Ontology-Based Extraction and Structuring of Information from Data-Rich Unstructured Documents

David W. Embley*, Douglas M. Campbell, Randy D. Smith
Department of Computer Science

Stephen W. Liddle†
School of Accountancy and Information Systems

Brigham Young University, Provo, Utah 84602, U.S.A.
{embley,campbell,smithr}@cs.byu.edu; liddle@byu.edu

Abstract

We present a new approach to extracting information from unstructured documents based on an application ontology that describes a domain of interest. Starting with such an ontology, we formulate rules to extract constants and context keywords from unstructured documents. For each unstructured document of interest, we extract its constants and keywords and apply a recognizer to organize extracted constants as attribute values of tuples in a generated database schema. To make our approach general, we fix all the processes and change only the ontological description for a different application domain. In experiments we conducted on two different types of unstructured documents taken from the Web, our approach attained recall ratios in the 80% and 90% range and precision ratios near 98%.

Keywords: unstructured data, semistructured data, information extraction, information structuring, ontology.

1 Introduction

A relation in a structured database consists of a set of n-tuples. Each n-tuple associates n attribute-value pairs in a relationship. These relationships constitute the information asserted by the relation. A well-chosen n-place predicate for the relation can make this information human understandable. An unstructured document lacks these structuring characteristics. There are no relations with associated predicates, no attribute-value pairs, and no n-tuples. Consequently, there is no information asserted by any relation about the contents of an unstructured document.

For some unstructured documents, it is possible and useful to impose structure by establishing relations over the information contents of the document. In such cases, it is beneficial to establish these relations automatically. This paper presents an automated approach to extracting information from unstructured documents and reformulating the information as relations in a database.

We characterize our approach as being ontology based. *Ontology* is a branch of philosophy that attempts to model things as they exist in the world [10]; it is particularly appropriate for modeling objects including their relationships and properties [33]. We use an augmented semantic data model to provide an ontology that describes the view we wish to have over a domain of interest. The semantic data model lets us create an ontological model instance as sets of objects, sets of relationships among these objects, and constraints over these objects. Further, as augmented, it lets us define data representation and expected contextual keywords for each object set within the ontology. Based on an application ontology with these characteristics, we apply a parser, a constant/keyword recognizer, and a structured-text generator to filter an unstructured document with respect to the ontology and populate a generated database schema with attribute-value pairs associated as relations. We thus extract information of interest from an unstructured document and reformulate it as a structured document.

We do not expect this approach to work well for all unstructured documents. We do, however, expect the approach to work well for unstructured documents that are data rich and narrow in ontological breadth. A document is *data rich* if it has a number of identifiable constants such as dates, names, account numbers, ID numbers, part numbers, times, currency values, and so forth. A document is *narrow in ontological breadth* if we can describe its application domain with a relatively small ontological model.

As case studies to test these ideas for this paper, we consider newspaper advertisements for automobiles and newspaper job listings for computer-related jobs. Both automobile ads and job listings are data rich and narrow in ontological breadth. Automobile ads typically include constants for and information about year, make, model, asking price, mileage, features, and contact phone numbers. Computer job listings include degree required, needed skills, and contact information. Other application areas whose documents have similar characteristics include travel, stocks, financial transactions, scheduling for meetings, sports information, genealogy, medical research, product information, and many others.

We present the details of our approach as follows. We first provide a context for our research in Section 2. We describe our approach in Section 3. We give the basic framework in Section 3.1, including each of the component parts

*Research funded in part by Novell, Inc.
†Research funded in part by Faneuil Research

and how they work together to process unstructured documents for any given application ontology. In Section 3.2, we show how we applied this general approach to two specific applications—automobile advertisements from the *Salt Lake Tribune* and computer job listings from the *Los Angeles Times*. Section 4 reports the results of experiments we conducted with these two applications. In Section 5 we describe ideas for future work, and we state our conclusions in Section 6.

2 Related Work

Our research reported here relates to recent efforts in several areas including Web data modeling, wrapper generation, natural-language processing, semistructured data, and Web queries.

Others have used semantic data models to describe Web documents and populate databases [8, 14]. Our work differs from these efforts because they do not attempt to populate their model instances automatically, populating them instead by hand, with the aid of tools, or by semiautomatic methods.

The most common approach to information extraction from the Web is through wrappers, which parse source documents to provide a layer to map source data into a structured or semistructured form. If the mapped form is fully structured, standard query languages such as SQL can be used to query the extracted information. Otherwise, special "semistructured" query languages may be used [1, 3, 9]. Wrappers can be written by hand [7, 11, 20, 21], or semiautomatically [4, 6, 15, 23, 32]. Approaches to semiautomatic wrapper generation include the use of hand-coded specialized grammars [2], wrapper generators based on HTML and other formatting information [5, 21], page grammars [8], and concept definition frames [31], all of which are similar in nature. A disadvantage of hand generation and semiautomatic generation is the work required to create the initial wrapper and the rework required to update the wrapper when the source document changes. In our approach, wrapper generation is fully automatic. Instead of generating wrappers with respect to document pages, we generate wrappers based on recognizable keywords and constants described in domain ontologies. This significant departure from past approaches is a key difference in what we present here. Our approach is also more resilient to unstructured document evolution than is a rigid, grammar-oriented wrapper approach.

