SCHEMA MATCHING AND DATA EXTRACTION OVER HTML TABLES

by

Cui Tao

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

September 8, 2003

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Cui Tao

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	David W. Embley, Chair
Date	Stephen W. Liddle
Date	Thomas W. Sederberg

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Cui Tao in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

David W. Embley
Chair, Graduate Committee

Accepted for the Department

David W. Embley
Graduate Coordinator

Accepted for the College

G. Rex Bryce
Associate Dean
College of Physical and Mathematical Sciences

ABSTRACT

SCHEMA MATCHING AND DATA EXTRACTION OVER HTML TABLES

Cui Tao

Department of Computer Science

Master of Science

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 for the case of mostly structured data in the form of HTML tables, based on document-independent extraction ontologies. The solution entails elements of table location and table understanding, data integration, and wrapper creation. Table location and understanding allows us to locate the table of interest, recognize attributes and values, 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 map data of interest 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.

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor, Dr. David W. Embley. Under his guidance, I successfully overcame many difficulties and learned a lot about how to be a good researcher.

Secondly, I would like to thank my other committee members. I thank Dr. Stephen W. Liddle for his unique insight into software implementation and his help in coding. I thank Dr. Thomas W. Sederberg for his time and effort on my thesis.

I would like to thank my husband, Zonghui, for his continued love, patience, understanding and support during this project and my entire endeavor to become more educated. I also want to thank my parents for teaching me how to learn and how to learn well and for encouraging my interest in science since I was a little girl. I thank my sister, Wei and my brother in law, Yue, for encouraging me and supporting my interest in Computer Science.

I thank the National Science Foundation for supporting this research under grant #IIS-0083127.

Last, but not least, I thank all the BYU data-extraction research group members for their support and suggestions on my research.

Contents

List of Figures

Chapter 1

INTRODUCTION

1.1 Background and Related Work

Currently, with the fast development of the Internet, both the amount of useful data and the number of sites on the World Wide Web (WWW) are growing rapidly. The Web is becoming an increasingly useful information tool for computer users. However, there are so many Web pages that no human being can traverse all of them to obtain the information needed. A system that can allow users to query Web pages like a database is becoming increasingly desirable.

The data-extraction research group at Brigham Young University (BYU) has developed an ontology-based querying approach, which can extract data from unstructured Web documents [19]. Although this approach improved the automation of unstructured data extraction, it does not work very well with mostly-structured data. Mostly-structured WWW pages, such as Web pages containing HTML tables, are major barriers to BYU's automated ontology-based data extraction. About 52% of HTML documents include tables [40]. Although some of these tables are only for physical layout, there is still a significant amount of online data that is stored in HTML tables.

The solution to querying data whose sources are HTML tables encompasses elements of (1) table understanding, (2) data integration, and (3) information extraction [27]. Table understanding allows us to recognize attributes and values in a given table. It is a complicated problem [31]. Most research to date uses low-level geometric information to recognize tables [41]. Some of the current geometric modeling techniques for identifying tabular structure use syntactic heuristics such as contiguous space between lines and

columns, position of table lines/columns, and pattern regularities in the tables [48]. Some of the techniques take a segmented table and use the contents of the resulting cells to detect the logical structure of the table [34]. Others use both hierarchical clustering and lexical criteria to classify different table elements [32]. Recent research on table understanding on the Web takes this research area to a higher level. SGML tags provide helpful information of table structures. But poor HTML table encoding, nontraditional use of the HTML tags, the presence of images, etc, all challenge the full exploitation of information contained in tables on the Web [33]. Existing approaches to determining the structure of an HTML table use source page pre-analysis [29, 38], HTML tag parsing [40], and generic ontological knowledge base resolution [52].

The schema-mapping problem for heterogeneous data integration is hard and is, by itself, worthy of study [42]. The problem is to find a *semantic correspondence* between one or more *source schemas* and a *target schema* [20]. 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 [7, 43]. Furthermore, as we shall see in this thesis, 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. 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.

The problem of identifying the text fragments that answer standard questions defined in a document collection is called information extraction (IE) [28]. Some IE techniques are based on machine learning algorithms [45]; others are based on application

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
...	

Figure 1.1: Sample Tables for Target Schema

ontologies [22, 23, 24]. In this thesis, we intend to use ontology-based extractors as an aid to do element-level and instance-level schema matching.

1.2 HTML Table Problems

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 Figure 1.1, whose schema is

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

Figures 1.2 [9], 1.3 [9], and 1.4 [5] show some potential source tables. The data in the tables in Figure 1.1 is a small part of the data that can be extracted from Figures 1.2, 1.3, and 1.4.

[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 [41] is also possible.

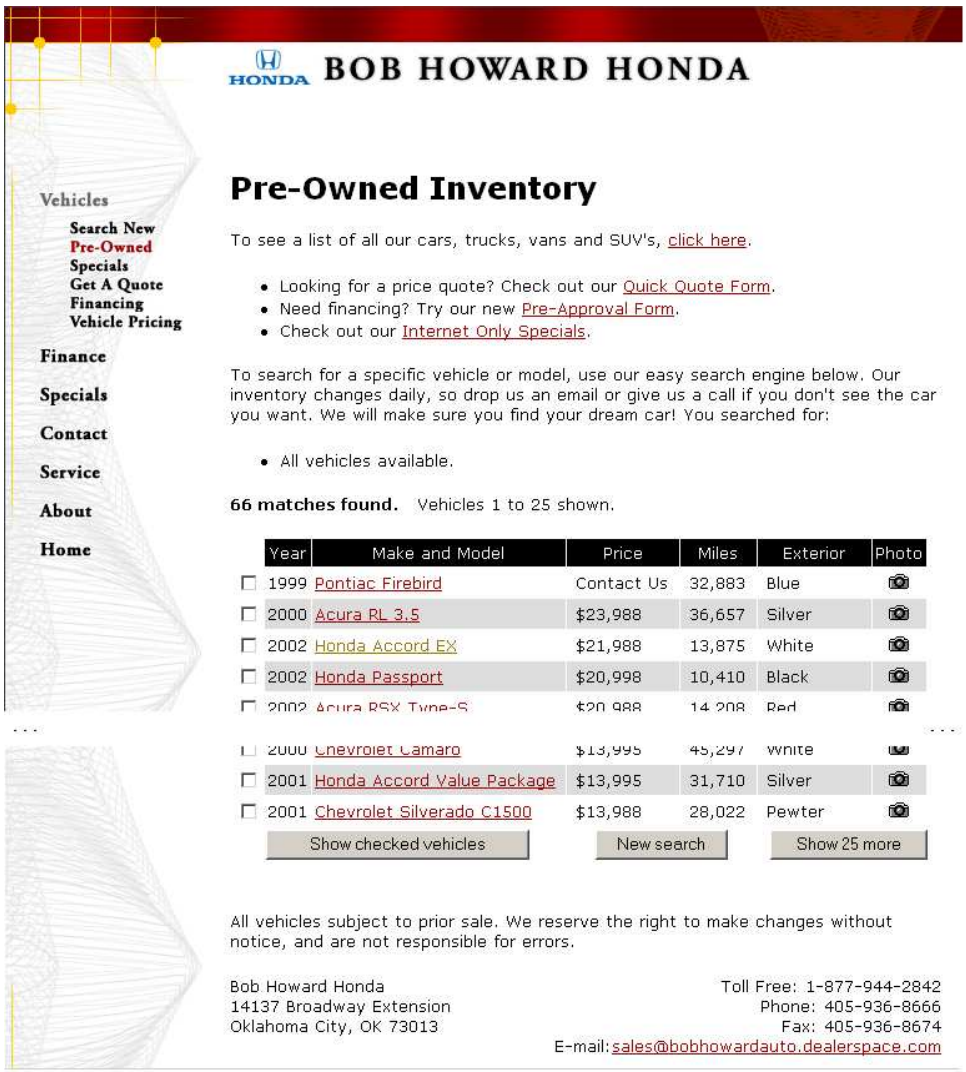


Figure 1.2: Web Page with Table from www.bobhowardhonda.com [9]

1.2.1 HTML Tables—Location Problems

It is easy for a human to locate the table of interest in Figure 1.2. Algorithmically finding the table of interest on an Web page, however, is often nontrivial, even when the system can tell that the page is of interest for the given application [26]. Figure 1.2, for example, presents several challenges for table location.

- *Multiple Panes.* The page in Figure 1.2 has three panes (HTML frames), but we are only interested in the one starting with *Pre-Owned Inventory*.

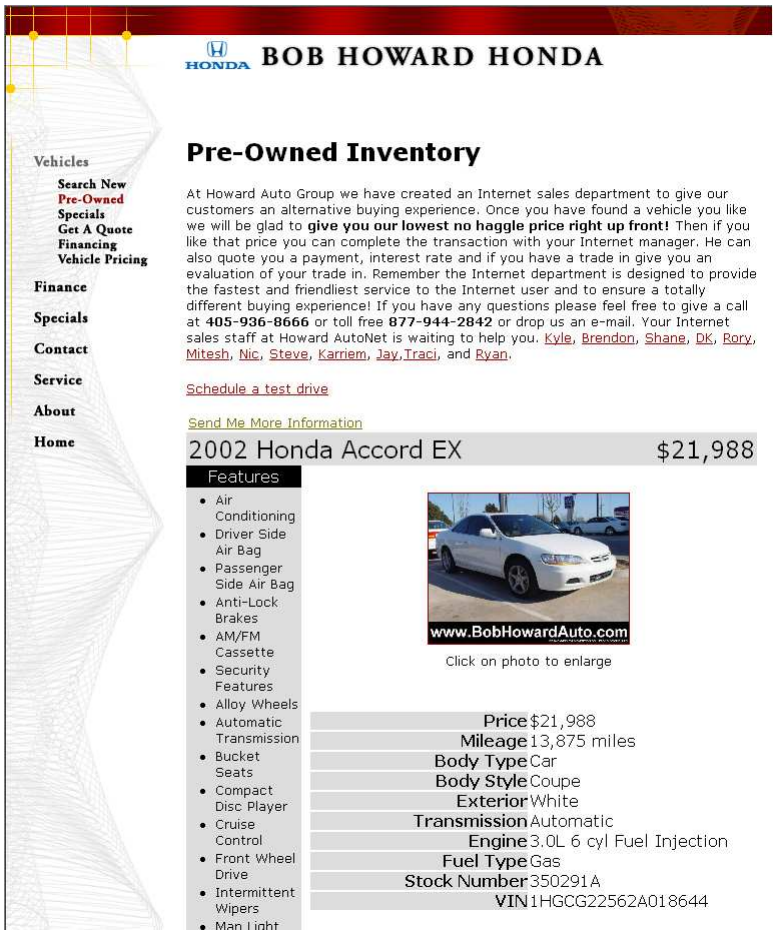


Figure 1.3: Linked Page with Additional Information [9]

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

Figure 1.4: Table from Autoscanada.com [5]

- *Tables for Layout*. In the pane of interest in Figure 1.2, 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 in Figure 1.2 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 nonresponsibility.

