Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages

D.W. Embley[*], D.M. Campbell, Y.S. Jiang, S.W. Liddle[†], D.W. Lonsdale, Y.-K. Ng, R.D. Smith

Data Extraction Group

Brigham Young University, Provo, Utah 84602, U.S.A.

{embley,campbell,jiang,ng,smithr}@cs.byu.edu; {liddle,lonz}@byu.edu

Abstract

Electronically available data on the Web is exploding at an ever increasing pace. Much of this data is unstructured, which makes searching hard and traditional database querying impossible. Many Web documents, however, contain an abundance of recognizable constants that together describe the essence of a document's content. For these kinds of data-rich, multiple-record documents (e.g. advertisements, movie reviews, weather reports, travel information, sports summaries, financial statements, obituaries, and many others) we can apply a conceptual-modeling approach to extract and structure data automatically. The approach is based on an ontology—a conceptual model instance—that describes the data of interest, including relationships, lexical appearance, and context keywords. By parsing the ontology, we can automatically produce a database scheme and recognizers for constants and keywords, and then invoke routines to recognize and extract data from unstructured documents and structure it according to the generated database scheme. Experiments show that it is possible to achieve good recall and precision ratios for documents that are rich in recognizable constants and narrow in ontological breadth. Our approach is less labor-intensive than other approaches that manually or semiautomatically generate wrappers, and it is generally insensitive to changes in Web-page format.

Keywords: data extraction, data structuring, unstructured data, data-rich document, World-Wide Web, ontology, ontological conceptual modeling, obituaries.

1 Introduction

The amount of data available on the Web has been growing explosively during the past few years. Users commonly retrieve this data by browsing and keyword searching, which are

[*]Research funded in part by Novell, Inc.

[†]Research funded in part by Faneuil Research Group

intuitive, but present severe limitations [4]. Browsing is not suitable for locating particular items of data because following links is tedious, and it is easy to get lost. Furthermore, browsing is not cost-effective as users have to read the documents to find desired data. Keyword searching is sometimes more efficient than browsing but often returns vast amounts of data, far beyond what the user can handle.

To retrieve data more efficiently from the Web, some researchers have resorted to ideas taken from database techniques. Databases, however, require structured data. Yet most Web data is unstructured and cannot be queried using traditional query languages. To attack this problem, various approaches for querying the Web have been suggested. These techniques basically fall into one of two categories: querying the Web with Web query languages (e.g. [5]) and generating wrappers for Web pages (e.g. [6]).

In this paper, we discuss a novel approach to extracting and structuring data from documents posted on the Web. Our data extraction method is based on conceptual modeling, and, as such, this approach also represents a new direction for research in conceptual modeling.

Our approach specifically focuses on unstructured documents that are data rich, narrow in ontological breadth, and contain multiple records of information for the ontology. 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 ontology. A document *contains multiple records for an ontology* if there is a sequence of chunks of information about the main entity in an ontology. None of these definitions is exact, but they express the idea that the kinds of Web documents we are considering have many constant values, are narrow in the domain they cover, and contain descriptions for several object instances that satisfy the ontology.

The unstructured documents we have chosen for illustration in this paper are obituaries. Figure 1 shows an example. An obituary is data rich, typically including several constants such as name, age, death date, and birth date of the decedent; a funeral date, time, and address; viewing and interment dates, times, and addresses; names of related people and family relationships. The information in an obituary is also narrow in ontological breadth, having data within a narrow domain of genealogical knowledge that can be described by a small ontological model instance. Obituaries generally appear in groups, being listed one

Brian Fielding Frost

Our beloved Brian Fielding Frost, age 41, passed away Saturday morning, March 7, 1998, due to injuries sustained in an automobile accident. He was born August 4, 1956 in Salt Lake City, to Donald Fielding and Helen Glade Frost. He married Susan Fox on June 1, 1981.

He is survived by Susan; sons Jordan (9), Travis (8), Bryce (6); parents, three brothers, Donald Glade (Lynne), Kenneth Wesley (Ellen), Alex Reed, and two sisters, Anne (Dale) Elkins and Sally (Kent) Britton. A son, Michael Brian Frost, preceded him in death.

