a single-column table structure. We confirm that the single-attribute table is of interest by checking whether the ontology recognizes at least 60% of the strings as values of interest.
- *Page-spanning tables*. We follow a selected number of links from the top-level table to obtain several linked table rows. We then check the variability—attributes tend to remain the same from page to page (although sometimes table rows have more or fewer attributes), while values tend to vary (although some, such as colors, body styles, and transmission types are often identical).

4. Derivation of source-to-target schema mappings

We assume that each tuple in the top-level table corresponds to a primary object of interest. [5] One consequence of this assumption is that we can simply generate an object identifier for each

[5] We do not address fundamental mismatches of primary objects in our work here. Whether this approach extends to cases where we cannot easily find an alignment for the main objects of interest (cars in our example here) is a question for future research.

of these objects. Indeed, this is how we obtain the values under the attribute *Car* in Fig. 1. Another consequence of this assumption is that we can easily group the source information into record chunks, one chunk of information for each object of interest. The information chunk for the *2002 Honda Accord EX* in Fig. 2, for example, is the third tuple in the table plus all the information in Fig. 3. Having record chunks allows us to more easily build the atomic relationships between an object of interest and its associated data once we find the semantic correspondence for each target attribute. Hence, we are able to reduce the problem of finding a semantic correspondence to just finding the semantic correspondence for each target attribute A, which we defined earlier as the problem of finding the set of values constructed from source elements that corresponds to A.

We accomplish this objective using a "back-door" approach. Instead of directly searching for a mapping that associates each target attribute A with a value set in a source, we use our extraction ontology to search for values in the source that are likely to be found in the value set for A. Then, from the pattern of values we find, we infer what the mapping must be. This approach more easily allows us to recognize some of the unusual indirect mappings we are likely to encounter such as *Attribute as Value*, *Externally Factored Data*, and *Merged Attributes/Values* as discussed and illustrated in the introduction. The following four subsections correspond to the four steps of our proposed approach that follow the step of locating the table of interest, discussed previously in Section 3.

4.1. Form attribute–value pairs

The table understanding problem takes as input a table (for our work here, an HTML table) and produces standard records as output. Each record produced is a set of attribute–value pairs. A successful table-understanding system, for example, would produce the first record in the table in Fig. 4 as

$$\{\langle Make: ACURA\rangle, \langle Model: legend\rangle, \langle Year: 1992\rangle, \langle Colour: grey\rangle, \quad \langle Price: \$9500\rangle,$$
$$\langle Auto: Yes\rangle, \langle Air\ Cond.: No\rangle, \langle AM/FM: Yes\rangle, \langle CD: No\rangle\}$$

The hard part of table understanding is to recognize which cells contain the attributes and which contain the values and then to recognize which attributes go with which values. Our table recognition heuristics help solve this problem, at least for top-level tables whose attributes are at the top of columns and for linked subtables, which may be tables in their own right, simple one-column lists, or table-row components of tables that span multiple pages. [6]

Once we have identified attributes, we can immediately associate each cell in the grid layout of a table with its attribute. If the cell is not empty, we also immediately have a value for the attribute and thus an attribute–value pair. If the cell is empty, however, we must infer whether the table has a value based on internal factoring or whether there is no value. Fig. 6a shows an example of

[6] We recognize that [18] had an earlier solution for a much smaller subclass of HTML tables. We also recognize that there is a larger class of HTML tables and an even larger class of tables in general [19]. In our work here, however, we accept this intermediate, simpler solution for now, but in some of our other work [12,27] we have explored the attribute-recognition problem for a larger class of tables, and in future work, we intend to provide a solution to finding attribute–value pairs in general tables. Once we or others have a general solution, the approach we propose here carries over without change.

Year	Make	Model	Price
1995	Ford	F150 Super Cab	$6,988
		Contour GL	$3,988
	ACURA	INTEGRA LS	$14,500
	Honda	Civic EX	
1994	Ford	F150	$4,488
		Probe	$3,988
		Taurus LX	$2,988

(a)

Year	Make	Model	Price
1995	Ford	F150 Super Cab	$6,988
1995	Ford	Contour GL	$3,988
1995	ACURA	INTEGRA LS	$14,500
1995	Honda	Civic EX	
1994	Ford	F150	$4,488
1994	Ford	Probe	$3,988
1994	Ford	Taurus LX	$2,988

(b)

Fig. 6. Internal factoring in tables.

internal factoring. The empty *Year* cell for the *Contour GL*, for example, is clearly *1995*, whereas the *Price* for the *Honda* is simply missing. We recognize internal factoring in a two-step process: [7] (1) we detect potential factoring by observing a pattern of empty cells in a column, preferably a leftmost column or a near-leftmost column; (2) we check to see whether adding in the value above the empty cell helps complete a record by adding a value that would otherwise be missing.

Once we recognize the factoring in a table, we can rewrite the table's schema as a nested schema that reflects the factoring and then unnest the table to distribute factored values and to return the schema for the nested table to an unnested schema. Textually, we represent a nested component of a schema by $(A_i, \ldots, A_n)$* where the A_i's are attribute names. In general, nested components may appear inside of and along side of other nested components. The nested schema that defines the internal factoring for the table in Fig. 6a is *Year*, (*Make*, (*Model*, *Price*)*)*.

To unnest a table with a nested schema, we use a μ operator whose definition is based on the unnest operator in [15]. We write $\mu_N t$ to unnest the nested component N of nested table t. To unnest table T_a in Fig. 6a, for example, we can apply $\mu_{(Make,Model,Price)^*}\mu^*_{(Model,Price)}T_a$, which yields the table in Fig. 6b. Here, we start with *Year*, (*Make*, (*Model*, *Price*)*)*. After the first operation $\mu_{(Model,Price)^*}$, the schema is *Year*, (*Make*, *Model*, *Price*)* and the *Make*s have been distributed to the *Model*s, i.e. *Ford* appears in the empty cells in the *Make* column in Fig. 6a in our example. Then after the second operation $\mu_{(Make,Model,Price)^*}$, which distributes the *Year*s to each empty cell in the *Year* column, we have the table in Fig. 6b. Alternatively, we could have achieved the same result by applying $\mu_{(Model,Price)^*}\mu_{(Make,(Model,Price)^*)^*}T_a$, which first distributes the years to each make and then distributes *Year*-*Make* pairs to each *Model*. In either case, once we have resolved internal factoring with the μ operator, we immediately have the table's attribute–value pairs.