- *Tables Displayed Piecemeal*. The table in Figure 1.2 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 Figure 1.3 by clicking on *Honda Accord EX* in the table in Figure 1.2. 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 Figure 1.3 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 Figures 1.2 and 1.3. We list other challenges in the discussion of future work in Chapter 6.

1.2.2 HTML Tables—Extraction Problems

Not only is it easy for a human to find the tables of interest in Figures 1.2 and 1.3, it is also easy for a human to parse the table and determine its meaning. Even with the constraint imposed of needing to match a source table with respect to a fixed target view, such as the one in Figure 1.1, semantic matching is mostly straightforward for a human. It is easy to see that *Year* in the source table in Figure 1.2 as well as *Year* in the source table in

Figure 1.4 map to *Year* in the target table in Figure 1.1. It is also easy to see that although $Make$ and $Model$ in Figure 1.4 match directly with $Make$ and $Model$ in Figure 1.1, we need to split *Make and Model* in Figure 1.2 to match $Make$ and $Model$ in Figure 1.1. It is not as easy, however, to see that both *Exterior* in Figure 1.2 and *Colour* in Figure 1.4 map to *Feature* in Figure 1.1, and it is a little harder to see that we should map the attributes *Auto*, *Air Cond.*, *AM/FM*, and *CD* in Figure 1.4 as values for *Feature* in Figure 1.1, but only for "Yes" values.

Algorithmically sorting out these semantic matches is significantly harder. We encounter the following list of challenges when trying to match source HTML tables in Figures 1.2, 1.3, and 1.4 with the target schema in Figure 4.8. We list other challenges in the discussion of our future work in Chapter 6.

- *Merged Attributes/Values*. *Make* and *Model* are separate attributes in Figure 1.1 but are merged as one attribute in Figure 1.2.

- *Subsets*. *Exterior* in Figure 1.2 and *Colour* in Figure 1.4 contain colors. Colors in the target are a special kind of *Feature* and thus the sets of colors in Figures 1.2 and 1.4 are subsets of the feature values we want for Figure 1.1. Indeed, these are proper subsets since there are also many other feature values in Figures 1.2, 1.3, and 1.4.

- *Synonyms*. *Mileage* in Figure 1.1 and *Miles* in Figure 1.2 have the same meaning, but the attribute names are not the same.

- *Extra Information*. The tables in Figure 1.1 make no request for photographs, which are present in Figure 1.2.

- *Linked Information*. The values for the attribute *Make and Model* are linked to further information. Clicking on *Honda Accord EX* in Figure 1.2 yields the information in Figure 1.3.

- *List Table*. A one-dimensional table and a list are similar in appearance. *Features* in Figure 1.3 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 Figure 1.3 with *Feature* in Figure 1.1.

- *Position of Attributes*. The linked subtable in Figure 1.3 has its attributes in the left column, rather than in the top row.

- *Missing Information*. The schema in Figure 1.1 expects a phone number, but none of the tables in Figures 1.2, 1.3, or 1.4 contains a phone number.

- *Externally Factored Data*. Although no phone number appears in the tables in Figures 1.2 or 1.3, phone numbers do appear in the footer text of the table in Figure 1.2 and in the text above the tables in Figure 1.3. 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 Figures 1.2 and 1.3 appears three times, once under *Price* in Figure 1.2, once as the value for the *Price* attribute in the (vertical) table row in Figure 1.3, and once at the top of the layout table in Figure 1.3. (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 Figure 1.1 expects at most one contact phone number for each vehicle, but there may be several as Figures 1.2 and 1.3 show.

- *Attribute as Value*. In Figure 1.4, 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 Figure 1.1.

In this thesis, we develop a system that can automatically locate the table of interest in a Web page using a set of heuristics and our ontology technology, infer mappings from source table attributes to the target schema, and extract information from source table(s) to the target database.

The rest of this thesis in organized as follows: Chapter 2 introduces our ontology based extraction technology. Chapter 3 discusses our approach to table location. Chapter 4

describes how the system infers mappings from source to target. Chapter 5 introduces and analyzes experimental results. Chapter 6 discusses our conclusions and future work.

Chapter 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 [22]. 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 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.[1] 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.

[1] Page-specific, handwritten wrappers (e.g. the early wrappers produced for TSIMMIS [15]) are not robust. Machine-learning-based wrappers (e.g. [37, 50]) are not robust since new and changed pages must be annotated and learned. Wrappers that automatically infer regular expressions for Web pages (e.g. [18]) 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. [22]) 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.

```
 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.  ...
```

Figure 2.1: Car-Ads Extraction Ontology (Partial)

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. Figure 2 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 nonlexical. Data frames with declarations for constants that can potentially populate the object set represent lexical object sets, and data frames without constant declarations represent nonlexical 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*, *Model*, *Price*, *Feature*, and *PhoneNr* are the remaining lexical object sets in our car-ads application; *Car* is the only nonlexical object set.

We describe the constant lexical objects and the keywords for an object set by regular expressions using Perl-like syntax.[2] 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 nonlexical 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 the initial work with semistructured and unstructured Web pages [22], 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

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

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. We discuss our approach in detail in the next two chapters.

Chapter 3

TABLE LOCATION AND UNDERSTANDING

Many Web sites, especially commercial sites, provide their users with much more than just basic information during browsing. As a result, one HTML page may contain many advertisements, a navigation panel, and other irrelevant sections. These make Web pages difficult to parse. Automatically finding the data-rich sections of the domain of interest from complex Web pages is not an easy task [11]. In this research, we detect a table of interest based on an application-dependent data-extraction ontology and several heuristics that we introduce and discuss in this chapter.

After detecting a table of interest, we then need to understand the structure of this table. In order to understand a table, we first locate the structural components such as table header(s), table factor(s), attributes, and values. We then associate values with their corresponding attributes.

The rest of this chapter is organized as follows: Section 3.1 overviews the basic elements of HTML tables. Section 3.2 describes how to parse an HTML table as a DOM tree. Section 3.3 discusses the method and heuristics we use to locate HTML tables. Section 3.4 introduces our approach to table preprocessing and understanding.

3.1 Overview of HTML Tables

HTML includes element types that represent paragraphs, hypertext links, lists, tables, forms, images, etc [30]. An HTML document usually consists of several HTML elements. Each element starts with a start-tag <TAGNAME> and ends with an end-tag </TAGNAME>. A table in an HTML document is delimited by the tags <TABLE> and </TABLE>. In each table element, there may be tags that specify the structure of the table.

For example, <TH> declares a heading, <TR> declares a row, and <TD> declares a data entry. We cannot, however, count on users to consistently apply these tags as they were originally intended. For example, as Figure 3.1 shows, the attributes *Year*, *Make*, *Model*, *Trim* and etc. are not tagged by <TH>, but by <TD>. Another important issue is that the presence of <TABLE>/</TABLE> tags does not necessarily indicate the presence of a *data table*[1]. For example, in Figure 1.2, <TABLE>/</TABLE> encloses all the information under *Pre-Owned Inventory*. <TABLE>/</TABLE> also encloses the address and contact information at the end of this page. But these two "table" elements are not part of the *real table*, but instead make use of tags for layout. Furthermore, not all *real tables* in a Web page contain information of interest. In addition, even for an HTML file within the specific domain, there may exist one or more *real tables* that present similar information not actually of interest. For example, a Web page for cell phone plans may also contains a table about different cell phones which is not of interest and not suitable for the cell-phone-plan application domain, or a Web site that introduces soccer players (which may be the domain of interest) may also contain a table about coaches.

Given an application domain, our table-location task is to determine if there is a table of interest for this domain and to find the fundamental table of interest in the top-level page and the tables of interest in linked pages, if applicable. 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 Chapter 6.

In order to identify the fundamental table of interest from a given Web document, we first parsed the Web page and represented all the elements in that document with the *document object model* (DOM) [21]. We then isolated all the potential table elements (all elements between <TABLE> begin and </TABLE> end tags) in order to facilitate further

[1]A *data table* here means a table that is for information storage which contains highly structured and database-like information such as table in Figure 1.2.

```
<HTML>
<Table border ="1">
<TR><TH ROWSPAN ="2"><TH COLSPAN ="2">FUEL ECONOMY
    <TH ROWSPAN ="2"><TH COLSPAN ="2">PRICE
    <TH ROWSPAN ="2">ENGINE
<TR><TH>CITY<TH>HWY
    <TH>INVOICE<TH>RETAIL
<TR><TH>2001 Honda Civic DX</TH>
    <TD>39mpg</TD>
    <TD>33mpg</TD>
    <TD>$NL</TD>
    <TD>$12,810</TD>
    <TD>1.7L I4 115HP</TD>
<TR><TH>2001 Honda Civic HX</TH>
    <TD>36mpg</TD>
    <TD>44mpg</TD>
    <TD>$NL</TD>
    <TD>$13,610</TD>
    <TD>1.7L I4 117HP</TD>
<TR><TH>2001 Honda Civic LX</TH>
    <TD>33mpg</TD>
    <TD>39mpg</TD>
    <TD>$NL</TD>
    <TD>$14,910</TD>
    <TD>1.7L I4 115HP</TD>
<TR><TH>2001 Honda Civic EX</TH>
    <TD>32mpg</TD>
    <TD>37mpg</TD>
    <TD>$NL</TD>
    <TD>$16,510</TD>
    <TD>1.7L I4 127HP</TD>
</Table>
</html>
```

Figure 3.1: An Example of an HTML Table

processing. Here we give a short introduction to DOM and how we used DOM to represent an HTML document in order to achieve our goal.