Funeral services will be held at 12 noon Friday, March 13, 1998 in the Howard Stake Center, 350 South 1600 East. Friends may call 5-7 p.m. Thursday at Wasatch Lawn Mortuary, 3401 S. Highland Drive, and at the Stake Center from 10:45-11:45 a.m. Friday. Interment at Wasatch Lawn Memorial Park.

Figure 1: A sample obituary.

after another on a single page.

Specifically, our approach consists of the following five steps. (1) We develop an ontological model instance over an area of interest. (2) We parse this ontology to generate a database scheme and to generate rules for matching constants and keywords. (3) To obtain data from the Web, we invoke a record extractor that divides an unstructured Web document into individual record-size chunks, cleans them by removing markup-language tags, and presents them as individual unstructured record documents for further processing. (4) We invoke recognizers that use the matching rules generated by the parser to extract from the cleaned individual unstructured documents the objects expected to populate the model instance. (5) Finally, we populate the generated database scheme by using heuristics to determine which constants populate which records in the database scheme. These heuristics correlate extracted keywords with extracted constants and use relationship sets and cardinality constraints in the ontology to determine how to construct records and insert them into the database scheme. Once the data is extracted, we can query the structure using a standard database query language.

To make our approach general, we fix the ontology parser, Web record extractor, keyword and constant recognizer, and database record generator; we change only the ontology as we move from one application domain to another. A significant contribution of our approach is that we only perform the manual step, ontology development, once for a particular domain. This ontology covers all Web pages for that domain, and is insensitive to changes in Web-page format. Other approaches that rely on HTML structure or the order of data within an unstructured record must specify multiple wrappers (one for each structure pattern), and when a Web page undergoes a format change (a common occurrence), such wrappers must be rewritten to accommodate the new format [20]. Our system generally does not rely on the order of data or the specific nature of a particular Web-page layout.

In an earlier paper [16], we presented some of these ideas for extracting and structuring data from unstructured documents. We also presented results of experiments we conducted on two different types of unstructured documents taken from the Web, namely, car ads and job ads. In those experiments, our approach attained recall ratios in the range of 90% and precision ratios near 98%. These results were encouraging; however, the ontology we used was very narrow, and was limited to a relatively flat database structure.

In this paper we enrich the ontology—the conceptual model—and we choose an application that demands more attention to this richer ontology. For example, our earlier model supported only binary relationship sets, but our current approach supports n-ary relationship sets. Furthermore, we enhance the ontology in two significant ways. (1) We adopt "data frames" as a way to encapsulate the concept of a data item with all of its essential properties. (2) We include lexicons to enrich our ability to recognize constants that are difficult to describe as simple patterns, such as names of people. Together, data frames and lexicons enrich the expressiveness of an ontological model instance. This paper also extends our earlier work by adding an automated tool for detecting and extracting unstructured records from HTML Web documents. We are thus able to fully automate the extraction process once we have identified a Web document from which we wish to extract data. Further enhancements are still needed to locate documents of interest with respect to the ontology and to handle sets of related documents that together provide the data for a given ontology. Nevertheless, the extensions we do add in this paper significantly enhance the approach presented earlier [16]. Some of these enhancements were discussed in [17], but the present paper gives a more comprehensive elaboration.

We present the details of our approach as follows. We first provide a context for our research in Section 2 by showing how it aligns with the work of other researchers. In Section 3, we discuss each of the component parts of our approach and show, for any given application ontology, how they work together to process data-rich, unstructured documents such as obituaries. In Section 4, we report results as recall and precision ratios for retrieved data for the experiment we conducted on obituaries; we also present experimental data supporting our claims of the generality of our approach. In Section 5, we state our conclusions.

2 Related Work

With the explosion of textual information available in electronic form, a large research effort has sprung up in the database community around finding ways to make Web querying more powerful than keyword search and browsing. Recent efforts have taken several directions including virtual database technology, Web data modeling, wrapper generation, natural-language processing, semistructured data, and Web queries.