Research in natural-language processing (NLP) and particularly research in the information-extraction subfield as described in [12] attempts to build systems that find and link relevant information in natural-language documents while ignoring extraneous, irrelevant information. NLP processes apply such techniques as filtering to determine the relevance of a document, part-of-speech tagging, semantic tagging, building relationships among phrasal and sentential elements, and producing a coherent framework for extracted information fragments. In the NLP approach, natural language is the dominant guide; in our approach an ontology expressed as a conceptual model is the dominant guide. One interesting point of similarity is that NLP approaches to information extraction typically use special-purpose rules to recognize semantic units such as names, currencies, and equipment. These techniques use context and string patterns, similar to ours, and sometimes use large lexicons of names and places. [12] observes that although these phrasal units usually cause no problems for human readers, they can and do cause ambiguity problems that are more difficult to resolve in automatic processing.

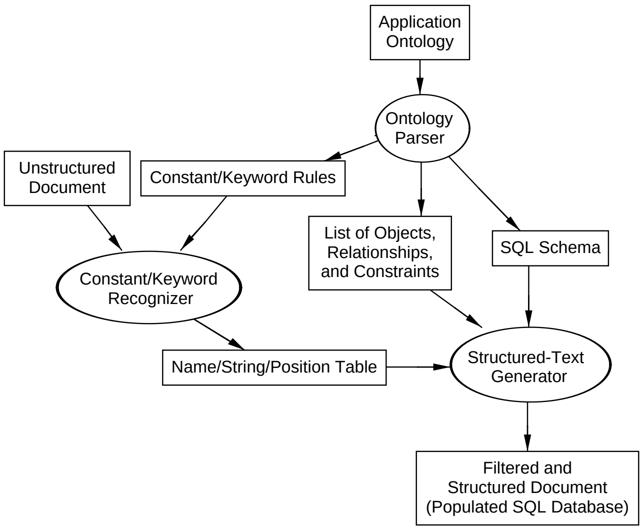

Figure 1: Framework for ontology-based information extraction and structuring.

As for similarities with semistructured-data research, one line of work consists of dynamically creating model instances over semistructured data. [19] introduces DataGuides as concise and accurate structural summaries of semistructured data; [28] and [34] attempt to do "schema discovery" to identify typical patterns found in semistructured data; and [29] considers the idea of dynamic schema construction as a way to transform semistructured data into structured data. We note that [29]'s concluding sentence states, "it is ... difficult to extract the attribute-value pairs needed to construct the schema." This is a drawback of their approaches, but we overcome this problem in our approach.

Our experimental data comes from the Web, and we believe that the techniques we propose here are useful for extracting and structuring Web information. Our approach, however, does not constitute a Web query language like [22], [24], and [27]. Instead, once we populate our model instance and produce a database, we can query the information using standard query languages such as SQL.

3 Extraction and Structuring Framework

Figure 1 shows the framework we use to extract and structure the data in an unstructured document. Boxes represent files and ovals represent processes. As Figure 1 shows, the input to our approach is an application ontology and an unstructured document, and the output is a filtered and structured document whose data is in a database. Since all the processes and intermediate file formats are fixed in advance, our framework in Figure 1 constitutes a general procedure that takes as input any declared ontology for an application domain of interest and an unstructured document within the application's domain and produces as output structured data, filtered with respect to the ontology.

The only step that requires significant human intervention is the initial creation of an application ontology. However, once such an application ontology is written, it can be applied to unstructured documents from a wide variety of sources, so long as these documents correspond to the given application domain. Also, because our extraction

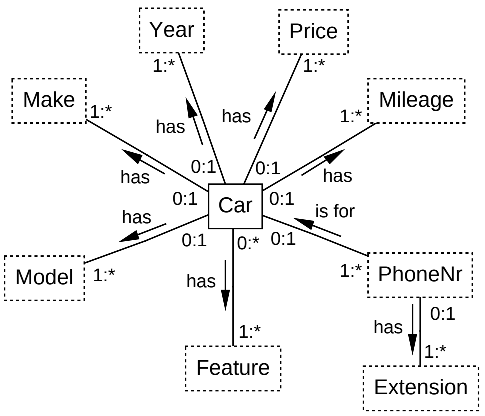

Figure 2: Graphical car-ads ontology (with regular expressions omitted).

is ontology-based, our approach is resilient to changes in source-document formats. For example, changes in HTML formatting codes do not affect our ability to extract and structure information from a given Web page.

We next give a high-level description of our approach in Section 3.1. We follow with a detailed example in Section 3.2. We use classified car ads as a running example throughout this section.