3.2 DOM Representation

A DOM tree is an ordered tree, where each node is either an element node or a text node [17]. An element node has a node name which indicates the HTML tag of this node (such as <TABLE>, <TH> and <LI>) and an ordered list of child nodes (this list can be empty). A text node has no child node and contains only a text string which is the text content of its parent node.

In our use of a DOM tree, we ioslate all the table subtrees. For example, Figure 3.2 is the DOM tree for the table in Figure 3.1. Observe that the leaves of the tree are all text nodes which contain the text values we see through the Web browser in each cell of the table, and the parents of these text nodes are element nodes that give us valuable information about the structure of the table. We can thus analyze the structure and the text contents of this table and decide if this table is a fundamental table of interest by using several heuristics, which we discuss in the next section.

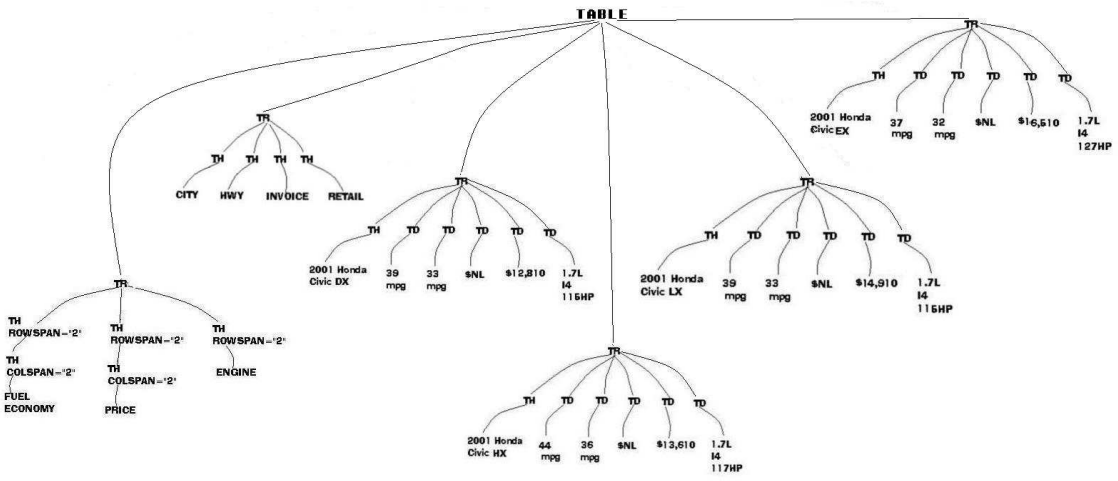


Figure 3.2: DOM Tree of the Table in Figure 3.1

3.3 Table Location Heuristics

Our table-location task is to find both the fundamental table of interest in the top-level page (*top-level Table*) and the table(s) of interest in the linked pages. Since tables of interest that appear in these two kinds of sources usually have different structures and features (although they do, of course, share some common features), we treated them differently by using two sets of heuristics.

3.3.1 Location – Top-Level Tables

In order to create a set of proper heuristics that covers as many cases as possible and maintain high accuracy, we first gathered information about top-level tables by considering several Web pages, which we call a "training set"[2]. Based on the training pages, we found the following features of interest about top-level tables:

1. A table of interest must look like a table to a human observer.
2. A table of interest must have a schema-like[3] row (or column) within the first few rows (or columns).
3. A table of interest must contain enough information of interest (i.e. information that our ontology recognizes).

Based on these features, we developed a set of heuristics for the main table-location task. Our system resolves the problems of finding the main table of interest as follows.

- *Table Size*. The main table must have at least three rows and at least three columns.

  As an example, by this heuristic, the system successfully discarded the table about contact information and address in Figure 1.2.

- *Grid Layout*. We can count the number of data cells in each row in a table. Letting N be the number of rows in the table that has the most common number of data cells

[2]We used this set of pages to help us identify needed heuristics. (This is not a training set in the machine-learning sense.)

[3]Here, "schema-like" means a set of descriptive names that are like attributes in a relational table.

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.

For example, suppose the table of interest in Figure 1.2 (the one in the middle with attribute names *Year*, *Make and Model*, *Price*, etc.) only has 9 columns. For the large table which starts with *Pre-Owned Inventory* and ends with the email address *E-mail:sales@bobhowardauto.dealerspace.com*, there are 9 rows that contain 1 value, 9 rows that contain 6 values and 4 rows that contain 4 values. Therefore, the Grid Layout Measure is $9/(9+9+4) = 0.41 < 2/3$. Thus, the system discards this large 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. For example, we include possible synonyms for each attribute such as *Miles*, *Mi*, and *Odometer* for *Mileage*, and *Manufacture* and *Brand* for *Make*. We also include some popular attributes that commonly appear in source tables but are not an attribute in our target schema because of a different granularity. For example, we include *Vehicle* which could possibly represents several target attributes such as *Make*, *Model*, and *Feature*; *Trim* which could be part of the target attribute *Model*; and *Color* which could be one of the subsets for target attribute *Feature*. We look for a row near the top which contains the most common number of data cells and 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. By using this heuristic, the system can also discard the large table which starts with *Pre-Owned Inventory* and ends with *E-mail:sales@bobhowardauto.dealerspace.com* in Figure 1.2 (no matter how many columns the middle table contains) because it cannot find an attribute row within the top few rows or columns.

- *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 [25] show that the density test should fail only for extremely low percentages, usually below 1%.) By using this heuristic, we ensure that the table contains some information of interest.

3.3.2 Location – Linked-Page Tables

For tables in linked pages, table detection is different. Tables that appear in a linked page usually are either an *attribute-value-pair Table* such as the table under the car picture starting with *Price $21,988* in Figure 1.3, or a *single-attribute Table* starting with *Features* and on the left side of the car picture in Figure 1.3. We use the following heuristics for these tables.

Attribute-Value-Pair Table

- *Table Size*. We do not expect sub-tables 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* 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 Figure 1.3 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.

- *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).

Single-Attribute Table

To find lists like the *Features* list in Figure 1.3, 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.

3.4 Table Preprocessing and Understanding

After detecting tables of interest (both on the top page and on linked pages, if applicable), the next step is to analyze the structure of these tables in order to fully "understand" them (by "understanding", we mean to associate values with their corresponding attributes in the tables).

A top-level table sometimes contains multiple rows (or columns) of attributes, table headers, table factors, or irrelevant information that needs to be ignored during extraction. In order to extract this kind of information correctly, the system must understand the given table properly and preprocess the table according the table structure. In this research, we first try to find the attribute row(s) (or column(s)). According to the attribute position and some other structural information, we can then locate table headers and factors and finally associate attributes and values.

As discussed in the previous section, the system has already detected all the attribute row(s) or column(s) for a table. 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 Figure 1.2, 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.

After determining the position of attributes, the system then determines the structure of the table. In this research we only consider tables with attributes on the top or attributes on the left. Since we can always convert an attributes-on-the-left table to an attributes-on-the-top table or vice versa, in this section we just discuss tables with attributes on the top.

We first list several pre-processing issues the system resolves.

- *Folded Tables*. Figure 3.3 shows an example of a folded table in a linked page (folded tables usually appear more frequently in linked pages than in top-level pages). Sometimes for layout reason or sometimes because a table has so many columns, 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. 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. For a folded table, we unfolded it and appended the the second (and third, ...) folded part(s) to the first one. Thus, after unfolding, the table in Figure 3.3 becomes the table in Figure 3.4.

- *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. Figure 3.5 shows an example of a table that has factors. We add factored values that are below the attribute row to all subsequent rows until the next row of factored values. For values that are above the attribute row, we check if the row right below the attribute row is a factor row. If it is, then we consider all the values above the attribute row as table headers, which we will discuss in the next

Figure 3.3: An Example of Folded Table in a Linked Page (www.jscars.com [35])

Year:	2002
Body	4DR
A/C	Yes
Make:	ISUZU
Color	PEWTER
Motor	6
Model:	AXIOM 4WD
Transmission	A
Miles	26245

Figure 3.4: The Unfolded Table for the Table in Figure 3.3

paragraph. If it is not, then the row right above the attribute row is considered as a factored value. Thus we also add these factored values to the all subsequent rows (except the attribute row) until the next row of factored values. Figure 3.6 shows the altered table for the table in Figure 3.5. 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 Figure 1.2.

Year	Make	Model	Stock	Km's	Sale Price
2001					
	Chrysler	Intrepid	F725	30,000	$17,988.00
	Chrysler	Sebring JXI	22189A	18,000	$30,988.00
	Dodge	Grand Caravan SE Sport	F719	21,000	$26,988.00
2000					
	Chrysler	Cirrus LX	F698	32,000	$13,988.00
	Chrysler	Cirrus LX	F706	38,000	
	Chrysler	Cirrus LX	F610B	36,000	$16,988.00
	Dodge	Dakota Quad Cab Sport	22177A	47,000	$23,988.00
	Honda	Civic SE	22077A	57,000	$15,988.00
	Jeep	TJ Sport	F722	21,000	$22,988.00
1999					
	Chrysler	Intrepid	F756M	46,000	$14,988.00
	Chrysler	Town & Country LTD	F771M	28,000	$32,988.00

Figure 3.5: An Example of an Internal Factor

- *Table Header*. A table header usually appears in a row above the attribute row. It only appears once and is normally short. For example, *Honda Civic* is a table header that factors all the cars in the table in Figure 3.7. Our system considers as table headers those rows that are above the attribute row, marked by only one <TD> or <TH>, and have not already been recognized as table factors. After detecting a table header, the system adds a new column with an empty attribute and places the header