4.2. Adjust attribute–value pairs

After discovering attribute–value pairs, we make some adjustments to prepare each record for data recognition by means of an extraction ontology. We format attribute–value pairs for easy recognition by the extraction ontology. We add in linked sub-information for each record. This includes the attribute–value pairs in subtables, the attribute–value pairs for page-spanning tables, sets of values from lists for any single-attribute tables or lists, and raw header and footer text from the linked page. For the linked page in Fig. 3, for example, we would add attribute–value pairs

[7] This procedure is more fully described in [11], where we explain how we use a multi-dimensional cosine measure and a hill-climbing procedure to recognize factored values and appropriately distribute them to their proper records.

Make	Model	Year	Colour	Price	Auto	Air Cond.	AM/FM	CD
ACURA	legend	1992	grey	$9500	Auto		AM/FM	
AUDI	A4	2000	Blue	$34,500	Auto	Air Cond.	AM/FM	CD
BMW	325e	1985	black	$2700.00			AM/FM	
CHEVROLET	Cavalier Z24	1997	Black	$11,995.00		Air Cond.	AM/FM	

Fig. 7. Table in Fig. 4 transformed by the β operator.

such as ⟨*Body Style*: *Coupe*⟩ and ⟨*Transmission*: *Automatic*⟩, the listed *Features* as a set of values, and the raw text surrounding the table. If we wish to process non-text items such as icons, we could replace them with text; for example, we could replace a color-swatch icon with the name of the color. [8] We also process Boolean indicators, such as "*Yes*/*No*" in Fig. 4, by replacing "Yes" value pairs simply with the attribute name and dropping "No" value pairs altogether.

For our running example, the adjusted attribute–value pairs in the first record in Fig. 4 become

$$Make: ACURA; Model: legend; Year: 1992; Colour: grey; Price: \$9500; Auto; AM/FM;$$

Note that the Boolean-valued attribute–value pairs ⟨*Auto*: *Yes*⟩ and ⟨*AM*/*FM*: *Yes*⟩ have become simply *Auto*, meaning that the car has an automatic transmission, and *AM*/*FM*, meaning that the car has an AM/FM radio, and that *Air Cond.* and *CD* have disappeared altogether, meaning that the car has neither air conditioning nor a CD player.

When attribute names are the values we want and the values are some sort of Boolean indicator (e.g. Yes/No, True/False, 1/0, cell checked or empty), we transform the Boolean indicators into attribute-name values with the help of a β operator which we introduce here. Syntactically we write $\beta^{A}_{T,F} r$ where A is an attribute of relation r and T and F are respectively the Boolean indicators for the *True* value and the *False* value given as A values in r. The result of the β operator is r with the *True* values of the A column replaced by the string A and the *False* values of A replaced by the null string. As an example, consider $\beta^{Auto}_{Yes,No}\beta^{Air\ Cond.}_{Yes,No}\beta^{AM/FM}_{Yes,No}\beta^{CD}_{Yes,No}T$ which transforms the table T in Fig. 4 to the table in Fig. 7.

4.3. Perform extraction

Once we have adjusted attribute–value pairs as just discussed, we apply our extraction ontology. For our running example, the extraction for the first record in Fig. 4 yields

$$\{\langle Car: 0011\rangle, \langle Year: 1992\rangle, \langle Make: ACURA\rangle, \langle Model: legend\rangle, \langle Mileage: \rangle, \langle Price: \$9500\rangle, \langle PhoneNr: \rangle\},$$
$$\{\langle Car: 0011\rangle, \langle Feature: grey\rangle\},$$
$$\{\langle Car: 0011\rangle, \langle Feature: Auto\rangle\},$$
$$\{\langle Car: 0011\rangle, \langle Feature: AM/FM\rangle\}.$$

We emphasize that our extraction ontology is capable of extracting from unstructured text as well as from structured text. Indeed, we can directly extract the phone numbers and features from the text, list, and page-spanning table in Fig. 3.

[8] Our current implementation does not replace non-text items with text. We leave this for future work.

For direct extraction we introduce the ϵ operator, which is based on a given extraction ontology. We define $\epsilon_S t$ as an operator that extracts a value, or values, from unstructured or semistructured text t for object set S in the given extraction ontology O according to the extraction expression for S in O. The ϵ operator extracts a single value if S functionally depends on the object of interest x in O, and it extracts multiple values if S does not functionally depend on x. As an example, $\epsilon_{PhoneNr} t$ extracts *1-877-944-2842* from the unstructured footer text t of the table in Fig. 2 and returns it as the single-attribute, single-tuple, constant relation $\{\langle PhoneNr$: 1-877-944-2842$\rangle\}$. We can use the ϵ operator in conjunction with a natural join to add a column of constant values to a table. For example, letting t represent the unstructured text on the page in Fig. 2 we could apply $\epsilon_{PhoneNr} t \bowtie T$ to add a column for *PhoneNr* to table T in Fig. 2.

At this point we could take a data-warehousing approach and directly insert this extracted information into a global database as Fig. 1 implies. Alternatively, instead of populating the global database, we can use this information to infer a mapping from the source to the target and extract information from sources whenever a user poses a query against the global database schema.

4.4. Infer mappings

We record the sequence of transformations produced when we form attribute–value pairs and when we adjust attribute–value pairs in preparation for extraction, and we observe the correspondence patterns obtained when we extract tuples with respect to a given target ontology. Based on this sequence of transformations and these correspondence patterns, we can produce a mapping of source information to a target ontology. As a simple example, consider mapping the table T_a in Fig. 6a to the target schema for the tables in Fig. 1. We first apply the μ operator to do the unnesting and obtain table T_b in Fig. 6b. We then observe that objects extracted for the *Year* object set in the target come from the *Year* column in T_b. Similarly, for *Make*, *Model*, and *Price*, we also observe a direct correspondence. Hence, we can record the semantic correspondence of T_a and the target schema as the mapping $Year = \pi_{Year}\mu_{(Make,Model,Price)^*}\mu_{(Model,Price)^*}T_a$, $Make = \pi_{Make}\mu_{(Model,Price)^*}T_a$, $Model = \pi_{Model}T_a$, and $Price = \pi_{Price}T_a$.