3.1 High-Level Description

As shown in Figure 1, there are three processes in our structuring framework: an ontology parser, a constant/keyword recognizer, and a structured-text generator. The input is an application ontology and a set of unstructured documents, and the output is a populated relational database. A main program invokes the parser, recognizer, and generator in the proper sequence. The ontology parser is invoked only once at the beginning of execution, while the recognizer and generator are repeatedly invoked in sequence for each unstructured document to be processed.[1]

The input to our system is an application ontology and a set of unstructured documents. The semantic data model we use for the input ontology is OSM [17], augmented to allow regular expressions as descriptors for constants and context keywords. Constants are potential values in lexical object sets, while context keywords may be associated with any object set (lexical or nonlexical). OSM has both a graphical and a textual representation, which are equivalent [26]. This equivalence allows us to present an ontology graphically, as in Figure 2 but parse it textually, as in Figure 3.

The unstructured document input is presented as a sequence of records separated by five pound signs, as shown in Figure 4. For now, we collect the pages manually, and feed them to a document cleanser and record separator that removes HTML tags and automatically divides the input into separate unstructured documents. We are working on improving the automatic record-boundary detection algorithm, and ultimately we will replace manual gathering of pages with an automatic agent to crawl the Web looking for relevant pages.

[1] In our current implementation, the main program and the recognizer are written in Perl, and we have combined them into a single script. The parser is implemented in btyacc, flex++, and C++. The generator is a C++ program.

```
Car [0:1] has Year [1:*];
Year {regexp[2]: "\d{2} : ([^\$\d]|^)\d{2}[^,\dkK]",
        "\d{2} : ([^\$\d]|^)\d{2},[^\d]",
        "\d{2} : \b'\d{2}\b" };
Car [0:1] has Make [1:*];
Make {regexp[10]: "\bchev\b", "\bchevy\b", ... };
Car [0:1] has Model [1:*];
Model {regexp[16]: "88 : \bolds\S*\s*88\b",
        "80 : \baudi\S*\s*80\b", "\bacclaim\b", ... };
Car [0:1] has Mileage [1:*];
Mileage {regexp[8]: "\b[1-9]\d{1,2}k",
        "[1-9]\d?,\d{3} : [^\$\d][1-9]\d?,\d{3}[^\d]" }
        {context: "\bmiles\b", "\bmi\.", "\bmi\b"};
Car [0:*] has Feature [1:*];
Feature {regexp[20]:
        -- Colors
                "\baqua\s+metallic\b", "\bbeige\b", ...
        -- Transmission
                "(5|6)\s*spd\b", "auto : \bauto(\.|,)",
        -- Accessories
                "\broof\s+rack\b", "\bspoiler\b", ...
        -- Engine characteristics
                "\bv-?(6|8)", "\b6\s*cyl\b", ...
        -- Body/Style
                "\b4\s*d(oo)?r\b", "\b2\s*d(oo)?r\b", ...
        -- Low mileage
                "\blow\s+miles\b", "\blow\s+mi\.", ...  };
Car [0:1] has Price [1:*];
Price {regexp[8]: "[1-9]\d?,\d{3} : \$[1-9]\d?,\d{3}",
        "[1-9]\d{2,3} : \$[1-9]\d{2,3}" };
PhoneNr [1:*] is for Car [0:1];
PhoneNr {regexp[8]:
        "[1-9]\d{2}-\d{4} : (\b|[^\d])[1-9]\d{2}-\d{4}([^\d]|$)"};
PhoneNr [0:1] has Extension [1:*];
Extension {context: "\bext\b"}
        {regexp[4]: "\d{1,4} : (x|ext\.\s+)\d{1,4}\b"};
```

Figure 3: Partial textual car-ads ontology.

```
'96 CHEV Monte Carlo Z34, loaded, bright Red
15,000 actual miles!  A great buy at $14,990,
$750 to 1000 down.  MURDOCK CHEVROLET 298-8090

#####

'94 CHEV Corsica, 88,281 miles.  Ask for #16. $4,900.
Government Surplus533-5885

#####

'89 AUDI 80, red, auto., p/w, p/l, sunroof, loaded, 128K,
new trans., new diff.  Runs perfect, must sell, $3300 obo.
gcall Nate, 554-4414
```

Figure 4: Sample unstructured input documents.

Given the inputs as described above, the next step is to invoke the ontology parser. For a given application ontology, the parser creates an SQL schema as a sequence of create-table statements. Object-set names from the ontology denote attributes in the generated SQL tables. Attributes are of type *varchar* for lexical object sets or type *integer* for nonlexical object sets. The parser includes a simple normalizer that produces its schema in an acceptable normal form assuming that the application ontology is canonical [16]. Not all information in the ontology is needed by the structured-text generator, so the parser also extracts the list of objects, relationships, and constraints to be used by the generator. This list provides a mapping between the relationships in the ontology and the table declarations in the SQL schema, and it also provides the cardinality constraints that designate which relationships are one-one, one-many, and many-many.[2] Finally, the parser also creates a file of constant/keyword rules. This file is a list of regular expressions in Perl syntax, tagged with object-set names from the OSM ontology. (We illustrate the intermediate file formats in Section 3.2.)