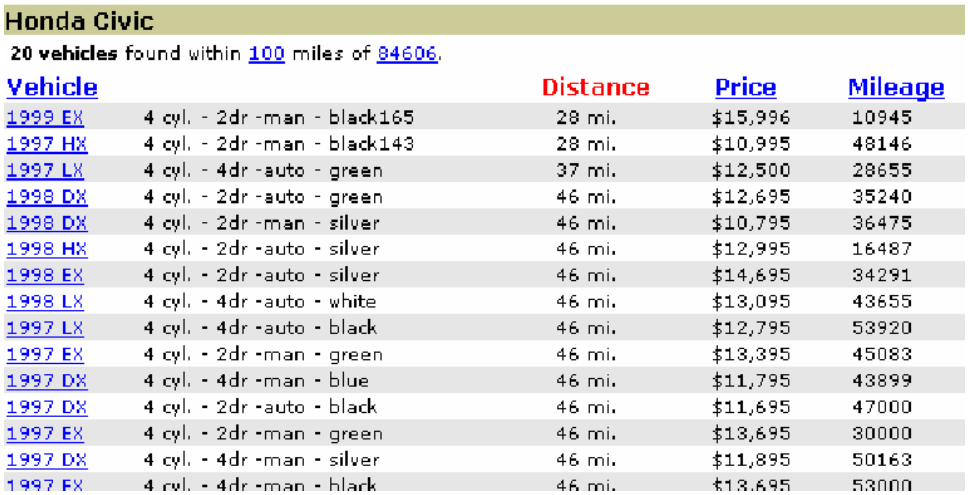
Year	Make	Model	Stock	Km's	Sale Price
2001					
2001	Chrysler	Intrepid	F725	30,000	$17,988.00
2001	Chrysler	Sebring JXI	22189A	18,000	$30,988.00
2001	Dodge	Grand Caravan SE Sport	F719	21,000	$26,988.00
2000					
2000	Chrysler	Cirrus LX	F698	32,000	$13,988.00
2000	Chrysler	Cirrus LX	F706	38,000	
2000	Chrysler	Cirrus LX	F610B	36,000	$16,988.00
2000	Dodge	Dakota Quad Cab Sport	22177A	47,000	$23,988.00
2000	Honda	Civic SE	22077A	57,000	$15,988.00
2000	Jeep	TJ Sport	F722	21,000	$22,988.00
1999					
1999	Chrysler	Intrepid	F756M	46,000	$14,988.00
1999	Chrysler	Town & Country LTD	F771M	28,000	$32,988.00

Figure 3.6: The New Table with Years Distributed to the Value Rows for the Table in Figure 3.5

in value rows of the table. The system repeats this process until all the table headers are processed.

- *Irrelevant Information*. Our heuristics may consider some irrelevant information as table headers or factors. Our heuristics, for example, consider the row *20 vehicles found within 100 miles of 84606* in Figure 3.7, as a table header. It is actually information that is not of interest. Because our system depends on value recognizors within an extraction ontology, our system ignores most of the irrelevant information; therefore, incorrectly distributing irrelevant phrases to value rows rarely affects the final mappings.

After being preprocessed (removing duplicate attribute rows, unfolding, distributing factored values and headers), a table in a top-level page has a format similar to a relation in a relational database. (Figure 3.8 shows an example.) For each record (row) in the original table, we keep track of the tuple ID (the first column in the new table) in order to facilitate the later extraction.



Honda Civic

20 vehicles found within 100 miles of 84606.

Vehicle		Distance	Price	Mileage
1999 EX	4 cyl. - 2dr -man - black165	28 mi.	$15,996	10945
1997 HX	4 cyl. - 2dr -man - black143	28 mi.	$10,995	48146
1997 LX	4 cyl. - 4dr -auto - green	37 mi.	$12,500	28655
1998 DX	4 cyl. - 2dr -auto - green	46 mi.	$12,695	35240
1998 DX	4 cyl. - 2dr -man - silver	46 mi.	$10,795	36475
1998 HX	4 cyl. - 2dr -auto - silver	46 mi.	$12,995	16487
1998 EX	4 cyl. - 2dr -auto - silver	46 mi.	$14,695	34291
1998 LX	4 cyl. - 4dr -auto - white	46 mi.	$13,095	43655
1997 LX	4 cyl. - 4dr -auto - black	46 mi.	$12,795	53920
1997 EX	4 cyl. - 2dr -man - green	46 mi.	$13,395	45083
1997 DX	4 cyl. - 4dr -man - blue	46 mi.	$11,795	43899
1997 DX	4 cyl. - 2dr -auto - black	46 mi.	$11,695	47000
1997 EX	4 cyl. - 2dr -man - green	46 mi.	$13,695	30000
1997 DX	4 cyl. - 4dr -man - silver	46 mi.	$11,895	50163
1997 EX	4 cyl. - 4dr -man - black	46 mi.	$13,695	53000

Figure 3.7: An Example of Table Header (www.autobytel.com [3])

TupleID	*Year*	*Make and Model*	*Price*	*Miles*	*Exterior*	*Photo*
t1	1999	Pontiac Firebird	Contact Us	32,883	Blue	
t2	2000	Acura RL 3.5	$23,988	36,657	Silver	
t3	2002	Honda Accord EX	$21,988	13,875	White	
t4	2002	Honda Passport	$20,988	10,410	Black	
:	:	:	:	:	:	:

Figure 3.8: Top Table After Preprocess for the Table in Figure 1.2

TupleID	*Body Type*	*Body Style*	*Transmission*	*Engine*	*Fuel Type*	*Stock Number*	*VIN*
t1							
t2							
t3	Car	Coupe	Automatic	3.0L 6 cyl Fuel Injection	Gas	350291A	1H...644
t4							
:	:	:	:	:	:	:	:

Figure 3.9: Extended Table of the Information in Figure 1.3

Sometimes, each record (row) in the top-level table may have one or more links that link to other pages. If tables have linked pages, they usually describe detailed information for the corresponding top-level records, and each table describes information for one record, as in Figure 1.3. As described in Section 3.3.2, tables that appear in a linked page usually are either an attribute-value-pair table or a single-attribute table. We consider these two kinds of tables as tables extending over several linked pages. Each table contains values for one record over a set of attributes. Therefore we can collect values for the "extended" table crossing linked pages.

For attribute-value-pair tables, consider the table under the car picture that starts with *Price $21,988* in Figure 1.3 as an example. Figure 3.9 shows the extended table for this example. The information in Figure 1.3 is for the third car in the table in Figure 1.2. Observe that in Figure 3.9, we do not include all the attribute-value pairs that appear in the attribute-value-pair table in Figure 1.3. Attributes *Price*, *Mileage* and *Exterior* are already in the top-level table, therefore the system does not duplicate them in the extended table. Although attribute *Mile* in the top-level table and attribute *Mileage* in the linked-page are not exactly the same, the system can detect these as synonyms with the help of keywords in the ontology. In addition, at the value level, our ontology recognized the same value *13,875* under these two attributes. Therefore, we know that these two attributes describe the same information.

When we encounter an attribute-value-pair table in another linked page, we can add values under their corresponding attribute in the specific position (according to their tupleID). For example, if the *Body Type* for the first car is *Sedan*, we add *Sedan* in the second row (TupleID t1) under attribute *Body Type*. It is possible that the attribute-value

TupleID	*Features*
t3	Air Conditioning
t3	Driver Side Air Bag
t3	Passenger Side Air Bag
t3	Anti-Lock Brakes
:	:

Figure 3.10: An Example Result Table for Single-Attribute Table

pairs differ (usually only slightly) on different linked pages. If there is a new attribute that is not included in the extended table, we add this new attribute in the extended table and add the corresponding value in its proper position under this new attribute. We repeat this process until all the information in attribute-value-pair tables in all linked pages are considered.

The table under *Features* in Figure 1.3 is a single-attribute table. It, by itself, can be converted into a column in a relation in which each row has the same tuple identifier (because every value pertains to the same object.). For the example in Figure 1.3, our system transforms it into the table in Figure 3.10.

Hence, our table recognizing system transforms the structured information (top-level table, attribute-value-pair table and single-attribute table) in both the top-level pages and linked pages into a format similar to relations in a relational database. In the next chapter, we discuss how to map source attributes to target attributes based on the information we have.

Chapter 4

MAPPING INFERENCE

After preprocessing and understanding the table, the system has converted the structured information into one or more table structures that are similar to relations in a relational database. We then can infer mappings from these source "relations" to the target object sets in our extraction ontology (i.e. infer a mapping from source attributes to target attributes). We infer mappings in two steps: (1) generate and adjust attribute-value pairs in preparation for mapping recognition and (2) use patterns of recognized attributes and values to infer mappings.

4.1 Generate and Adjust Attribute-Value Pairs

In one column in a source table, the attribute names the type of values under it. Although we can sometimes determine the type of a value without an attribute, the attribute often provides valuable context information for ontology extraction. Therefore, pairing a value with its corresponding attribute may help our ontology recognize more information from the table. For example, the attribute-value pairs we form for the attribute-value-pair table in Figure 3.4 is:

{*Year: 2002, Body: 4DR, A/C: Yes, Make: ISUZU, Motor: 6; Model: AXIOM 4WD, Transmission: A, Miles: 26245*}.

Observe that in this example, we may not be able to determine the type of some values without their attributes such as values in *Motor: 6* and *Transmission: A*. The digit *6* and letter *A* may not provide enough information by themselves without their corresponding

attributes. With the help of attributes, however, we can determine the type and meaning of those values.

Another interesting issue our example shows is *A/C: Yes* — an attribute with a Boolean value. We process Boolean values by replacing them with attribute-name values. For our running example, the adjusted attribute-value pairs become {*Year: 2002, Body: 4DR, A/C, Make: ISUZU, Motor: 6; Model: AXIOM 4WD, Transmission: A, Miles: 26245*}. Here, the Boolean-valued attribute-value pair ⟨*A/C*: *Yes*⟩ has become *A/C*, meaning the car has A/C (air conditioning). If there is an attribute-value pair with value *no*, we simply replace it with an empty string, meaning the car does not have the feature indicated by the attribute. For example, the first value row of the table in Figure 4.1a would be transformed to {*Make: ACURA, Model: legend, Yr: 1992, Colour: Grey, Price: $9500, Auto, AM/FM* }. Note that the attributes *Air Cond.* and *CD* disappeared because this car does not have these features.