Junglee coined the phrase "virtual database technology" [20]. Just as "virtual memory" makes the disk an extension of main memory, "virtual database technology" makes external data an extension of the database. Junglee developed a sophisticated proprietary system to extract information from Web pages; one part of their system uses hints provided by HTML tags. In contrast, we extract information from Web pages using hints provided by an ontology. We only use HTML tags in our record-boundary detection phase, not in the actual data extraction.

"Structured maps" are another technique to model Web-based information sources [12]. Similar to our ontological model instance, a semantic model provides a schema over a domain of interest. The schema is populated with information elements from the Web. [7] introduces a data model to describe the schema for a user view over information on the Web along with a set of languages for synthesizing the schema for the particular application and to manage and restructure data with respect to the schema. We have not yet gone beyond populating a database with extracted values and tuples. Our work differs from those that populate databases by hand, because in our approach the same ontology can be used to extract data from multiple Web pages requiring minimal changes (if any at all), even for Web pages authored in different styles.

The most common way to extract information from the Web is by generating a *wrapper*, which parses unstructured data and then maps it into a structured or semistructured form.

If the mapped form is structured, then standard query languages such as SQL are used to query the extracted information. If the mapped form is semistructured, then special semistructured query languages are used [1, 2, 9]. Wrappers can be written by hand, as they were in the TSIMMIS project [10] (whose main thrust was information integration). Wrappers can also be written semiautomatically. Approaches to semiautomatic wrapper generation include generators using (i) hand-coded specialized grammars [1], (ii) formatting information [6, 21], (iii) page grammars [7], and (iv) concept definition frames [29]; these approaches are all similar. Wrappers have been written either fully manually [8, 20, 21], or with some degree of automation [3, 6, 13, 23, 30]. Hand generation and semiautomatic generation have two disadvantages: (i) the amount of work to create the initial wrapper, and (ii) the amount of work required to update the wrapper when a source document changes. Although we also produce wrappers in our work, our approach differs fundamentally; our work uses conceptual modeling to control the information-extraction process. In our approach, wrapper generation is fully automatic once the conceptual-model instance representing the application ontology has been written.

Natural-language processing (NLP) systems (particularly in the information-extraction subfield as described in [11]) find relevant information in natural-language documents while ignoring irrelevant information. To find relevant information, NLP systems apply such techniques as filtering, part-of-speech tagging, parsing, lexical semantic tagging, building relationships among phrasal and sentential elements, and producing a grammatically coherent framework. In the NLP approach, natural language is the dominant guide; in our approach, an ontology expressed as a conceptual model is the dominant guide. Unlike many NLP systems, our work does not attempt to extract "deep-level understanding" and our work does not depend upon complete sentences. Our approach is more appropriate for Web pages that publish information (like classified ads) which rarely contains complete sentences.

"Case frames" [30] are a middle ground between classical NLP approaches and wrapper-based approaches. Case frames parse Web pages into coherent segments based on page layout cues. These "sentence-length" segments of text are then fed into a NLP system that relies on a semantic lexicon for the domain of interest instead of using part-of-speech tagging. The output is a case frame, from which a populated relational database could be generated. Although case frames have some similarities to our ontology-based approach, the approaches are quite different: we are guided by a domain ontology; they are guided by NLP techniques.

"Concept definition frames" [29] are also used to extract information from text-based data sources. Concept definition frames are similar to the object definitions ("data frames") in our conceptual model. However, our conceptual-model approach is richer; for example, we include cardinality constraints, which are used in our heuristics for composing extracted attribute values into object structures.

Our experimental data comes from the Web. We believe that our techniques are useful for extracting and structuring Web information. Our approach, however, does not constitute a query language that is "Web aware" [5, 22, 24, 27, 28], Instead, once we populate our model instance and produce a database, we can perform queries using standard query languages such as SQL.