After invoking the parser, the main program invokes the constant/keyword recognizer and then the structured-text

[2] For our current implementation, all relationship sets are binary.

generator for each unstructured document. The recognizer applies each regular expression to the unstructured document. When the Perl program recognizes a string S according to a regular expression E with tag T, it emits T as the name, S as the string, and the beginning and ending character numbers in the document as the position. We call this list the "name/string/position table."

The structured-text generator uses the object/relationship/constraint list and the SQL schema to match attributes (object-set names in the ontology) with values (constants described in the name/string/position table). Then the generator forms tuples for relations in the generated SQL schema. The generator forms tuples according to five heuristics, applied in the following order.

1. **Keyword Proximity.** If the constraints in the ontology require at most one constant for an object set, and if there is a context keyword for the object set in the name/string/position table, we use keyword proximity to reject all but the closest constant tagged with the same object-set name as the keyword's object-set name. For constants that are equally close before and after a keyword, we favor the constant value that appears after the keyword. By "reject" we mean that we remove the entry from the name/string/position table.

2. **Subsumed and Overlapping Constants.** The constant/keyword recognizer may associate a single string S in the source with more than one object set, but a given string from the text may only generate a single constant. Thus, if constants overlap, we must reject all but one. We favor constants that are associated with keywords, so we begin by rejecting overlapping constants that are not associated with a keyword. Next we reject constants that are properly contained in another constant. If overlapping constants are still present, we favor the one that appears first. (Typically, if a constant could belong to one of several object sets in an ontology, there is a keyword associated with the constant, and the keyword-proximity heuristic will resolve the issue. Otherwise, a human would also find the constant to be ambiguous.)

3. **Functional Relationships.** If the ontology dictates that the database can accept one constant for an object set S and exactly one constant appears for S, we insert the constant in the database (regardless of keyword proximity).

4. **Nonfunctional Relationships.** If the ontology dictates that the database can accept many constants for an object set and if there are one or more constants, we insert them all.

5. **First Occurrence without Constraint Violation.** If the ontology dictates that the database can accept at most one constant for an object set S, but if there are several constants, we insert only the first one listed. Since we sort the name/string/position table according to the position of the recognized string, we assume that the first constant found that satisfies the regular expression for an object set should belong to the object set.

3.2 Application Examples

For our application case studies, we consider automobile advertisements from the *Salt Lake Tribune* and a jobs listing from the *Los Angeles Times*. We used actual unstructured documents for these applications taken from the Web. To clean these Web documents, we remove HTML markers and separate individual documents by five pound signs as described earlier. Our cleaning technique is the subject of ongoing research, and we will present its results in another paper.

Figure 4 shows three sample unstructured documents for our car-ads application. (We continue here by showing and explaining examples of the files for our car-ads application. Files for our jobs-listing application are similar in form, but different in content.)

Figure 2 gives a graphical layout of our car-ads ontology (minus the regular expressions which we have chosen to omit here). Figure 3 gives the equivalent textual representation (with some of the regular-expression lists cut short). We show only 30 regular-expression components in Figure 3; all together, our car-ads ontology had 165 regular-expression components. Observe the direct correspondence between the graphical representation and the textual representation. The relationship set connecting "Car" and "Year" in Figure 2, for example, corresponds to the declaration "Car [0:1] has Year [1:*]" in Figure 3. The bracketed numbers/number-pairs in the textual version here are participation constraints, which are displayed graphically near the connections between object sets and relationship sets. The numbers in brackets in the regular-expression declarations in Figure 3 give the maximum length of strings that can match any of the regular expressions in the declaration.

From this ontology, we generate three intermediate files (constant/keyword rules, objects/relationships/constraints, and SQL schema). Due to space limitations, we do not illustrate most of our intermediate file formats here, but they can be found on our Web site [13].

The relational schema generated from the ontology in Figure 3 includes three tables: Car(Car, Year, Make, Model, Mileage, Price, PhoneNr), PhoneNr(PhoneNr, Extension), and CarFeature(Car, Feature). Note that table attributes are object sets from the ontology. We use participation constraints to determine one-one, one-many, and many-many relationships.

The body of a regular expression rule for extracting a keyword or constant may either be simple or compound. A compound regular expression is of the form "x : y" and indicates that expression x is to be matched, but only in the context of expression y. Thus, "`\d{2} : \b'\d{2}\b`" for "Year" in Figure 3 specifies a match with two digits ("`\d{2}`") but only in the context of a word boundary followed by an apostrophe and ending with another word boundary ("`\b'\d{2}\b`"). This allows more precise matching than is possible with a single expression.

Figure 5 shows the name/string/position table generated by the constant/keyword recognizer when its document input file is the first unstructured document in Figure 4. A bar "|" separates the fields in the generated table. The first field is the name, possibly tagged as a context keyword; the second column is the recognized string (either a constant or a keyword); and the last two fields are the character positions in the unstructured document where the string begins and ends.

Observe that in Figure 5 there are several constants that are candidates for the year, mileage, and price of this vehicle. Our heuristics help us sort out the results. The first heuristic, keyword proximity, matches the mileage keyword with the "`15,000`" constant instead of "`1000`" because "`15,000`" is closer ($60-51 = 9$ characters compared to $100-64 = 36$).