When attribute names are the values and the values are Boolean indicators (e.g. Yes/No, True/False, 1/0, cell checked or empty, $\surd$ or x), we need to decide what the Boolean indicators mean. We have a dictionary of Boolean indicators that defines potential meanings for each indicator. For an indicator like *Yes* or *No*, we can know for sure what they mean. Some other indicators, however, could have different meanings in different situations. For example, an *x* could mean *yes* when an empty cell means *no*; it also could mean "no" when a $\surd$ means *yes*. If a Boolean value is in the dictionary, we check the potential meaning of the indicator. If the indicator has only one meaning, we assign the opposite meaning to the other Boolean value (if any) in the same column. If it has more than one meaning, we then need to check the other Boolean value that appears in the same column. For example, if we encounter an *x*, and it has two meanings: *yes* and *no*, in the dictionary, we then check other operators in the same column. For example, we found a $\surd$, which has only one meaning, *yes*, in the dictionary. We then can decide the *x* here does not mean *yes*, but means *no*. Based on this heuristic, we can understand what a pair of Boolean indicators mean as long as we can find them in our dictionary.

Make	*Model*	*Yr*	*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

(a)

Make	*Model*	*Yr*	*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	

(b)

Figure 4.1: A Table that has Boolean Values and the Table Transformed by the β Operator

After understanding the meanings of the Boolean indicators, we can transform them 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 Figure 4.1a to the table in Figure 4.1b.

4.2 Infer Mapping

In this section, we discuss how to infer mappings from the source-table attributes to our target schema. We first describe the patterns and regularity in the source table that we need to recognize and define the threshold we used to recognize those patterns. We then discuss how to infer mappings according to those patterns with the help of standard relational algebra operators and some extended relational algebra operators.

4.2.1 Pattern Recognition

Our system represents a source table (top-level table as well as tables in linked pages) as one or more relational structures that are similar to relations in a relational

database. The values in each relational structure are formatted as attribute-value pairs and each Boolean value is transformed into a proper value. Based on this source representation, we infer mappings by using our ontology-extraction technology. As we mentioned in Chapter 2, our extraction ontology is source independent, which means we do not have to generate a new ontology when a new source document is encountered. It is hard, however, to guarantee that our ontology covers everything. We do not expect our system to recognize all the source values. Instead, our purpose is to find data regularity and infer mappings depending on the recognized results.

Because of the special layout structured tables have, we know all the values under a single attribute in a source table should be extracted to a same attribute or set of attributes in the target schema. Given the recognized extraction and its regularity in the source document, our system can measure how many values under a source attribute are actually extracted to a particular set of target attributes. If the number is greater than a threshold, the system can infer mappings between source and target attributes according to the regularity observed. Given a set of mappings, the system can then extract data into the target database, including not only the recognized values, but also all other values that fit the pattern. By doing so, we are likely to be able to increase both the precision and recall of the extraction. We talk about the experimental results in detail in Chapter 5.

In order to infer as many correct mappings as possible and, at the same time, avoid unnecessary incorrect mappings, an appropriate threshold is important. A high threshold would result in the low mapping rates (low recall) while a low threshold would result in many error mappings (low precision). In this thesis, we define the threshold to be the Golden Mean, also called the "divine proportion"[2]. This constant can be calculated by $(\sqrt{5}$ -1)$/2 \approx 0.618$. The term "Golden Mean" is derived from Horace's Latin translation of "aurea mediocratas," which means a sensible way of doing things or the avoidance of extremes. In mathematics and real life, the Golden Mean often represents a balanced threshold [2]. Therefore, in our research, we also use this ratio as our threshold.

```
1.  Make matches [10]
2.      constant
3.              { extract "\bacura\b"; },
4.              { extract "\balfa((\s*|-)romeo)?\b"; },
5.              { extract "\bamc\b"; },
6.              { extract "\bam(\s*|-)general\b"; },
7.              { extract "\baudi\b"; },
8.              { extract "\bbentley\b"; },
9.              { extract "\bbertone\b"; },
10.             { extract "\bbmw\b"; },
11.             { extract "\bbuick\b"; },
12.             { extract "\bcad(illac)?\b"; },
13.             { extract "\bchev(y|rolet)?\b"; },
14.             { extract "\bchrysler\b"; },
15.  ...
```

Figure 4.2: Data Frame for the Make Object Set in Car-Ads Extraction Ontology (Partial)

4.2.2 Mapping Inference

Our purpose is to match source table attributes with target attributes (object sets in the ontology). As discussed in Chapter 2, an extraction ontology contains information about object sets, relationships, and data frames. Each object set has 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. In order to find mappings from source attributes to the object sets, we apply each data frame to each column in the source table to see if we recognize enough values, so that the percentage of recognized values is greater than the threshold.

Figure 4.2 shows a partial data frame for object set *Make* in our car ontology. Now let us see if there is any attribute in Figure 4.1 from which we can map this object set. We attempt to recognize values in each column in Figure 4.1 with regular expressions in the *Make* data frame, and we keep track of the number recognized. In our example, for the first column we recognized 100% (e.g. *ACURA* matches using Line 3 in Figure 4.1, *AUDI* matches using Line 7, *BMW* matches using Line 10 and *CHEVROLET* matches using Line 13). For the rest of columns, however, we recognized nothing. Therefore, we can infer

Make	*Model*
Honda	Civic
Nissan	Sentra

Figure 4.3: Columns Added to the Table in Figure 4.4 by the δ Operator

a mapping from *Make* in the source table to *Make* in the target ontology. This is a direct mapping. Similarly, we can obtain mappings from *Model* to *Model*, *Yr* to *Year*, and *Price* to *Price*. The mapping from *Yr* to *Year*, however, is not a direct mapping, because we need a renaming operator ρ.

Except for simple renaming, indirect mappings are more complicated, and the system needs the help of more operators. As we can see, values in the 6^{th}-9^{th} columns in Figure 4.1 should all go under a single target attribute *Feature*. In this case, the system needs to gather them together and consider each of them as a separate value under one target attribute. We gather values together with the union operator $\cup$.

Another case is recognizing a value that should be split. For example, we detect that all the values under attribute *make/model* in Figure 4.4 are merged and need to be mapped separately to *Make* and *Model* in the target schema as Figure 4.5 shows. We can divide values into smaller components with a δ operator which we introduce here. We define $\delta^{A}_{B_1,...,B_n} r$ to mean that each value v for attribute A of relation r is split into $v_1, ..., v_n$, one for each new attribute $B_1, ..., B_n$ respectively. Associated with each B_i is a procedure p_i that defines which part of v becomes v_i. In this thesis we specify each procedure p_i by regular expressions similar to those defined for extraction ontologies in Figures 2 and 4.2. The result of the δ operator is r with n new attributes, B_1, ..., 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 $\delta^{Make/Model}_{Make,Model} T$, where T is the table in Figure 4.4, 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 remanning characters in the string after the first space. This operation adds the two columns in Figure 4.3 to the table in Figure 4.4.

year	*make/model*	*color*	*bodytype*
1999	Honda Civic	Green	4 dr sedan
1998	Nissan Sentra	grey	2 door coupe

Figure 4.4: An Sample Source Table

Car	Year	Make	Model	Mileage	Price	PhoneNr
0001	1999	Honda	Civic			
0002	1998	Nissan	Sentra			

Car	Feature
0001	Green
0001	4 dr
0001	sedan
0001	grey
0002	2 door
0002	coupe

Figure 4.5: Extracted Result from the Table in Figure 4.4

If we consider the right-most table in Figure 4.5 as the source table and the schema in table in Figure 4.4 as the target schema, we encounter another issue. Values under the a same attribute need to be associated with different target attributes. In our example, *Green* and *grey* under the source attribute *Feature* associate with the target attribute *color*, and other values under *Feature* associate with *features* in target. In this case, we need to apply a selection operator σ. Here the σ operator is not standard because it may have a regular expression as an argument. The selection operator $\sigma_{A\sim e}r$ selects those rows in a relation r whose values under attribute A contain a string recognized by regular expression e.

Sometimes, the values of interest are scattered in unstructured or semistructured documents. For this kind of 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, $\epsilon_{PhoneNr}P$ extracts *1-877-944-2842* from the unstructured text in page P in Figure 1.3 and returns it as the single-attribute, single-tuple, constant relation {*PhoneNr*: *1-877-944-2842*}. We can use the ϵ operator in conjunction with a natural join to add a column of constant values to a table. For example,

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

Figure 4.6: Application of the γ Operator to Table T_1 Yielding Table T_2

assuming the phone number *1-877-944-2842* appears in page P with the table in Figure 1.2, which indeed it does, we could apply $\epsilon_{PhoneNr} P \bowtie T$ to add a column for *PhoneNr* to table T in Figure 1.2.

Figure 4.6 shows another case we need to handle. Values under *Model* and *Trim* in the source table (T_1 in Figure 4.6) should go together as a single value under *Model* in the target.[1] If we can recognize which values we need to merge, we can merge them with a γ operator which we introduce here. Syntactically, we write $\gamma_{B \leftarrow A_1+...+A_n} r$ 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 $\gamma_{Model\ with\ Trim \leftarrow Model+\texttt{" "}+Trim} T_1$ which converts Table T_1 in Figure 4.6 to Table T_2.

Sometimes, one mapping may involve more than one operator. We can, for example, take a union of *color* and *body type* in Figure 4.4 to form part of the set for *Feature* in Figure 4.5. After adding needed projection , split, and renaming operations, this union is $\rho_{color \leftarrow Feature} \pi_{color} T \cup \rho_{bodytype_1 \leftarrow Feature} \pi_{bodytype_1} \delta^{bodytype}_{bodytype_1, bodytype_2} T \cup \rho_{bodytype_2 \leftarrow Feature} \pi_{bodytype_2} \delta^{bodytype}_{bodytype_1, bodytype_2} T$, where T is the table in Figure 4.4.

Now that we have the operators we need, we can give examples. Figure 4.7 gives the mapping from the source table in Figure 4.1a to the target schema in Figure 4.8. 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. Figure 4.9 gives the mapping for the car ads from the site for Figures 1.2 and 1.3. Observe that we have split the makes and

[1]Currently, our system does not handle this case. To implement value merging, we should apply data-frame value recognizers to all possible concatenations of values in unrecognized columns.