Creating an inferred mapping has two important advantages. (1) The global view can be virtual. Since we have a formal mapping, we can translate any query applied to the global view to a query on the source, optimize it, execute it, and return the results from the source for the global query. [9] (2) We can obtain additional values not recognized by the ontology, but which are nevertheless valid values in the source. For example, *Super Cab* may not be technically part of the model for the *1995 Ford F150* in Fig. 6a, and the ontology may therefore not recognize it as part of

[9] Since the main contribution of this paper is the derivation of the mappings, not local query rewriting [20,28], we leave for future research a full explanation of this procedure. The idea, however, is quite straightforward once we have the semantic correspondence. Since we know the record structure of the source, we can add object identifiers for each value in the value sets. We can then join over these OID-augmented value sets to obtain a universal relation. Since the mappings result in sets associated with target attributes, if we add columns of nulls for each target attribute with no correspondence, we will have a universal relation over the target attributes. We can now project onto the schema for each of the target tables. We thus have a standard view definition, which we can substitute for each of the table references in a global query. We can then optimize the query and execute it on the source, returning only those values from the source that contribute to the global query result.

the model. Nevertheless, someone declared *Super Cab* to be part of the model, and we should therefore extract it as such. Using the mapping, we extract full strings under *Model* in Fig. 6a and thus we obtain *F150 Super Cab* as the model for the *1995 Ford* even though the extraction ontology may only pick up *F150* as the model. As another example, the mapping approach would obtain *all* the *Features* in the list in Fig. 3 even though the ontology may not recognize all of them as features. When we use the mapping, we generalize over the structure and infer additional information not specifically recognized by the ontology. [10]

To complete our task, we now define a few more operators. These operators, together with the ones we defined earlier, provide the complete set of operators we need for mapping all the HTML tables we have encountered. [11]

For merged attributes we need to split values. We can divide values into smaller components with a δ operator which we introduce here. We define $\delta^{A}_{B_1,\ldots,B_n} r$ to mean that each value v for attribute A of relation r is split into $v_1,\ldots,v_n$, one for each new attribute $B_1,\ldots,B_n$ respectively. Associated with each B_i is a procedure p_i that defines which part of v becomes v_i. In this paper we specify each procedure p_i by regular expressions with `extract` and `context` phrases similar to those defined for extraction ontologies discussed earlier in Section 2. The result of the δ operator is r with n new attributes, $B_1,\ldots,B_n$, where the B_i value on row k is the string that results from applying p_i to the string v on row k for attribute A. As an example, consider $\delta^{Make\ and\ Model}_{Make,Model} T$, where T is the table in Fig. 2, the expression associated with *Make* is `extract` "\S+" context "\S+\s" which extracts the characters of the string value up to the first space, and the expression associated with *Model* is `extract` "\S.*" context "\s.+" which extracts all the remaining characters in the string after the first space. [12] This operation adds the two columns in Fig. 8 to the table in Fig. 2.

For split attributes we need to merge values. We can gather values together and merge them with a γ operator which we introduce here. Syntactically, we write $\gamma_{B \leftarrow A_1+\cdots+A_n} r$ where B is a new attribute of the relation r and each A_i is either an attribute of r or is a string. The result of the γ operator is r with an additional attribute B, where the B value on row k is a sequential concatenation of the row-k values for the attributes along with any given strings. As an example, consider $\gamma_{Model\,with\,Trim \leftarrow Model + " " + Trim} T_1$ which converts table T_1 in Fig. 9 to table T_2.

We can use standard set operators to help sort out subsets, supersets, and overlaps of value sets. We can, for example, take a union of the exterior colors in Fig. 2 and features in Fig. 3 to form part of the set for *Feature* in Fig. 1. After adding needed projection and renaming operations, this union is $\rho_{Exterior \leftarrow Feature}\pi_{Exterior} T \cup \rho_{Features \leftarrow Feature} T/Make\ and\ Model/Features$, where T is the table in Fig. 2 and $T/Make\ and\ Model/Features$ is a path expression that follows the link under *Make and Model* in table T to the *Features* list in Fig. 3. When we need subsets of a set, we can extend the standard selection operator $\sigma_C r$ to allow C to be a regular expression that identifies the subset of values we wish to include. Given that we can also apply set-difference operations, we can resolve overlapping sets by operator combinations.

[10] For future research, we are considering the possibility of strengthening recognizers for extraction ontologies as they encounter additional values recognized through mappings, but not recognized directly through regular expressions. Strengthening ontologies in this way is similar to the work on learning reported in [14,25].

[11] In future research, we would like to obtain a general completeness result.

[12] In Perl, "\s" matches a white space character, "\S" matches a non-space character, "." matches any character, "+" indicates one or more repetitions, and "*" indicates zero or more repetitions.

Make	Model
Pontiac	Firebird
Acura	RL 3.5
Honda	Accord EX
Honda	Passport

...

Honda	Accord Value Package
Chevrolet	Silverado C1500

Fig. 8. Columns added to the table in Fig. 2 by the δ operator.

T_1

Make	Model	Trim
Ford	Contour	GL
Ford	Taurus	LX
Honda	Civic	EX

T_2

Make	Model	Trim	Model with Trim
Ford	Contour	GL	Contour GL
Ford	Taurus	LX	Taurus LX
Honda	Civic	EX	Civic EX

Fig. 9. Application of the γ operator to table T_1 yielding table T_2.

Target Attribute	Source Derivation Expression for Value Sets
Year	$\pi_{Year}T$
Make	$\pi_{Make}T$
Model	$\pi_{Model}T$
Price	$\pi_{Price}T$
Feature	$\rho_{Colour \leftarrow Feature}\pi_{Colour}T$ $\cup\ \rho_{Auto \leftarrow Feature}\pi_{Auto}\beta^{Auto}_{Yes,\ No}T$ $\cup\ \rho_{Air\ Cond. \leftarrow Feature}\pi_{Air\ Cond.}\beta^{Air\ Cond.}_{Yes,\ No}T$ $\cup\ \rho_{AM/FM \leftarrow Feature}\pi_{AM/FM}\beta^{AM/FM}_{Yes,\ No}T$ $\cup\ \rho_{CD \leftarrow Feature}\pi_{CD}\beta^{CD}_{Yes,\ No}T$

Fig. 10. Inferred mapping from source table T in Fig. 4 to target table in Fig. 1.