Year|96|2|3
Make|CHEV|5|8
Model|Monte Carlo|10|20
~~Year|34|23|24~~
Feature|Red|42|44
Mileage|15,000|46|51
KEYWORD(Mileage)|miles|60|64
Price|14,990|84|89
~~Price|750|93|95~~
~~Mileage|1000|100|103~~
~~Make|CHEVROLET|120|128~~
PhoneNr|298-8090|130|137

Figure 5: Using heuristics to accept some constants and reject others.

Thus, we reject "1000" as a possible mileage (and thus we cross out this line in Figure 5). In this example, our second heuristic is not needed since none of the recognized constants overlaps any other. Our third heuristic, functional relationships, allows us to select the model ("Monte Carlo") and phone number ("298-8090") values. By the fourth heuristic, nonfunctional relationships, we accept "Red" as a feature (and if other features had been present, they would have been accepted as well). Finally, we use the fifth heuristic to reject "34" as a year, "750" as a price, and "CHEVROLET" as a make.

The structured-text generator creates two tuples from the constants and keywords in Figure 5: Car(1001, "96", "CHEV", "Monte Carlo", "15,000", "14,990", "298-8090") and CarFeature(1001, "Red"). These tuples are emitted as SQL insert statements.

After feeding the car ads one at a time to our extraction and structuring procedure, we obtain a fully populated database. We can then pose standard SQL queries to the automatically populated database. As an example, an SQL query to get the year, make, model, and price for 1987 or later cars that are red or white yielded the following result.

```
Year Make       Model           Price
---- ---------- --------------- --------
94   DODGE                      4,995
94   DODGE      Intrepid        10,000
91   FORD       Taurus          3,500
90   FORD       Probe
88   FORD       Escort          1000
```

In our second case study we extracted information from computer jobs listed in the *Los Angeles Times*. Figure 6 shows the ontology we used (minus the regular expressions). Our jobs-listing ontology included 120 regular-expression components. The ontology in Figure 6 declares the structure information about the degree needed, the skills needed, and how to contact someone about the job. After populating the database with respect to this filter, we were again able to perform useful queries using SQL.

4 Experimental Results

For our experiments, we took 216 car ads from a Web page provided by the *Salt Lake Tribune* (www.sltrib.com) and 100 advertisements for computer jobs from a Web page provided by the *Los Angeles Times* (www.latimes.com). In

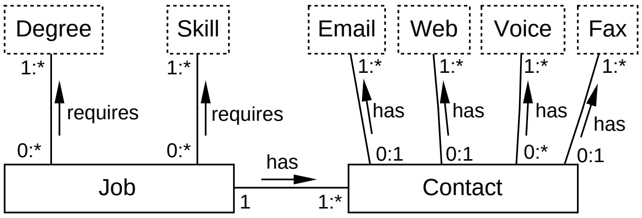

Figure 6: Graphical computer job-listing ontology.

both cases, we selected advertisements from one day to be our tuning set[3] and from another day (about a week and a half later) to be our test set. We used the tuning set to determine which object and relationship sets would be in the ontology and what regular expressions would recognize constants and keywords. We refined the ontology until it described information in the tuning set as completely as possible. In generating regular expressions, we did not limit ourselves to patterns found in the tuning set. We used our own experience to generalize some of the patterns, but we did not attempt to be comprehensive—just to be as accurate as possible on the tuning set. We developed and tuned our application ontologies using 100 cars ads and 50 job ads. We then applied these ontologies to the test sets and obtained the results in Table 1 (car advertisements) and in Table 2 (computer jobs).[4]

As reported in Tables 1 and 2, we counted the number of facts (N) in the test-set documents for each attribute. A "fact" is a particular attribute-value pair about an item of interest, such as "(Year, 94)" for a car ad, or "(Fax, 353-9278)" for a job listing. For the most part, counting these facts was straightforward. We note, however, that for car ads, we only counted one phone number even when the ad had more (our car-ads ontology only requested one); similarly, for our job ads, we only counted one email address, one Web URL, and one fax number, although our jobs ontology did allow for several voice phone numbers. Features for car ads and skills for job ads are unbounded. We limited features to actual physical features of cars; we limited skills to computer languages, tools, and systems. Thus, for example, we eliminated "Government Surplus" and "Runs perfect" as features of cars and "works well with others" and "willing to relocate" as job skills. For the number of facts declared correctly in Tables 1 and 2, we counted the number of facts in the test-set documents that our automated system correctly extracted and inserted into the database. For the number of facts declared incorrectly in Tables 1 and 2, we counted the number of facts our automated system inserted into the database, but which were not facts in the test-set documents.

In information retrieval, *recall* is the ratio of the number relevant documents retrieved to the total number of relevant documents, and *precision* is the ratio of the number of relevant documents retrieved to the total number of documents

[3] We choose the term "tuning set" instead of "training set" to avoid confusion with machine learning. In our case we are not training the system; rather a domain expert is encoding knowledge into the application ontology through a process of successive refinement.