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

Figure 4.7: Inferred Mapping from Source Table T in Figure 4.1a to the Target Table in Figure 4.8

Car	*Year*	*Make*	*Model*	*Mileage*	*Price*	*PhoneNr*
0001	1992	ACURA	legend		$9500	
0002	2000	AUDI	A4		$34,500	
0003	1985	BMW	325e		$2700.00	
				...		
0005	1999	Pontiac	Firebird	32,883	Contact Us	1-877-944-2842
0006	2000	Acura	RL 3.5	36,657	$23,988	1-877-944-2842
				...		

Car	*Feature*
0001	grey
0001	Auto
0001	AM/FM
	...
0005	Blue
0005	Power Steering
	...
0006	Silver
0006	Power Brakes
	...

Figure 4.8: Sample Tables for Target Schema

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.

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	$\epsilon_{PhoneNr} P \bowtie T$
Feature	$\rho_{Exterior \leftarrow Feature} \pi_{Exterior} T$ $\cup\ \rho_{Body\ Type \leftarrow Feature} T$' $\cup\ \rho_{Body\ Style \leftarrow Feature} T$' $\cup\ \rho_{Transmission \leftarrow Feature} T$' $\cup\ \rho_{Engine \leftarrow Feature} T$' $\cup\ \rho_{Fuel\ Type \leftarrow Feature} T$' $\cup\ \rho_{Features \leftarrow Feature} T$''

Figure 4.9: Inferred Mapping from the Source Tables T in Figure 1.2, T' in Figure 3.9, and T'' in Figure 3.10 and from P, the page in in Figure 1.2, to the Target Table in Figure 4.8

Chapter 5

EXPERIMENTAL ANALYSIS

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 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 (Figures 1.2 and 1.3 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.[1] 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

[1] Although expert judgement for tables can sometimes be hard [31], establishing correctness results for car-ads for our target table was not difficult.

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 generating 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 matches, in the sense that the attributes in the source and target schemas were identical, and 175 (59.1%) were 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*.

Discussion—Car Advertisements

Locating the correct table and understanding it properly is not a trivial problem. As we discussed in Chapter 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 3,000 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-U.S. 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 U.S. car ads, and so non-U.S. ads introduced car 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.

5.1.2 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 U.S. cell-phone sales domain. Figure 5.1 shows part of a top page of a typical cellular-phone application, and Figure 5.2 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

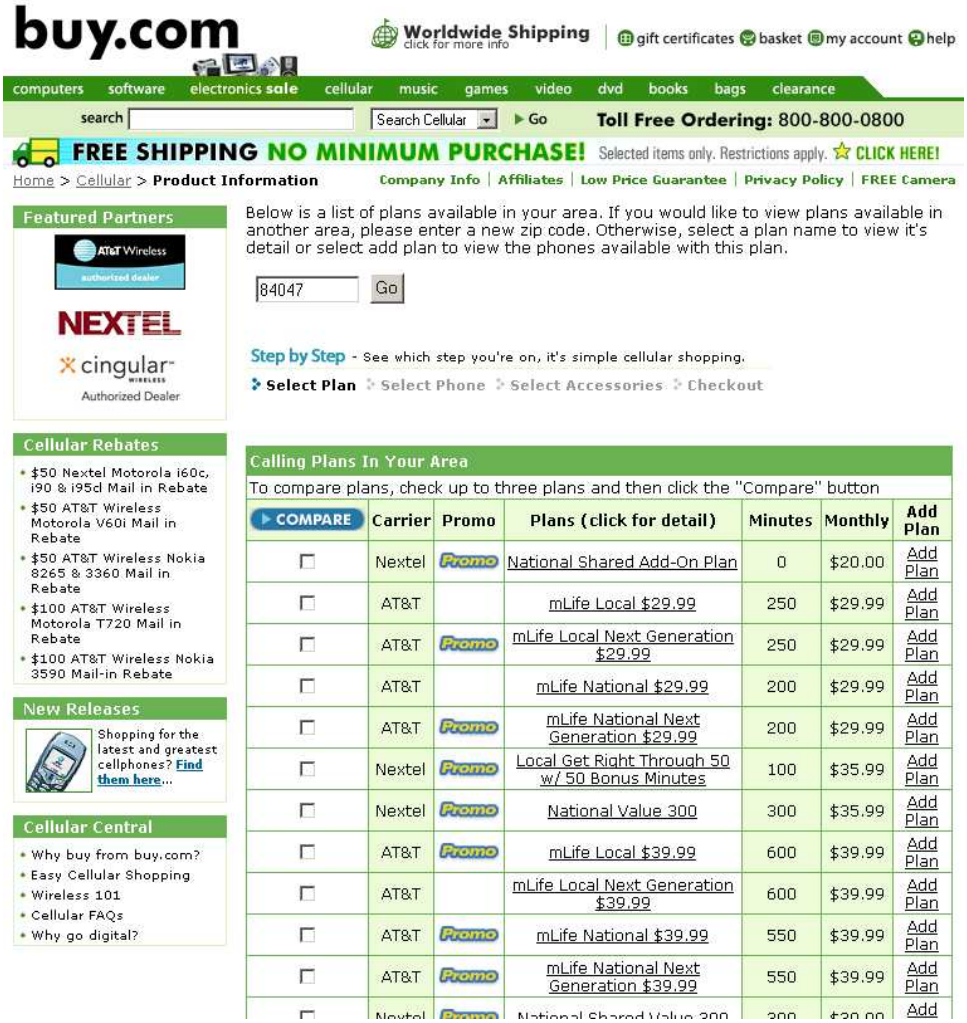


Figure 5.1: Web Page with Table of Cell Phones from Buy.com

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

Figure 5.2: Linked Page with Additional Cell Phone Information from Buy.com

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 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.

Chapter 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In this thesis, we designed and implemented a system which can automatically locate HTML tables for a specified application domain and then extract information of interest from the located tables by inferring mappings from source tables to a target schema. 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.

6.2 Future Work and Interesting Unresolved Table Problems

As a next step of the table location and understanding problem, we would like to improve our approach and make it handle more cases such as some of the problems listed in Appendix A. We also do not want to just focus on HTML tables. To locate and understand tables in other formats is also an interesting problem.

Tables also provide useful information for data frames. After we infer a mapping, we know that all the data under one source attribute should map to one target object. Therefore, we could update our ontology with values found in the source table.

Beyond table location and understanding, we recognize that many tables are behind forms, in the so-called “hidden Web” [39, 49]. 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. 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 can use techniques developed previously [22] to extract the data. Further, we also plan to add our work to the data-extraction work done previously [19].

Bibliography

[1] www.ads4autos.com, July 2003.

[2] P. G. Anderson. Multidimensional golden means. In *Applications of Fibonacci numbers, Vol. 5 (St. Andrews, 1992)*, pages 1–9. Kluwer Academic Publishers Group, Dordrecht, 1993.

[3] Autobytel.com, Spring 2001.

[4] www.autoexecutives.com, July 2003.

[5] autoscanada.com, Summer 2001.

[6] www.autoweb.com, July 2003.

[7] J. Biskup and D.W. Embley. Extracting information from heterogeneous information sources using ontologically specified target views. *Information Systems*, 28(3):169–212(44), May 2003.

[8] www.bobandersonford.com, July 2003.

[9] www.bobhowardhonda.com, May 2003.

[10] www.bostoncellular.com, July 2003.

[11] D. Buttler, L. Liu, and Calton Pu. A fully automated object extraction system for the world wide web. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDC'01)*, Phoenix (Mesa), Arizona, April 2001.

[12] www.carbuyer.com, July 2003.

[13] www.carcast.com, July 2003.

[14] www.cardirect.com, July 2003.

[15] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *Information Processing Society of Japan Conference (IPSJ)*, pages 7–18, Tokyo, Japan, October 1994.

[16] www.classyauto.com, July 2003.

[17] W.W. Cohen, M. Hurst, and L.S. Jensen. A flexible learning system for wrapping tables and lists in html documents. In *Proceedings of International World Wide Web Conferences (WWW02)*, pages 232–241, Honolulu, Hawaii, May 2002.

[18] V. Crescenzi, G. Mecca, and 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)*, pages 109–118, Rome, Italy, September 2001.

[19] Homepage for BYU data extraction research group. URL: http://www.deg.byu.edu.

[20] A. Doan, P. Domingos, and 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)*, pages 509–520, Santa Barbara, California, May 2001.

[21] The w3c architecture domain. http://www.w3.org/dom/.

[22] D.W. Embley, D.M. Campbell, Y.S. Jiang, S.W. Liddle, D.W. Lonsdale, Y.-K. Ng, and R.D. Smith. Conceptual-model-based data extraction from multiple-record Web pages. *Data & Knowledge Engineering*, 31(3):227–251, November 1999.

[23] D.W. Embley, D.M. Campbell, Y.S. Jiang, Y.-K. Ng, R.D. Smith, S.W. Liddle, and D.W. Quass. A conceptual-modeling approach to extracting data from the Web. In *Proceedings of the 17th International Conference on Conceptual Modeling (ER'98)*, pages 78–91, Singapore, November 1998.

[24] D.W. Embley, D.M. Campbell, S.W. Liddle, and R.D. Smith. Ontology-based extraction and structuring of information from data-rich unstructured documents. In *Proceedings of the 7th International Conference on Information and Knowledge Management (CIKM'98)*, pages 52–59, Washington D.C., November 1998.

[25] D.W. Embley, Y.S. Jiang, and Y.-K. Ng. Record-boundary discovery in Web documents. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD'99)*, pages 467–478, Philadelphia, Pennsylvania, May/June, 1999.

[26] D.W. Embley, Y.-K. Ng, and L. Xu. Recognizing ontology-applicable multiple-record web documents. In *Proceedings of the 20th International Conference on Conceptual Modeling (ER2001)*, pages 555–570, Yokohama, Japan, November 2001.

[27] D.W. Embley, C. Tao, and S.W. Liddle. Automatically extracting ontologically specified data from HTML tables with unknown structure. In *Proceedings of the 21th International Conference on Conceptual Modeling (ER2002)*, pages 322–337, Tampere, Finland, October 2002.

[28] D. Freitag. Information extraction from html: Application of a general machine learning approach. In *Proceedings of the 15th National Conference on Artificial Intelligence and 10th Innovative Applications of Artificial Intelligence Conference*, pages 517–523, Madison, Wisconsin, January, 1998.