Target Attribute	Source Derivation Expression for Value Sets
Year	$\pi_{Year}T$
Make	$\pi_{Make}\delta^{Make\ and\ Model}_{Make,\ Model}T$
Model	$\pi_{Model}\delta^{Make\ and\ Model}_{Make,\ Model}T$
Mileage	$\rho_{Miles \leftarrow Mileage}\pi_{Model}T$
Price	$\pi_{Price}T$
PhoneNr	$\epsilon_{PhoneNr}t$
Feature	$\rho_{Exterior \leftarrow Feature}\pi_{Exterior}T$ $\cup\ \rho_{Features \leftarrow Feature}T/Make\ and\ Model/Features$ $\cup\ \rho_{Body\ Type \leftarrow Feature}T/Make\ and\ Model/Body\ Type$ $\cup\ \rho_{Body\ Type \leftarrow Feature}T/Make\ and\ Model/Body\ Style$ $\cup\ \rho_{Body\ Type \leftarrow Feature}T/Make\ and\ Model/Exterior$ $\cup\ \rho_{Transmission \leftarrow Feature}T/Make\ and\ Model/Transmission$ $\cup\ \rho_{Engine \leftarrow Feature}T/Make\ and\ Model/Engine$

Fig. 11. Inferred mapping from the source tables T with surrounding text t in Fig. 2 and $T/Make\ and\ Model$ in Fig. 3 to the target table in Fig. 1.

Now that we have the operators we need, we can give examples. Fig. 10 gives the mapping from the source table in Fig. 4 to the target table in Fig. 1. Observe that we have transformed all the Boolean values into attribute-name values and that we have gathered together all the features as *Feature* values. Fig. 11 gives the mapping for the car-ads from the site for Figs. 2 and 3. Observe that we have split the makes and models as required, matched the synonyms *Miles* and *Mileage*, extracted the *PhoneNr* from the free text, and gathered together all the various features as *Feature*

values. The astute reader will have noticed that we extracted only one *Price* and *Mileage* even though two or more appear—since they functionally depend on car, we pick up only the first—and that we union in the color twice—once from the top-level table in Fig. 2 and once from the page-spanning table in Fig. 3 since *Feature* does not depend functionally on car. The astute reader will have also noticed that we did not pick up the *Fuel Type*, *Stock Number*, and *VIN*—these, like *Photo*, neither appear separately in our ontology nor as recognized features in our ontology.

5. Experimental results and discussion

We now present the results of two experiments in the domains of car advertisements and cell-phones.

5.1. Car advertisements

We gathered tables of car advertisements from many more than a hundred different English-language Web sites. Because of human resource limitations, however, we analyzed only 60.

Of the 60 car-ads tables we analyzed, 28 included links to other pages containing additional information about an advertised car (Figs. 2 and 3 show a typical example). For all 60 tables, we first applied our system to identify and list attribute–value pairs for tuples of top-level tables, and then for the 28 tables with links, we appropriately associated linked information with each tuple. We then applied our extraction step and looked for mapping patterns.

Since our objective was to obtain mappings (rather than data), it was not necessary for us to process every tuple in every table. Hence, from every table, we processed only the first 10 car-ads. As a threshold, we required six or more occurrences of a pattern to declare a mapping. A human expert judged the correctness of each mapping. [13] We considered a mapping declaration for a target attribute to be completely correct if the pattern recognized led to exactly the same mapping as the human expert declared, partially correct if the pattern led to a unioned (or intersected) component of the mapping, and incorrect otherwise. For data outside of tables, the system mapped an individual value to either the right place or the wrong place or did not map a value it should have mapped.

5.1.1. Results—car advertisements

We divided the 60 car-ads tables into two groups: 7 "training" tables and 53 "test" tables. We used the 7 "training" tables to generate the heuristics we used in table locating and table understanding. For the 7 training tables, we were able to locate 100% of the top-level tables as well as all the applicable tables in the linked pages. For the 53 test pages, we were able to locate 46 top-level tables successfully (86.8%). Among these 46 tables, 28 had links to additional pages with more detail about each car ad. Of the 28 additional pages, 13 had structured car-ad information, while 15 included unstructured information (which is fine for data extraction, but does not apply for gen-

[13] Although expert judgement for tables can sometimes be hard [13], establishing correctness results for car-ads for our target table was not difficult.

erating table mappings). The system correctly analyzed 12 out of the 13 linked pages of structured information; it also incorrectly declared that it found structured information in 2 linked pages.

We also analyzed our mapping approach for successfully located tables from the test set. For the 46 recognized tables, there were 319 mappings, of which we correctly or partially correctly discovered 296 (92.8%), missing 23 (7.2%); we (incorrectly) declared 13 false mappings (4.2% of 309 declared mappings). Of the correct mappings, 228 of 296 (77%) came from top-level tables, while 58 (19.6%) came from linked tables, and 10 (3.4%) came from both top-level and linked tables. Of the 296 mappings we correctly discovered, 121 (40.9%) were direct, in the sense that the attributes in the source and target schemas were identical, and 175 (59.1%) indirect matches. Of the 175 indirect matches, 28 used synonyms and thus required only renaming with a ρ operator, 1 had Boolean values and thus required a β operator, 93 included features scattered under various attributes and in raw text and thus required $\cup$ and ϵ operators, 2 provided only factored telephone numbers and thus required ϵ and $\bowtie$ operators, 53 needed to be split and thus required a δ operator, and some required combinations of these operators (e.g. synonyms and union). The values we needed to split came in a variety of different combinations and under a variety of different names. We found, for example, *Description* as an attribute for the combination *Year* + *Make* + *Model* + *Feature*, *Model Color* as an attribute for *Make* + *Model* + *Color*, and *Model* as an attribute for *Year* + *Make* + *Model*.

5.1.2. Discussion—car advertisements

Locating the correct table and understanding it properly is not a trivial problem. As we discussed in Section 3, we consider attributes, values, and table layout when seeking a table. Our system failed to locate 7 out of 53 test tables. All but one of the 7 missed tables contained uncommon attributes in the source page. For example, some Web sites use abbreviations like "PW", "PL", "CC", and "AC"; others include attributes that are irrelevant to the extraction ontology, such as "Image", "Click on Thumbnail", and "Location". This caused our system to fail to detect an attribute row (or column) in a table, leading to a failure to identify the correct table. The other table-location failure occurred because the top table only had 2 columns, but we required a minimum size of 3 columns by 3 rows.