[4] In our experiments, the ratio of tuning documents to test documents is quite high (on the order of one-to-one) because we had to manually examine the experiment results exhaustively, which is time-consuming. In practice, one would build the application ontology based on a relatively small number of documents, as we did, and then apply the ontology to a much larger set of documents (e.g., all Web-based car ads from a set of on-line newspapers known to supply such classified advertisements).

Table 1: *Salt Lake Tribune* Automobile Advertisements

	N	C	I	$\frac{C}{N}$	$\frac{C}{C+I}$
Car	116	116	0	1.00	1.00
Year	116	116	0	1.00	1.00
Make	116	113	0	0.97	1.00
Model	114	93	0	0.82	1.00
Mileage	31	28	0	0.90	1.00
Price	103	103	0	1.00	1.00
PhoneNr	116	109	0	0.94	1.00
Extension	2	1	0	0.50	1.00
Feature	289	264	1	0.91	0.996
All Attributes	1003	943	1	0.94	0.9989

Table 2: *Los Angeles Times* Computer Jobs Listing

	N	C	I	$\frac{C}{N}$	$\frac{C}{C+I}$
Job	50	50	0	1.00	1.00
Degree	6	6	0	1.00	1.00
Skill	165	122	0	0.74	1.00
Contact	50	50	0	1.00	1.00
Email	11	10	2	0.91	0.83
Web	0	0	0	-	-
Fax	33	30	0	0.91	1.00
Voice	14	11	1	0.79	0.92
All Attributes	329	279	3	0.84	0.98

retrieved [18]. To compute our recall and precision ratios, we let *facts* be *documents*. If N is the number of facts in the source, C is the number of facts declared correctly, and I is the number declared incorrectly, the recall ratio is $\frac{C}{N}$, and the precision ratio is $\frac{C}{C+I}$. For our tuning sets, we were able to achieve recall and precision ratios of 0.98 and 0.995 respectively for car ads and to 0.99 and 0.99 respectively for computer job listings. For our test sets, Tables 1 and 2 show the results for each attribute and for all attributes combined.[5]

Several comments about our results are in order. Nonlexical object sets ("Car", "Job", "Contact") are always identified correctly because they are represented by surrogate identifiers we generate. Their presence in a record is assumed by our extraction algorithm. If we discard the data for these "given" nonlexical object sets, recall drops to 86% and 78% respectively for car ads and job ads, but precision remains high at 99% and 98% respectively.

Lexical object sets are subdivided into bounded and unbounded sets. Unbounded sets, such as car features and job skills, generally dominate overall precision and recall numbers. For example, the 94% recall for all car attributes is due mostly to unbounded model and feature sets. Similarly, the unbounded skill set dominates the overall recall for job attributes. In both cases, if we had used larger tuning sets, we could have done better. For example, we missed the car make "MERC" and a number of models (e.g., "Continental", "Town Car", "98 Royale"). If we had used a comprehensive catalog of car makes and models, we would have achieved near 100% recall. Similarly, for jobs, if we had cataloged a larger set of skills, including skills we missed such as "CICS", "DB2", and "BAL", we would have achieved near 100% recall. This kind of error is due to incomplete domain analysis, which is relatively easy to correct in practice.

[5] Often, IR literature shows recall and precision information as curves, but in our case, only a single point for each attribute applies.

Another similar category of errors was due to variations in patterns we had already seen. For example, "5 speed" appeared instead of "5 spd", "p.l" instead of "p.l.", "Wind95" instead of "Win95", and "888-60TITLE" or "818.546.1619" instead of a more common phone number format.

Misidentification of attributes was infrequent because of the fine degree of precision in our regular expressions (due in part to our two-level context scheme). On the blind test sets for both applications this led surprisingly to 98% or better precision. The only incorrect identifications involved the "C" language and the abbreviation "AUTO" (representing automatic transmission).

Typographical mistakes also led to lower recall. For example, we missed the make "Chrystler", the model "Camero", the feature "casss.", and the phone number "805) 295-8323". Sometimes missing white space caused lower recall, as with the seven phone numbers missed in the car experiment. All seven were run together with the dealership address ("I-15566-2441"), causing the regular expression to miss the phone number. (We suspect that even humans outside of Utah, not realizing that Interstate 15 is the main north-south highway, might have trouble extracting the "I-15" prefix as part of an address.)

Finally, some attributes were missed due to weaknesses in our heuristics. We miscategorized a couple of fax numbers as voice numbers instead because the context keyword "phone" appeared in the ad, but there was no separate voice phone. Another fax number was miscategorized because the keyword "fax" appeared after the number instead of before.

Within our existing framework, there are several ways we can improve our results. Larger tuning sets and more complete domain analysis will (1) provide more complete descriptions of unbounded categories (job skills and car features), (2) enhance our regular expressions to match more variations of patterns we have already included, and (3) improve how we use context keywords to recognize and categorize constants. With more experience, we may also be able to lessen the impact of typographical errors by correcting for common mistakes. For example, we missed one phone number because the typist entered the letter "l" ("ell") instead of the digit "1" ("one").