[29] J. Hammer, H. Garcia-Molina, J. Cho, A. Crespo, and R. Aranha. Extracting semistructured information from the Web. In *Proceedings of the Workshop on Management of Semistructured Data*, Tucson, Arizona, May 1997.

[30] HTML 4.01 specification. http://www.w3.org/TR/html401/, December 1999.

[31] J. Hu, R. Kashi, D. Lopresti, G. Nagy, and G. Wilfong. Why table ground-truthing is hard. In *Proceedings of the Sixth International Conference on Document Analysis and Recognition*, pages 129–133, Seattle, Washington, September 2001.

[32] J. Hu, R. Kashi, D. Lopresti, and G. Wilfong. Table structure recognition and its evaluation. In P.B. Kantor, D.P. Lopresti, and J. Zhou, editors, *Proceedings of Document Recognition and Retrieval VIII*, volume SPIE-4307, pages 44–55, San Jose, California, January 2001.

[33] M. Hurst. Layout and language: challenges for table understanding on the web. In *Proceedings of the First International Workshop on Web Document Analysis (WDA2001)*, pages 27–30, Seattle, Washington, September 2001.

[34] M. Hurst and S. Douglas. Layout and language: Preliminary investigations in recognizing the structure of tables. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR'97)*, pages 1043–1047, Ulm, Germany, August 1997.

[35] www.jscars.com, May 2003.

[36] www.kernautos.com, July 2003.

[37] N. Kushmerick, D.S. Weld, and R. Doorenbos. Wrapper induction for information extraction. In *Proceedings of the 1997 International Joint Conference on Artificial Intelligence*, pages 729–735, NAGOYA, Aichi, Japan, 1997.

[38] K. Lerman, C.A. Knoblock, and S. Minton. Automatic data extraction from lists and tables in web source. In *Proceedings of Automatic Text Extraction and Mining Workshop (ATEM-01)*, Seattle, Washington, August 2001.

[39] S.W. Liddle, S.H. Yau, and 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)*, pages 106–119, Yokohama, Japan, November 2001.

[40] S. Lim and Y. Ng. An automated approach for retrieving heirarchical data from HTML tables. In *Proceedings of the Eighth International Conference on Informaiton and Knowledge management (CIKM'99)*, pages 466–474, Kansas City, Missouri, November 1999.

[41] D. Lopresti and G. Nagy. Automated table processing: An (opinionated) survey. In *Proceedings of the Third IAPR Workshop on Graphics Recognition*, pages 109–134, Jaipur, India, September 1999.

[42] J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 49–58, Rome, Italy, September 2001.

[43] R. Miller, L. Haas, and M.A. Hernandez. Schema mapping as query discovery. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB'00)*, pages 77–88, Cairo, Egypt, September 2000.

[44] www.motorzones.com, July 2003.

[45] I. Muslea. Extraction patterns for information extraction tasks: survey. In *Proceedings of American Association for Artificial Intelligence*, pages 1–6, Orlando, Florida, July 1999.

[46] www.palmbeachclassifieds.com, July 2003.

[47] www.peakcellular.com, July 2003.

[48] P. Pyreddy and W.B. Croft. TINTIN: A system for retrieval in text tables. In *Proceedings of the 2nd ACM International Conference on Digital Libraries*, pages 193–200, Philadelphia, Pennsylvania, July 1997.

[49] S. Raghavan and 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.

[50] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1–3):233–272, 1999.

[51] www.usedcars.com, July 2003.

[52] M. Yoshida, K. Torisawa, and J. Tsujii. A method to integrate tables of the world wide web. In *Proceedings of the International Workshop on Web Document Analysis (WDA 2001)*, pages 31–34, Washington, DC, September 2001.

Appendix

Appendix A

INTERESTING UNRESOLVED TABLES

CARS FOR SALE

Search Results Exceeded Max : 1 - 25 of first 500 cars **Search Again**

Our inventory is updated daily in realtime, so if you don't see exactly what you want, please check back soon. Or tell the world about the vehicle you want by placing a Free Vehicle Wanted ad.

Platinum Listings:

1993 Buick Lesabre Green - 880,000 miles- $500

PHOTO	SELLER	DESCRIPTION
No Image	**Private Seller**	Front end damage cars good for parts or repair (500.00 to 1,000 In repairs) ... *more*

1993 Buick Lasabler Green - 880,000 miles- $700

PHOTO	SELLER	DESCRIPTION
No Image	**Private Seller**	front end damage about $800.00 -1,000 in parts needed ... *more*

1994 Buick Regal Green - 134,108 miles- $2,210

PHOTO	SELLER	DESCRIPTION
	New Jersey State Auto Auction	This Regal Custom model vehicle has a 3.8 V-6 Multi-Pt Fuel Injection and Automatic With Overdrive. It includes Power Steering, Power Brakes, Power Door Locks, Power Windows, Power Driver's Seat, AM/ ... *more*

Figure A.1: Table with Complicated Structure (Multiple Schemas and Long Factors) [51]

Dodge Neon

We found **11 vehicles** within **25 miles** of **84604** that are between **$0** and **$999999**

My Favorites Modify Search Criteria

Sort By: Distance GO Page 1 of 2

	Vehicle	Distance	Mileage	Price
	2003 Neon Neon 4 Cylinders - 4 Door - Automatic	24 mi.	5,023 mi.	See Dealer
	2003 Neon Neon 4 Cylinders - 4 Door - Automatic	24 mi.	0 mi.	See Dealer
	2003 Neon Neon 4 Cylinders - 4 Door - Automatic	24 mi.	6,793 mi.	See Dealer
	2003 Neon Neon 4 Cylinders - 4 Door - Automatic	24 mi.	15,706 mi.	See Dealer
	2003 Neon Neon 4 Cylinders - 4 Door - Automatic	24 mi.	4,959 mi.	See Dealer
	2003 Neon Neon 4 Cylinders - 4 Door - Automatic	24 mi.	6,274 mi.	See Dealer
	2003 Neon Neon 4 Cylinders - 4 Door - Automatic	24 mi.	8,375 mi.	See Dealer
	2003 Neon Neon 4 Cylinders - 4 Door - Automatic	24 mi.	7,713 mi.	See Dealer
	2003 Neon Neon 4 Cylinders - 4 Door - Automatic	24 mi.	6,380 mi.	See Dealer
	2001 Neon Highline	24 mi.	35,143 mi.	See Dealer

Figure A.2: Table with Complicated Structure (Each Record Takes Multiple Rows) [6]

Year	Vehicle	Distance To Seller	Mileage	Price	Photo
Seller		New/Used	New Ad	Color	
	2000 Honda ACCORD COUPE			$12,900	 Stock Photo
	PB Post	Used			
	2002 Honda ACCORD COUPE			$18,994	 Stock Photo
	PB Post	Used			
	2000 Honda ACCORD COUPE			$12,900	 Stock Photo
	PB Post	Used			

Figure A.3: Table with Complicated Schema (Attributes Take Multiple Rows) [46]

Search: Ford in UT

Email Notifier

Found 16 Cars

Price a New One

Used Car Locator

make	model	year	mileage	city	state	price
Ford	Tempo	1984	104000	Salt Lake	UT	$300
Ford	Mustang LX	1988	77400	Ogden	UT	$1800
Ford	Mustang	1966	--	Centerville	UT	$15000
Ford	British	1968	10000	Tooele	UT	$10000
Ford	Mustang Convertible	1973	2178	St George	UT	$23750
Ford	Taurus	1993	169000	Draper	UT	$1500
Ford	Escort GT	1991	106000	Provo	UT	$3000
Ford	TEMPO GL	1992	120000	SLC	UT	$2300
Ford	Super Deluxe	1942	--	Sandy	UT	$21000
Ford	Duluxe	1941	--	Sandy	UT	$2400
Ford	Mustang	1997	65000	Salt Lake City	UT	$9900
Ford	Mustang LX	1991	82000	Salt Lake City	UT	$4200
Ford	Taurus	1998	60999	Ogden	UT	$10000
Ford	Taurus	1998	70000	Ogden	UT	$10000
Ford	Focus ZTS	2001	23000	Draper	UT	$12995
Ford	ESCORT ZX2	1999	18100	Orem	UT	$10900

Figure A.4: Table with Image Attributes [12]

Photo	Information	Mileage	Internet Price
	**Vehicle:** Used 1998 FORD CONTOUR SE **Color:** MAROON **Engine:** 4 CLY **Trans:** Automatic Front Wheel Drive **Vehicle Type:** Car **Stock #: 9453**	52,562 Mi	Ask Dealer
	**Vehicle:** Used 1998 FORD F-150 4X4 SUPER CAB XLT **Color:** WHITE **Engine:** 4.6L V8 **Trans:** Automatic 4 Wheel Drive **Vehicle Type:** Truck **Stock #: 98649**	66,851 Mi	Ask Dealer
	**Vehicle:** Used 1998 FORD RANGER RC 4X4 XL **Color:** WHITE **Engine:** 4CLY **Trans:** Manual 4 Wheel Drive **Vehicle Type:** Truck **Stock #: 32948**	115,850 Mi	Ask Dealer
	Vehicle: Used 1998 FORD WINDSTAR GL **Color:** MAROON **Engine:** V6 **Trans:** Automatic Front Wheel Drive **Vehicle Type:** MiniVan **Stock #: D84062**	61,451 Mi	Ask Dealer
	**Vehicle:** Used 2001 CHEVROLET CK2500 HD LS **Color:** WHITE **Engine:** 6.0 v8 **Trans:** Automatic 4 Wheel Drive **Vehicle Type:** Truck **Stock #: 45484**	39,332 Mi	Ask Dealer

Figure A.5: Tables in Table [8]

Figure A.6: Table with Irrelevant Images Inside [14]

Figure A.7: Each Record in the Top-Level Table Contains an Attribute-Value Table [1]

763 Vehicle(s)

Click on link for price, vehicle details, photo, and quote.