For linked pages, the system was not able to find the correct table (single attribute–value pairs) for one site (out of 13). This is because all cars shared a single linked page containing information for all the cars (a case we had not considered). Our system incorrectly identified two linked pages as containing tables of interest because it interpreted some values as attributes or vice versa.

As mentioned in our earlier discussion, discovering correct mappings can lead to an increase in values extracted compared to what would have been found by the extraction ontology alone and can also therefore lead to the acquisition of additional knowledge for the extraction ontology. In our experiments, we required 60% of the values to match to declare a mapping. Overall, we actually achieved roughly 90–95%, a much higher percentage. This, however, leaves about 5–10% of the approximately 3000 values encountered in tables as being unrecognized by the extraction ontology (and potentially many hundreds more since we processed only 10 car-ads per site). Examples include non-US models such as the Toyota Starlet or Nissan Presea; elaborately described features such as "telescoping steering wheel"; abbreviations not encountered previously such as "leath int" for "leather interior"; and features simply not encountered before, such as "trip computer".

We missed 23 mappings and only declared 13 false mappings. Our system missed 10 mappings of car model because the extraction ontology was targeted to US car-ads, and so non-US car-ads introduced models that our system did not recognize. The system missed 2 price mappings, 1 mileage mapping, and 2 feature mappings because the extraction ontology was overly restrictive. All of these problems can be corrected by minor adjustments to the extraction ontology. The system missed another 5 mappings because of bugs in the original documents (ill-formed HTML) or due to the use of special codes to indicate particular values like colors. Sometimes Web sites use generic "filler" values such as "contact us," "please call," or "unknown" instead of listing actual prices, mileages, and so on; we missed 3 mappings for this reason. For the 13 incorrect mappings, 8 came from incorrectly understood linked pages. One of these 8, for example, maps "Location" to "Model," because the location is "Aurora" (a city in Colorado) which is also a model name for Oldsmobile. The other primary confusion for our tool was distinguishing between numbers such as price and mileage.

5.2. Cell-phone sales

The car-advertisements example uses our most mature extraction ontology, and so we expected to achieve good results using it. To see how our table understanding approach works with less developed extraction ontologies, we tested our system in the US cell-phone sales domain. Fig. 12 shows part of a top page of a typical cellular-phone application, and Fig. 13 shows part of a page obtained by clicking on *Mlife Local $29.99*. We used the following target schema:

$$\{Plan, Carrier, MonthlyFee, AnyTimeMin, OffPeakMin, ContractLength,$$
$$ActivationFee\ CancellationFee\}$$
$$\{Plan, Feature\}.$$

Similar to our car-ads experiment, we started with a training set of 5 cell-phone Web pages, and we tuned our extraction ontology to handle these 5 cases. We then gathered pages from 12 cell-phone Web sites for our test set, and found that our tool correctly located top-level tables in 11 of them (91.7%). The table that was missed was excluded because it only had two columns, thus failing our minimum size threshold of 3 rows and 3 columns. There were also 4 linked pages, all of which contained relevant tables that our tool properly located. A human expert judged that there were 97 mappings relevant to our extraction ontology in the 11 sites for which we correctly identified the top-level table. Our system declared 103 mappings, of which 15 were false mappings (14.6%), and 88 were correct or partially correct mappings (85.4%). In this experiment, 48 mappings came from top-level tables (46.6%), while 52 came from linked pages (50.5%), and 3 came from both top-level and linked pages. Our system missed 9 mappings (9.3% of 97).

Of the 15 false mappings, 6 came from linked tables, and in all cases these were for the target attribute *OffPeakMin*. The other 9 false mappings came from top-level tables: 5 for *AnyTimeMin*, 2 for *ActivationFee*, and 1 each for *CancellationFee* and *OffPeakMin*. In all cases, the false mappings were related to numeric attributes. With more context information in the extraction ontology, our tool could do a better job distinguishing the meaning of different numbers. As expected, the false-positive rate for the cell-phone domain is several times higher than for car-ads (14.6% versus 4.2%), which can be attributed partially to the relative amount of effort spent in developing