3 Web Data Extraction and Structuring

Figure 2 shows the overall process we use for extracting and structuring Web data. As depicted in the figure, the input (upper left) is a Web page, and the output (lower right) is a populated database. Figure 2 shows the application ontology as an independent input describing the application of interest. When applications change (for example from car ads, to job ads, to obituaries), it suffices to change the ontology, in order to apply the process to Web pages associated with the new application. Everything else remains the same: the routines that extract records, the routines that parse the ontology, the routines that recognize constants and keywords, the routines that generate the populated database instance—*none of these routines change.* In this way, our process is generally applicable to any domain.

We proceed by describing in succeeding subsections each major component of Figure 2.

3.1 Ontological Specification

As Figure 2 shows, the application ontology consists of an object-relationship model instance, data frames, and lexicons. An ontology parser takes this information as input and produces constant/keyword matching rules and a database description as output. First we describe our object-relationship model; then we describe data frames and lexicons.

Figure 3 gives the object-relationship model instance for our obituary application in

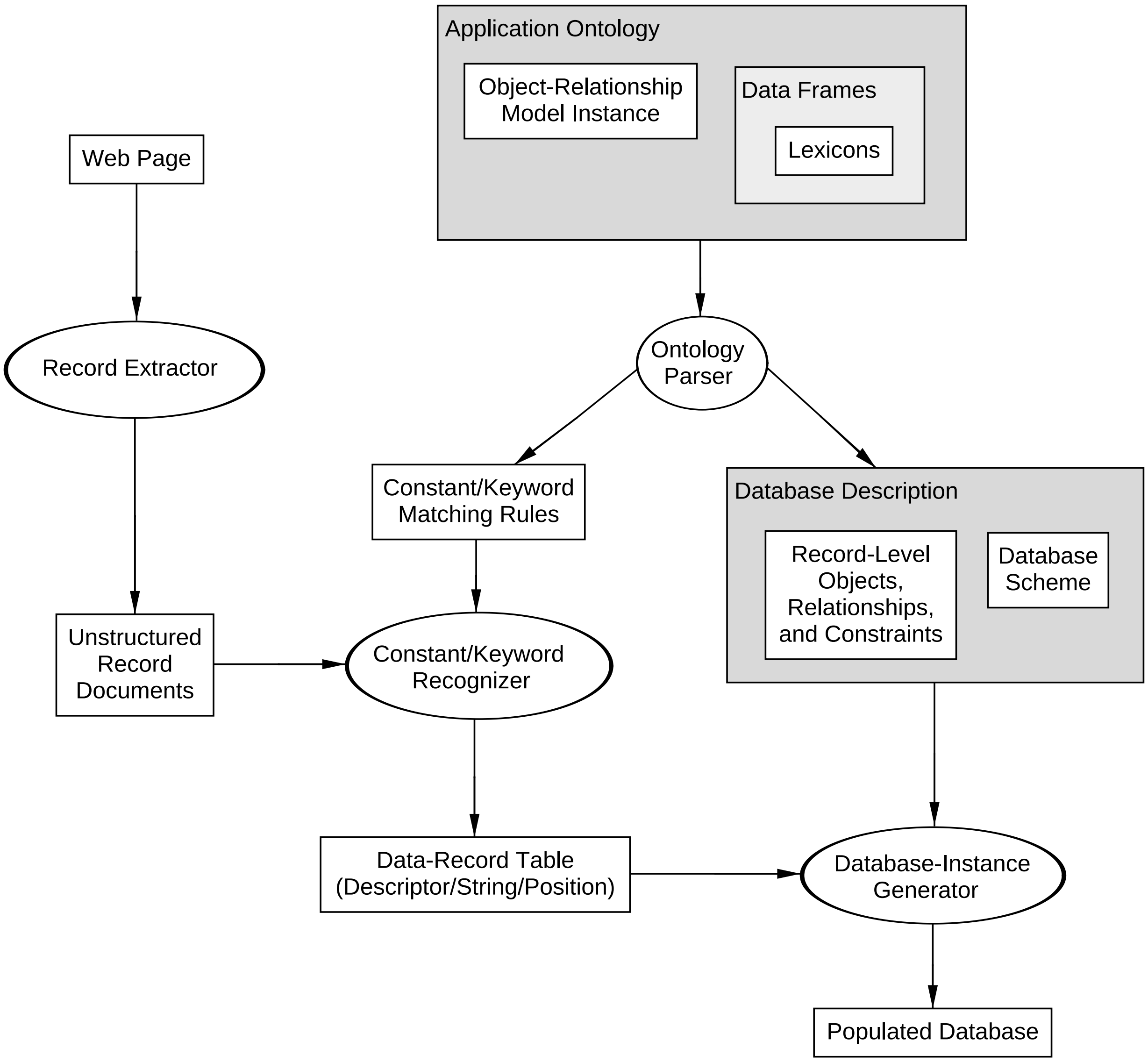


Figure 2: Data extraction and structuring process.

graphical form; Figure 4 displays excerpts of this model instance in its equivalent textual form. We use the Object-oriented Systems Model (OSM) [15] for our object-relationship model. We briefly discuss four aspects of OSM: (1) object sets, (2) relationship sets, (3) participation constraints, and (4) generalization/specialization.