5 Future Work

We have provided a framework for converting data-rich unstructured documents into structured documents. In addition, we have implemented the procedures in our framework, and we have demonstrated that our framework and implemented procedures achieve good results. However, much remains to be done. Three particular tasks lie ahead: (1) improve and fine-tune the implemented procedures, (2) add front-end page processors, and (3) diversify back-end display generators.

There are many things we can do to fine-tune our approach. For example, we can make the ontological descriptions richer. OSM already has a much richer semantics than the small subset of the language we presented here [17]. Additional modeling constructs include n-ary relationship sets, generalization/specialization hierarchies, aggregation, declared-view generation, predicate-calculus-based constraints, and computation-based constraints. Corresponding model-equivalent textual components have been defined as well and parsers have been built [25]. Also, procedures exist for synthesizing schemas—both flat schemas as well as nested schemas [16]. We have begun to explore the impact of richer ontological descriptions by applying our framework to extract genealogical information from obituaries. Preliminary results indicate that we can obtain recall and precision ratios similar to our car ads and job listings but in the much richer application domain of genealogy.

We can also improve schema generation and data population. From the regular expressions, we have the information we need to generate better types—e.g., integer, float, money, date, and time. We also need to convert the values obtained to these types. As currently implemented, the strings we obtain make comparisons difficult. "55,000" is the same as "55K", but we cannot directly compare them for equality. Furthermore, ordering of these and other constants with multiple formats makes direct less-than and greater-than comparisons meaningless. It would also be useful to allow replacement of constants with more descriptive phrases; for example, we might convert both "p.l." and "pl" to "power locks".

We can improve and fine-tune our heuristics. Finding and testing better heuristics is definitely possible and deserves attention. Representing them declaratively and processing them with different strategies is also possible and should be investigated.

We can make use of techniques from other research areas. Examples include natural-language processing [12] (to improve our semantic understanding—currently, our approach is purely syntactic, which has the advantage of being much faster to process), constraint programming (e.g., for determining whether candidate constant values fall within equational constraints such as "birth date must precede death date"), and theory of evidence [30] (e.g., for weighting the results of different, competing heuristics to decide which path to choose).

We can develop tools and libraries for ontology specification. The only step in our process that is human-resource intensive is the specification of an application ontology. Indeed, this is a complex step that, like programming, requires an expert. Graphical tools for editing the ontology and seeing the effects of changes within a sample set of documents would be very helpful in reducing the burden of regular-expression generation. We have begun to develop such graphical tools in Java. Also, we can provide pre-built libraries of common ontology types (e.g., currency, date, time, phone, e-mail, URLs) or domains.

Front-end page processors are needed to prepare documents. For our case studies we searched the Web, found documents of interest, saved the HTML pages to a file, identified record boundaries within the HTML text, and processed the files to insert record separators and remove HTML markers. How much of this can be automated? As it turned out, we were able to insert record separators and remove HTML markers with just a few minutes of effort. We are currently working on the nontrivial task of automatically separating records. Interestingly, our ontology can also help us automatically recognize record boundaries. For example, if we discover key constants repeating, we can infer that we have crossed a record boundary. We are working on refining our approach to boundary recognition.

Diversification of back-end display generators can broaden the range of usability. In our implementation, we generate a populated relational database and use SQL select statements for retrieval. Many user-friendly interfaces have already been built over standard database interfaces, so this is certainly feasible. Besides standard databases with user-friendly interfaces, we could generate XML pages to organize and store extracted information. Indeed, we need not actually extract the information, but can instead establish links to the information and use it in place. This, of course, is particularly useful when the information to be extracted includes multimedia objects such as images, video, and sound.

6 Conclusions

Contributions of this paper include the following:

1. We proposed a framework for an ontology-based system that extracts and structures information found in data-rich unstructured documents. Except for ontology creation, the processes in our framework are automatic and do not require human intervention.
2. We built a prototype system based on this framework.
3. We applied our prototype system to two application areas—car advertisements and a computer jobs listing. As raw data for these applications, we used documents placed on the Web by the *Salt Lake Tribune* and the *Los Angeles Times*.
4. In experiments we conducted, we obtained near 99% recall and precision on tuning data and roughly 90% recall and 98% precision on test data.

For applications that are data rich and narrow in ontological breadth, the approach presented here shows great promise. We observed that most of the errors in recall and precision were due to incomplete lexicons and incomplete ontologies. *Without changing the framework*, better lexicons and richer ontologies will overcome both of these shortcomings. Improvements in heuristics, front-end processing, and back-end processing are also possible.

References

[1] S. Abiteboul, S. Cluet, V. Christophides, T. Milo, G. Moerkotte, and Jérôme Siméon. Querying documents in object databases. *International Journal on Digital Libraries*, 1(1):5–19, April 1997.

[2] S. Abiteboul, S. Cluet, V. Christophides, T. Milo, G. Moerkotte, and Jérôme Siméon. Querying documents in object databases. *International Journal on Digital Libraries*, 1(1):5–19, April 1997.

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1), April 1997.

[4] B. Adelberg. Nodose - a tool for semi-automatically extracting structured and semistructured data from text documents. To appear in Proceedings of SIGMOD'98, 1998.