▶ BUICK

Year/Make/Model		Offered By	Color *
2002 BUICK CENTURY CU		Runde Chevrolet	
1999 BUICK CENTURY CU		Runde Chevrolet	
1998 BUICK CENTURY CU		Runde Chevrolet	
2001 BUICK LESABRE CU		Runde Chevrolet	
1997 BUICK LESABRE CU		Runde Chevrolet	
1990 BUICK LESABRE		Runde Chevrolet	
1990 BUICK LESABRE CUSTOM	SPECIAL!	Runde Chevrolet	
1997 BUICK PARK AVENUE		Runde Chevrolet	
2001 BUICK REGAL LS		Runde Chevrolet	
2001 BUICK REGAL LS		Runde Chevrolet	
2000 BUICK REGAL LS		Runde Chevrolet	
1996 BUICK REGAL CUST		Runde Chevrolet	
1995 BUICK REGAL GRAN		Runde Chevrolet	
2002 BUICK RENDEZVOUS		Runde Chevrolet	

▶ CADILLAC

Year/Make/Model	Offered By	Color *
1999 CADILLAC DEVILLE	Runde Chevrolet	

Page 1 of 51 Next Last

Figure A.8: Top-Level Table with Image Colors [13]

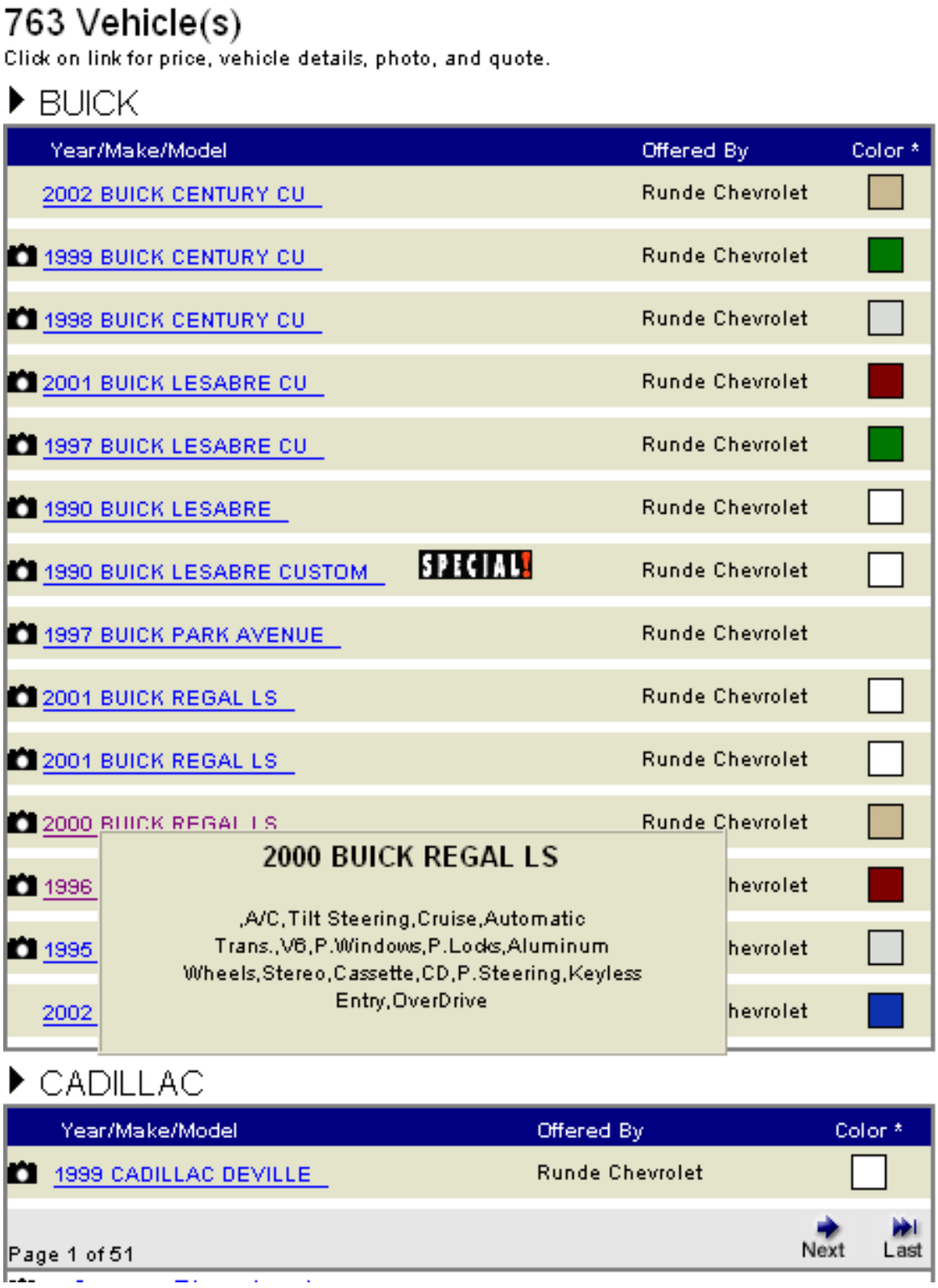


Figure A.9: View Detailed Information by Moving a Mouse [13]

Photo	Year Make/Model	Color	Mileage	Price
	1930 Duesenberg J254 Imperial Cabriolet	Black and Red		$1,250,000 OBO
	Baton Rouge LA			
	1965 Mercedes-Benz 230 SL	Metallic dark blue/Tan leather	19,123	$1,000,000
	New York City NY			
	1995 Mercedes-Benz C 1000	silver/red leather	700	$600,000
	Franklin NC			
	1932 Cadillac Rumbleseat Roadster	White		$400,000
	Solon OH			

Figure A.10: Top-Level Table with Complicated Structure [16]

Pre-Owned Vehicle Inventory

Vehicle Make and Model	Picture
1963 Corvette Split Window Coupe	
1999 Toyota 4Runner SR5 Sport Utility 4D	
1997 Honda Civic LX Sedan 4D	
1997 Lexus ES 300 Sedan 4D	
1999 Ford Crown Victoria 4D	
2000 Chevrolet 2500 Pickup Heavy Duty	
2000 Chevrolet 1500 Silverado Pickup Short Bed	
1999 GMC Suburban 1500 Sport Utility	
2000 Chevrolet S10 Pickup Extended Cab	

Figure A.11: Top-Level Table with Only 2 Columns [4]

AT&T Digital Advantage

AT&T Regional Advantage Rate
AT&T National Network Rate
AT&T One Rate
AT&T Shared Advantage Rate

Monthly Service Charge	$29.99	$39.99	$49.99	$69.99	$99.99	$149.99	$199.99
Minutes Included	200	350	500	800	1200	2000	3000
Bonus Anytime Minutes Included	100 for life	200 for life	200 for life	200 minutes per month for life			
Additional Airtime per Minute	$0.40	$0.40	$0.35	$0.35	$0.30	$0.30	$0.25
Domestic Wireless Long Distance	Nationwide Long Distance from the Home Calling Area included for life						
National Roaming	$0.69 per minute, long distance charges may apply						
Free Nights and Weekends	UNLIMITED Night & Weekend Minutes for life!						
Mobile-to-Mobile Minutes	NA	AT&T Wireless Mobile-to-Mobile Minutes					
Included Features	Caller ID, Voicemail with Message Waiting Indicator, Call Waiting, Call Forwarding, 3-Way Conference Calling, Text Messaging and Detailed Billing						
Activation Fee	$36 for 1 Year contract, $10 for 2 Year contract						

Figure A.12: Top-Level Table with Complicated Structure [10]

Monthly Service Charge	$29.99	$39.99	$49.99	$69.99	$99.99	$149.99	
Digital Multi-Network Phone Airtime Minutes Included	Up to 200	Up to 350	Up to 500	Up to 800	Up to 1200	Up To 2000	
Additional Airtime per minute	$0.40	$0.40	$0.35	$0.35	$0.30	$0.30	
Domestic Wireless Long Distance	$0.20 per Minute**						
Roaming	$0.69/min						
Night & Weekend Packages	500/mo for just $4.99, or 1000/mo for just $9.99						
AT&T Toll Free Nationwide	Not Available	Enjoy no domestic long distance on calls made from your home calling area to anywhere in the 50 United States for just $4.99/mo					
Home Service Area							

Figure A.13: Top-Level Table with Complicated Structure and Image [47]

Figure A.14: Complicated Attribute-Value-Pair Table in a Linked Page [8]

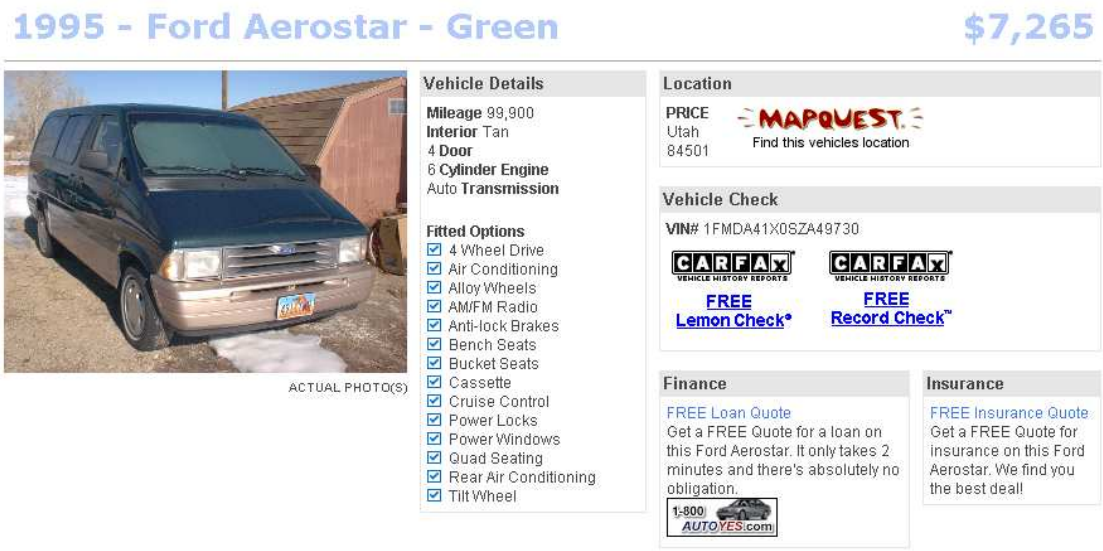


Figure A.15: Non-fixed Positions of Attribute and Value in Attribute-Value-Pair Table in Linked Pages [44]

Figure A.16: List with Checked Marks [36]