1. **Object Sets.** Rectangles represent sets of objects; dotted rectangles represent lexical object sets (such as *Age* and *Birth Date* whose objects have identifiers that represent themselves); solid rectangles represent nonlexical object sets (such as *Deceased Person* and *Viewing* whose objects are object identifiers that represent nonlexical real-world entities).

2. **Relationship Sets.** Lines connecting rectangles represent sets of relationships; binary relationship sets have a verb phrase and reading-direction arrow (e.g. *Funeral is on Funeral Date* names the relationship set between *Funeral* and *Funeral Date*); n-ary relationships have a diamond and a full descriptive name that includes the names of its connected object sets.

3. **Participation Constraints.** Participation constraints (located near connection points between object sets and relationship sets) designate the minimum and maximum number of times an object in the set participates in the relationship. For example, the *1..** near *Age* indicates that an age must associate with at least one decedent, and perhaps many.

4. **Generalization/Specialization.** A colon (:) after an object-set name (e.g. *Birth Date: Date*) denotes that the object set is a specialization (e.g. the set of objects in *Birth Date* is a subset of the objects in the implied *Date* object set).

For our ontologies, an object-relationship model instance gives both a global view (e.g. across all obituaries) and a local view (e.g. for a single obituary). We express the global view as an object-relationship model instance, as we described above. We then specialize the global view to a particular obituary by imposing additional constraints. We denote these specializing constraints by a "becomes" arrow (–>). For example, in Figure 3, by adding "–> •", the *Deceased Person* object set: (a) becomes a single object, meaning that in an obituary there is exactly one decedent of interest, and (b) the *1..** participation constraint on both *Deceased Name* and *Relative Name* becomes *1.* These specializing constraints