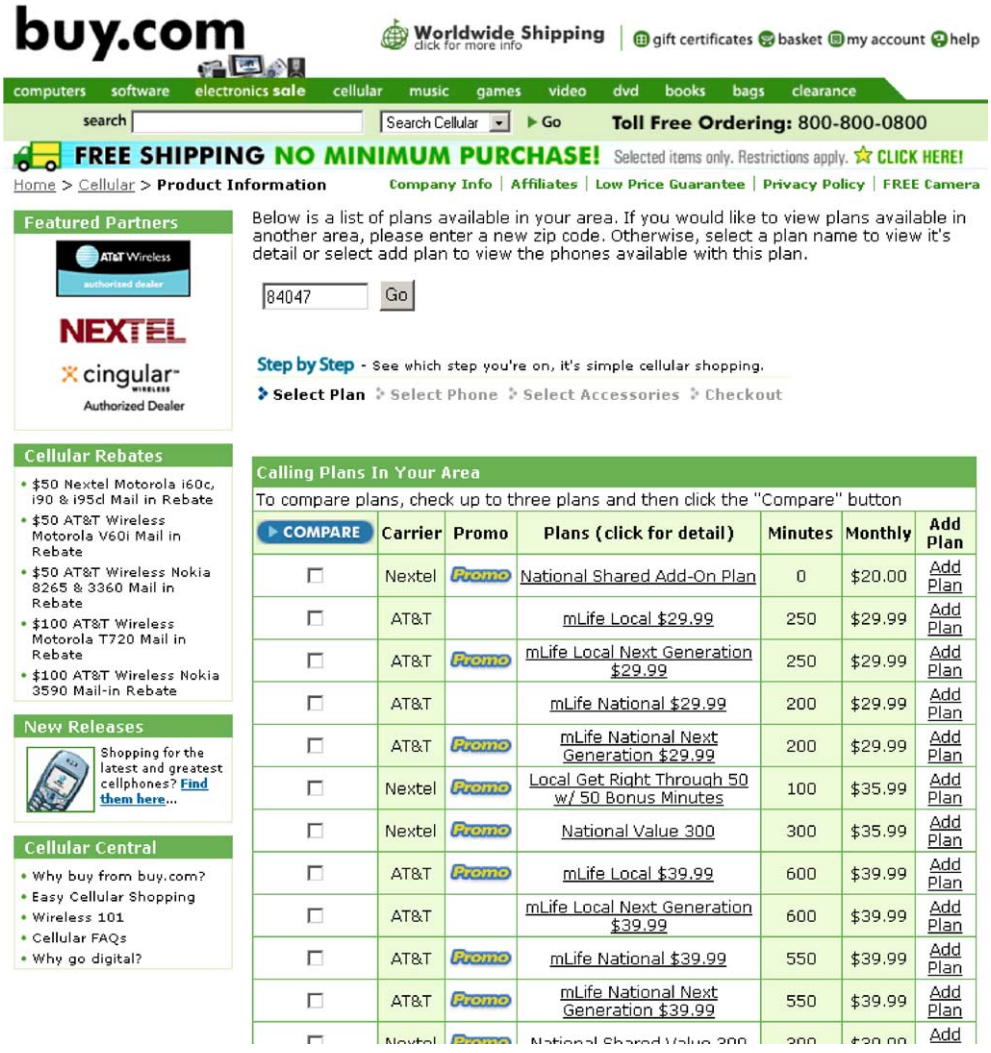
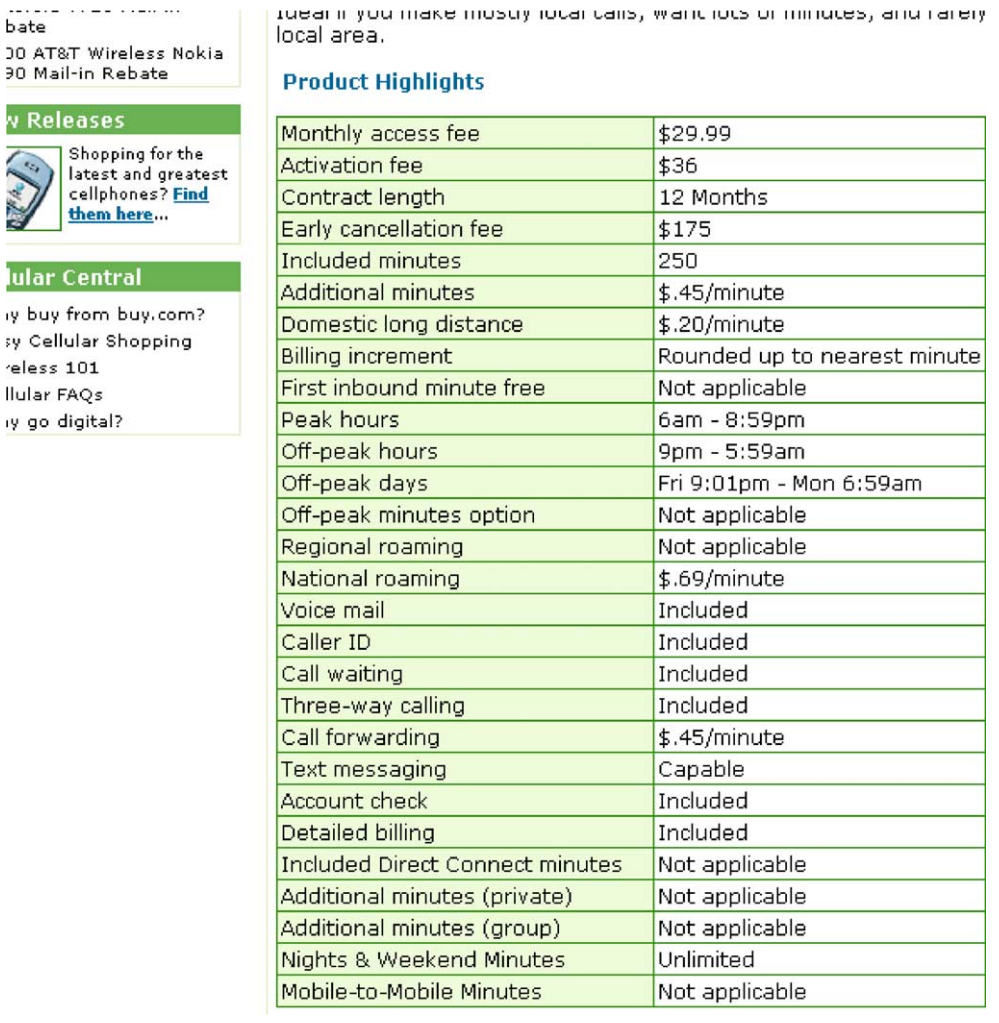


Fig. 12. Web page with table of cell-phones from Buy.com.

the corresponding extraction ontologies, and partially to the high degree of similarity between the domains of attributes in the cell-phone application.

An interesting aspect of the cell-phone domain is that of the 88 mappings we correctly discovered, none were direct (as compared to 40.9% for car-ads). That is, we had to apply some transformation operator to every mapping in the cell-phone application: 38 mappings used synonyms and thus required a ρ operator for renaming; 34 mappings included features scattered under various attributes, and thus required $\cup$ operators; and 24 mappings needed to be split and thus required a δ operator. Some mappings required combinations of operators.

The overall performance of our tool in the cell-phone sales domain is reasonably good and generally in line with our expectations. We could improve the outcome by tuning the extraction ontology more carefully.



Monthly access fee	$29.99
Activation fee	$36
Contract length	12 Months
Early cancellation fee	$175
Included minutes	250
Additional minutes	$.45/minute
Domestic long distance	$.20/minute
Billing increment	Rounded up to nearest minute
First inbound minute free	Not applicable
Peak hours	6am - 8:59pm
Off-peak hours	9pm - 5:59am
Off-peak days	Fri 9:01pm - Mon 6:59am
Off-peak minutes option	Not applicable
Regional roaming	Not applicable
National roaming	$.69/minute
Voice mail	Included
Caller ID	Included
Call waiting	Included
Three-way calling	Included
Call forwarding	$.45/minute
Text messaging	Capable
Account check	Included
Detailed billing	Included
Included Direct Connect minutes	Not applicable
Additional minutes (private)	Not applicable
Additional minutes (group)	Not applicable
Nights & Weekend Minutes	Unlimited
Mobile-to-Mobile Minutes	Not applicable

Fig. 13. Linked page with additional cell-phone information from buy.com.