[5] N. Ashish and C. Knoblock. Wrapper generation for semi-structured internet sources. *SIGMOD Record*, 26(4):8–15, December 1997.

[6] N. Ashish and C. Knoblock. Wrapper generation for semi-structured internet sources. *SIGMOD Record*, 26(4):8–15, December 1997.

[7] P. Atzeni and G. Mecca. Cut and paste. In *Proceedings of the PODS'97*, 1997.

[8] P. Atzeni, G. Mecca, and P. Merialdo. To weave the web. In *Proceedings of the Twenty-third International Conference on Very Large Data Bases*, pages 206–215, Athens, Greece, August 1997.

[9] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of SIGMOD'96*, June 1996.

[10] M.A. Bunge. *Treatise on Basic Philosophy: Vol. 3: Ontology I: The Furniture of the World*. Reidel, Boston, 1977.

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

[12] J. Cowie and W. Lehnert. Information extraction. *Communications of the ACM*, 39(1):80–91, January 1996.

[13] Data Extraction Group Home Page. URL: http://osm7.cs.byu.edu/deg.

[14] L.M.L. Delcambre, D. Maier, R. Reddy, and L. Anderson. Structured maps: Modeling explicit semantics over a universe of information. *International Journal on Digital Libraries*, 1(1):20–35, April 1997.

[15] R. Doorenbos, O. Etzioni, and D. Weld. A scalable comparison-shopping agent for the world-wide web. In *Proceedings of the first international conference on autonomous agents 97*, 1997.

[16] D.W. Embley. *Object Database Development: Concepts and Principles*. Addison-Wesley, Reading, Massachusetts, 1998.

[17] D.W. Embley, B.D. Kurtz, and S.N. Woodfield. *Object-oriented Systems Analysis: A Model-Driven Approach*. Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[18] W.B. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures & Algorithms*. Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[19] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *Proceedings of the Twenty-third International Conference on Very Large Data Bases*, pages 436–445, Athens, Greece, August 1997.

[20] A. Gupta, V. Harinarayan, and A. Rajaraman. Virtual database technology. *SIGMOD Record*, 26(4):57–61, December 1997.

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

[22] D. Konopnicki and O. Shmueli. A query system for the world wide web. In *Proceedings of the Twenty-first International Conference on Very Large Data Bases*, pages 54–65, Zürich, Switzerland, 1995.

[23] N. Kushmerick, D.S. Weld, and Doorenbos. Wrapper induction for information extraction. In *Proceedings of the 1997 International Joint Conference on Artificial Intelligence*, 1997.

[24] L.V.S. Lakshmanan, F. Sadri, and I.N. Subramanian. A declarative language for querying and restructuring the web. In *Proceedings of the 6th International Workshop on Research issues in Data Engineering, RIDE'96*, New Orleans, Louisiana, 1996.

[25] S.W. Liddle. *Object-Oriented Systems Implementation: A Model-Equivalent Approach*. PhD thesis, Department of Computer Science, Brigham Young University, Provo, Utah, June 1995.

[26] S.W. Liddle, D.W. Embley, and S.N. Woodfield. Unifying Modeling and Programming Through an Active, Object-Oriented, Model-Equivalent Programming Language. In *Proceedings of the Fourteenth International Conference on Object-Oriented and Entity Relationship Modeling (OOER'95), Lecture Notes in Computer Science, 1021*, pages 55–64, Gold Coast, Queensland, Australia, December 1995. Springer Verlag.

[27] A.O. Mendelzon, G.A. Mihaila, and T. Milo. Querying the world wide web. *International Journal on Digital Libraries*, 1(1):54–67, April 1997.

[28] S. Nestorov, J. Ullman, J. Wiener, and S. Chawathe. Representative objects: Concise representations of semistructured, hierarchiacl data. In *Proceedings of the Thirteenth International Conference on Data Engineering*, Birmingham, UK, April 1997.

[29] D.-Y. Seo, D.-H. Lee, K.-S. Moon, J. Chang, J.-Y. Lee, and C.-Y. Han. Schemaless Representation of Semistructured Data and Schema Construction. In *Proceedings of the 8th International Conference on Databases and Expert Systems Applications (DEXA'97)*, pages 387–396, Toulouse, France, September 1997. Springer Verlag.

[30] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[31] D. Smith and M. Lopez. Information extraction for semi-structured documents. In *Proceedings of the Workshop on Management of Semistructured Data*, Tucson, Arizona, May 1997.

[32] S. Soderland. Learning to extract text-based information from the world wide web. In *Proceedings of the Third International Conference on Knowledge discovery and Data Mining*, pages 251–254, Newport Beach, California, August 1997.

[33] Y. Wand. A proposal for a formal model of objects. In W. Kim and F.H. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, pages 537–559. ACM Press, New York, 1989.

[34] K. Wang and H. Liu. Schema discovery for semistructured data. In *Proceedings of the Third International Conference on Knowledge discovery and Data Mining*, pages 271–274, Newport Beach, California, August 1997.