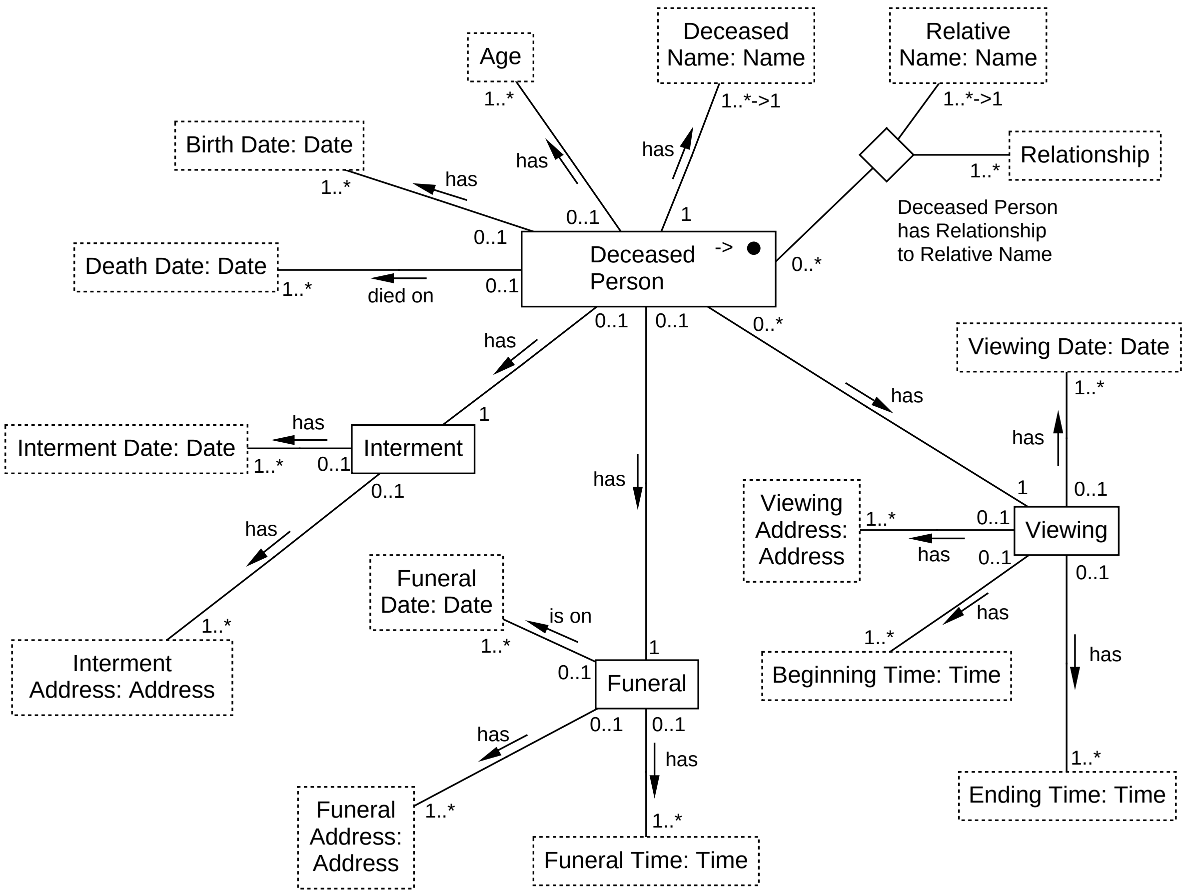


Figure 3: Sample object-relationship model instance.

```
Deceased Person [-> object];
Deceased Person [1] has Deceased Name [1..* -> 1];
...
Deceased Person [0..*] has Relationship [1..*] to Relative Name [1..* -> 1];
...
Funeral [0..1] is on Funeral Date [1..*];
...
Birth Date, Death Date, Interment Date, Viewing Date, Funeral Date : Date;
...
```

Figure 4: Sample textual object-relationship model instance.

declare that a name either identifies the decedent or the family relationship of a relative of the decedent[1]. From these specializing constraints the system derives facts about individual obituaries; for example, it now knows that there is only one funeral, there is only one interment, there may be several viewings, there may be several relatives.

Since a model-equivalent language has been defined already for OSM [25], we can faithfully convert any OSM model instance to an equivalent textual form. We use the textual representation for parsing. Figure 4 shows part of the sample ontology written as text.

We now describe data frames. Whether an object set is lexical or nonlexical depends on whether its associated data frame [14] describes a set of possible strings as objects for the object set. In general, a *data frame* describes the relevant knowledge about an object set. If a data frame is for a lexical object set, it describes the string patterns for its constants (member objects). For example, date data frames match dates using regular expressions, and name data frames match names using a combination of regular expressions and lexicons of common first names and last names. Whether lexical or nonlexical, an associated data frame can describe context keywords that indicate the presence of an object in an object set. For example, "died" and "passed away" may be context keywords for *Death Date*; "buried" may be a context keyword for *Interment*. A data frame for a lexical object set also defines common representation conversion routines, but its main emphasis is on recognizing constants/keywords.