6. Implementation status

Like most of our other work, our table-understanding tool is implemented in Java. We characterize its current status as "prototype quality." It accomplishes a great deal, but does not handle every situation we have encountered. Table 1 gives a summary of the implementation status with respect to some of the problems we discovered in the course of this project. In Sections 1.1, 1.2, and 3 we described a number of problems related to locating and extracting data from HTML tables. Table 1 indicates our progress with each problem. We have partially addressed some of the problems, such as *Folded Tables* (see Section 3). In this case, we have marked the problem as *Implemented* because we have devised an initial solution, but also as *Future Work* because we would like to create a more general solution to the problem.

Table 1
Implementation status and future work

Problem[a]	Implemented	Future work	
		Within scope	Beyond scope
Multiple panes	×		
Tables for layout	×		
Table rows not in table	×		
Tables displayed piecemeal	×		
Tables spanning multiple pages	×		
No ⟨Table⟩ tag	×		
Folded tables	×	×	
Merged attributes/values	×		
Split attributes/values		×	
Subsets	×		
Supersets		×	
Synonyms	×		
Homonyms	×		
Extra information	×		
Missing information	×	×	
Linked information	×	×	
List table	×		
Position of attributes	×	×	
Externally factored data	×	×	
Internally factored data	×	×	
Duplicate data	×		
Unexpected multiple values	×		
Attribute as value	×		
Image (Text) attributes			×
Image (Text) values			×
Image (Icon) attributes			×
Image (Icon) values			×
Image (Picture) attributes			×
Image (Picture) values			×

[a] This list of problems are those we have personally encountered—this list is representative, but not exhaustive.

The last column in Table 1, *Future Work Beyond Scope*, highlights an area that is outside the scope of our current project, but is potentially of interest in the domain of table understanding. Images in Web pages have numerous purposes. Sometimes images contain text that is readily understood by humans but not by standard DOM-tree parsers. This may occur because Web site designers wish to use a special symbol or a special font that is hard to render in plain HTML. Some designers use images to achieve greater control over cross-platform look-and-feel. In any event, an OCR step could render these text-containing images into a format that our tool can process. In other cases, images are icons representing a function (e.g. click on the icon to perform some task or to link to another page) or an encoding of information (e.g. "editor's choice" icon marks a product that received high acclaim from an editorial board). If we could understand the meaning of icons, we could handle some cases where attribute values are encoded as images. Finally, it is often the case that an image is just a photo or picture that conveys some information to

a human, such as, say, "this particular car is red and appears to be in good external condition." We have seen cases of all three usages for images (as text, icons, or pictures), at both attribute and value levels. There are unique challenges to understanding the use of images in these six contexts, and this is a promising and interesting area for future research.

7. Conclusion

In this paper, we suggested a different approach to the problem of schema matching, one which may work better for the heterogeneous HTML tables encountered on the Web. In essence, we transformed the table location problem and the schema matching problem into an extraction problem that provides information to distinguish tables from surrounding text and layout and to infer the semantic correspondence between a source table and a target schema. We gave experimental evidence to show that our approach can be successful. In particular, we tested two applications: car advertisements and cell-phone sales. We correctly located 90% of the tables (top-level and linked) in pages for these two applications. Then, from the located tables we inferred 93% of the appropriate mappings with a precision of 96% for our car-ads application and inferred 91% of the appropriate mappings with a precision of 85% for our cell-phone application.

As a next step in our work on extraction from HTML tables, we would like to implement the ideas we marked as being "Within Scope" in Table 1. To take the next step beyond implementing these ideas, we believe that significant work on layout appearance and intelligent-character/graphical-object recognition is needed.

Beyond table location and understanding, we recognize that many tables are behind forms, in the so-called "hidden Web" [24]. Thus, in order to arrive at much of the data we can process with the system we have proposed in this paper, we need to access the hidden Web, a problem on which we are currently working [21,17]. Once extracted, if the result is a table, we can use the techniques presented here to extract the data into a target view. If the result is not a table, we use techniques we have previously developed [8] to extract the data. Further, we also plan to piece together all the components we have developed in our data-extraction work [7] into a comprehensive extraction tool.

Acknowledgement

Supported in part by the National Science Foundation under grant IIS-0083127. Also supported in part by the Kevin and Debra Rollins Center for eBusiness at Brigham Young University.

References

[1] Available from <autoscanada.com>, Summer 2001.
[2] J. Biskup, D.W. Embley, Extracting information from heterogeneous information sources using ontologically specified target views, Information Systems 28 (3) (2003) 169–212.
[3] Available from <www.bobhowardhonda.com>, April 2003.