Figure 5 shows partial data frames for *Name*, *Relative Name*, and *Relationship*. A number in brackets (e.g. *80* in the *Name* data frame) designates the longest expected constant for the data frame; the system uses this number to generate upper bounds for declarations in our database scheme. A data frame also declares constant patterns, keyword patterns, and lexicons of constants. A pattern can be declared to be case sensitive/insensitive. Patterns are written using Perl 5 regular expression syntax. The lexicons referenced in *Name* in Figure 5 are external files consisting of a simple list of names: *first.dict* contains 16,167 first names from "aaren" to "zygmunt" and *last.dict* contains 16,522 last names from "aalders" to "zywiel". To use these lexicons, a pattern in *first.dict* is referred to as *First*; a pattern in *last.dict* is referred to as *Last*. Our rule for *Name* says that the first constant must match one of the names in the first-name lexicon, followed by one or more white-space characters, followed by one of the names in the last-name lexicon. A second rule for *Name* allows a

[1]Without the specializing constraints we would not be able to make the stronger assertion that a name functionally determines a decedent or a relative.

```
...
Name matches [80] case sensitive
    constant
             { extract First, "\s+", Last; },
             ...
             { extract "[A-Z][a-zA-Z]*\s+([A-Z]\.\s+)?", Last; },
             ...
    lexicon  {
               First case insensitive;
               filename "first.dict";
             },{
               Last case insensitive;
               filename "last.dict";
             };
end;
Relative Name matches [80] case sensitive
    constant { extract First, "\s*\(", First, "\)\s*", Last;
               substitute "\s*\([^)]*\)" -> "";
             ...
end;
...
Relationship matches [14]
    constant
             { extract "brother";       context "\bbrothers?\b"; },
             { extract "sister";        context "\bsisters?\b"; },
             ...
             { extract "step-?father";  context "\bstep-?father\b";
                                        filter  "-"; },
             { extract "\bstepfather\b"; }
             ...
    keyword "\bspouse\b",
            "\bmarried\b",
            ...
end;
...
```

Figure 5: Sample data frames.

pattern to match a string of letters starting with a capital letter (i.e. a first name, not necessarily in the first-name lexicon), followed by white space, optionally followed by a capital-letter/period pair (a middle initial), followed by white space and one of the names in the last-name lexicon.

The *Relative Name* data frame in Figure 5 is a specialization of the *Name* data frame. In many obituaries, spouse names of blood relatives appear parenthetically inside names. In Figure 1, for example, we find “Anne (Dale) Elkins”. Here, Anne Elkins is the sister of the decedent, and Dale is the husband of Anne. To extract the name of the blood relative, the *Relative Name* data frame applies a substitution that discards the (optional) parenthesized name when it extracts a relative name. Besides *extract* and *substitute*, a data frame may also use *context* and *filter* clauses, which are illustrated in the *Relationship* data frame. A *context* clause defines the context for an extraction; a *filter* clause defines what to filter out for an extraction. The third rule in the *Relationship* data frame in Figure 5 has context and filter clauses indicating that if the system finds "... step-father ...", it extracts "step-father" and filters out the hyphen, leaving "stepfather". Thus constants extracted using this rule are treated the same as constants extracted using the fourth rule for *Relationship* (which only matches the word "stepfather" without a hyphen).

3.2 Unstructured Record Extraction

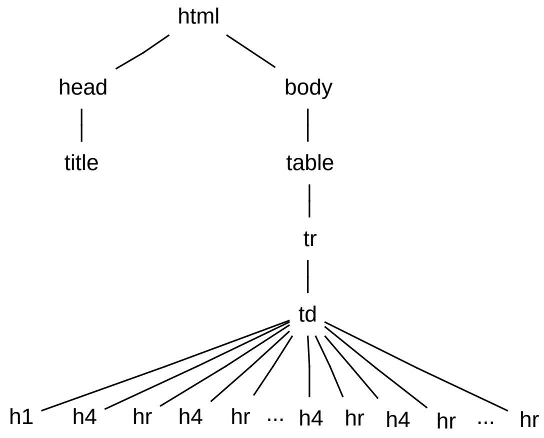
In our data extraction approach, obtaining pages of interest from the Web is done in two steps: (1) identify pages containing multiple records of interest with respect to the given application ontology, and (2) separate the information on such pages into records. A record is a chunk of data that represents one instance of the main item specified in the ontology. For obituaries, these two steps become: (1) find pages containing obituaries, (2) partition these pages into individual obituaries. We are still working on the first problem: identifying pages with multiple records of interest; we do not report on this work here. Instead, we assume we are given HTML pages containing multiple records of interest with respect to the ontology.