[4] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, J. Widom, The TSIMMIS project: Integration of heterogeneous information sources, in: IPSJ Conference, Tokyo, Japan, October 1994, pp. 7–18.
[5] V. Crescenzi, G. Mecca, P. Merialdo, Roadrunner: Towards automatic data extraction from large web sites, in: Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01), Rome, Italy, September 2001.
[6] A. Doan, P. Domingos, A. Halevy, Reconciling schemas of disparate data sources: A machine-learning approach, in: Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD 2001), Santa Barbara, CA, May 2001, pp. 509–520.
[7] Home page for BYU Data Extraction Research Group, Available from <http://www.deg.byu.edu>.
[8] D.W. Embley, D.M. Campbell, Y.S. Jiang, S.W. Liddle, D.W. Lonsdale, Y.-K. Ng, R.D. Smith, Conceptual-model-based data extraction from multiple-record Web pages, Data and Knowledge Engineering 31 (3) (1999) 227–251.
[9] D.W. Embley, Y.S. Jiang, Y.-K. Ng, Record-boundary discovery in Web documents, in: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD'99), Philadelphia, PA, 31 May–3 June 1999, pp. 467–478.
[10] D.W. Embley, Y.-K. Ng, L. Xu, Recognizing ontology-applicable multiple-record Web documents, in: Proceedings of the 20th International Conference on Conceptual Modeling (ER2001), Yokohama, Japan, November 2001, pp. 555–570.
[11] D.W. Embley, L. Xu, Record location and reconfiguration in unstructured multiple-record Web documents, in: Proceedings of the Third International Workshop on the Web and Databases (WebDB2000), Dallas, TX, May 2000, pp. 123–128.
[12] T.B. Haas, The development of a prototype knowledge-based table-processing system, Master's thesis, Brigham Young University, Provo, UT, April 1998.
[13] J. Hu, R. Kashi, D. Lopresti, G. Nagy, G. Wilfong, Why table ground-truthing is hard, in: Proceedings of the Sixth International Conference on Document Analysis and Recognition, Seattle, WA, September 2001, pp. 129–133.
[14] R. Jones, A. McCallum, K. Nigam, E. Riloff, Bootstrapping for text learning tasks, in: IJCAI-99 Workshop on Text Mining: Foundations, Techniques, and Applications, Stockholm, Sweden, 1999, pp. 52–63.
[15] H.F. Korth, A. Silberschatz, Database System Concepts, second ed., McGraw-Hill, Inc., New York, NY, 1991.
[16] N. Kushmerick, D.S. Weld, R. Doorenbos, Wrapper induction for information extraction, in: Proceedings of the 1997 International Joint Conference on Artificial Intelligence, 1997, pp. 729–735.
[17] S.W. Liddle, D.W. Embley, D.T. Scott, S.H. Yau, Extracting data behind Web forms, in: Proceedings of the Joint Workshop on Conceptual Modeling Approaches for E-business: A Web Service Perspective (*eCOMO* 2002), Tampere, Finland, October 2002, pp. 38–49.
[18] S. Lim, Y. Ng, An automated approach for retrieving hierarchical data from HTML tables, in: Proceedings of the Eighth International Conference on Information and Knowledge Management (*CIKM'99*), Kansas City, MO, November 1999, pp. 466–474.
[19] D. Lopresti, G. Nagy, Automated table processing: An (opinionated) survey, in: Proceedings of the Third IAPR Workshop on Graphics Recognition, Jaipur, India, September 1999, pp. 109–134.
[20] A.Y. Levy, A. Rajaraman, J.J. Ordille, Querying heterogeneous information sources using source descriptions, in: Proceedings of the Twenty-second International Conference on Very Large Data Bases, Mumbai (Bombay), India, 1996, pp. 109–134.
[21] S.W. Liddle, S.H. Yau, D.W. Embley, On the automatic extraction of data from the hidden Web, in: Proceedings of the International Workshop on Data Semantics in Web Information Systems (*DASWIS-2001*), Yokohama, Japan, November 2001, pp. 106–119.
[22] J. Madhavan, P.A. Bernstein, E. Rahm, Generic schema matching with Cupid, in: Proceedings of the 27th International Conference on Very Large Data Bases (*VLDB'01*), Rome, Italy, September 2001, pp. 49–58.
[23] R. Miller, L. Haas, M.A. Hernandez, Schema mapping as query discovery, in: Proceedings of the 26th International Conference on Very Large Databases (*VLDB'00*), Cairo, Egypt, September 2000, pp. 77–88.
[24] S. Raghavan, H. Garcia-Molina, Crawling the hidden Web, in: Proceedings of the 27th International Conference on Very Large Data Bases (*VLDB'01*), Rome, Italy, September 2001.

[25] E. Riloff, R. Jones, Learning dictionaries for information extraction by multi-level bootstrapping, in: Proceedings of the Sixteenth national Conference on Artificial Intelligence (*AAAI-99*), Orlando, FL, July 1999, pp. 474–479.
[26] S. Soderland, Learning information extraction rules for semi-structured and free text, Machine Learning 34 (1–3) (1999) 233–272.
[27] K. Tubbs, Recognizing records from the extracted cells of genealogical microfilm tables. Master's thesis, Brigham Young University, Provo, UT, December 2001, Available from <http://www.deg.byu.edu>.
[28] J.D. Ullman, Information integration using logical views, in: F.N. Afrati, P. Kolaitis (Eds.), Proceedings of the 6th International Conference on Database Theory (*ICDT'97*), Lecture Notes in Computer Science, vol. 1186, Springer-Verlag, Delphi, Greece, 1997, pp. 19–40.

David W. Embley received a B.A. in Mathematics (1970) and an M.S. in Computer Science (1972), both from the University of Utah. In 1976 he earned his Ph.D. in Computer Science from the University of Illinois. From 1976 to 1982 he was a faculty member in the Department of Computer Science at the University of Nebraska, where he was tenured in 1982. Since then he has been a faculty member in the Department of Computer Science at Brigham Young University.

His research interests include database systems and theory and the extraction and integration of web data. He is the author of the book *Object Database Development: Concepts and Principles* and a coauthor of the book *Object-oriented Systems Analysis*: A Model-driven Approach. He is a member of the steering committee for the International Conferences on Conceptual Modeling (the ER Conferences), and he has served as chair for the committee.

Cui Tao is a Ph.D. candidate at Brigham Young University. She received a B.S. degree in 1997 from Beijing Normal University, where she majored in Biology and minored in Computer Science. In 2003, she received an M.S. in Computer Science from Brigham Young University. She has been working with the data-extraction research group at Brigham Young University for more than three years, studying data extraction, data integration, source discovery, and table understanding. Her current research focuses on data integration over heterogeneous biological data.

Stephen W. Liddle is the Academic Director of the Kevin and Debra Rollins Center for eBusiness at Brigham Young University, and is Associate Professor of Information Systems at the Marriott School of Management, where he has worked since 1995. He holds the Grant and David Faculty Fellowship in the School of Accountancy and Information Systems. He received his B.S. and Ph.D. degrees in Computer Science from Brigham Young University in 1989 and 1995 respectively. His research interests include applications of conceptual modeling, data extraction from unstructured and semistructured sources, web application development, object-oriented systems, and the application of computer science to business needs, especially for e-business related topics. He has published dozens of refereed articles in journals, book collections, conferences, and workshop proceedings since the early 1990s. He has also been active in organizing major computer science conferences and workshops, serving recently as program co-chair for ER2003 and ISTA2003, and general chair for ISTA2004. He is a member of the ER Steering Committee, and over the past decade has served on numerous international program committees. He also is a trustee for the Utah Information Technology Association, and is a member of the Scientific Advisory Board for CARE Technologies of Spain, a developer of model-execution software. Dr. Liddle is a member of the ACM and the IEEE Computer Society.