Our approach (1) builds a tree of the page's structure, (2) heuristically searches the tree for the subtree most likely to contain the records, and (3) heuristically finds the most likely separator among siblings in this subtree of records. Our HTML-tag tree approach is applicable both to Web pages created using an automated tool and to pages written manually. When an automated tool is used, there is a high likelihood that the use of HTML tags will be consistent across records. For HTML documents with multiple records that are generated manually, it is usually true that the creator of these documents follows certain patterns, though these patterns are not necessarily rigid or well-formed. To deal with inconsistency within a Web page and variation across individually consistent pages, we use multiple heuristics in identifying record separators. The experimental results of applying the different heuristics on randomly chosen sets of HTML documents show a high success rate in determining the correct record separators [18]. The results support our claim that the tag-tree approach is reliable and general. We now describe our implementation.

HTML tags define discrete regions within an HTML document. Some HTML start tags have corresponding end tags that together determine the boundary of a region in an HTML

```
<html><head><title>Classifieds</title></head>
<body bgcolor="#FFFFFF">
<table width="475">
<tr><td>
<h1 align="left">Funeral Notices</h1>
<h4> </h4>
<hr size="4" noshade>
<h4> Lemar K. Adamson ...</h4>
<hr>
...
<h4> Brian Fielding Frost ...</h4>
<hr>
<h4> Leonard Kenneth Gunther ...</h4>
<hr>
...
<hr>
</td></tr>
</table>
All material is copyrighted.
</body>
</html>
```

(a) A sample obituary HTML document



(b) Tag-tree of HTML document in (a).

Figure 6: An HTML document and its tag-tree.

document. Between a start-tag/end-tag pair (or start-tag/implied-end-tag pair), other tags can be nested. Based on the nested structure of start and end tags, we build what is called a *tag tree*. Figure 6(a) gives part of a sample obituary HTML document, and Figure 6(b) gives its corresponding tag tree. Figure 6(a) shows that the tag pair *<html>*-*</html>* surrounds the entire document; thus, *html* becomes the root of the tag-tree. Similarly, *title* is nested within *head*, which is nested within *html*; *body* (which has its own nested structure) is a sibling to *head*. The leaves nested within the *<td>*-*</td>* pair are the ordered sequence of sibling nodes *h1*, *h4*, *hr*, *h4*, ... A node in a tag tree contains the tag of each region and its associated text. Because of space restrictions we do not show text in Figure 6(b). But the text field for the *title* node would be “Classifieds”; the text field for the first *h4* field following the first ellipsis in the leaves would be the obituary for Brian Fielding Frost.

To detect the region in the document containing the records of interest, we search the tag tree for the subtree with the largest fan-out—*td* in Figure 6(b). Call this subtree S. If a document matches the desired profile (i.e. it contains many records of interest), S generally contains the records we wish to extract[2]. Other portions of the tag tree likely contain page headers or trailers. To extract records we restrict our attention to S, and we look for an HTML tag that separates records in S. Tags that are direct children of the root of S are called *candidate tags* (*h1*, *h4*, and *hr* in our example). We discard any candidate tags that

[2]This property is a consequence of our key assumption that the Web page contains multiple records of interest. Given this assumption, it is highly unlikely that the largest-fan-out subtree does not contain the multiple records of interest. In our experiments, we never found such a pathological example.

occur relatively few times (*h1* in this case).

To discover the record separator, we independently rank the candidate tags using the following five heuristics.

- *HC* is the *Highest Count* heuristic. It ranks the candidate tags on the number of occurrences. *HC* assumes that the separator tag occurs frequently when there are many records.

- *IS* is the *Identifiable Separator* heuristic. It uses a predetermined list of ranked HTML separator tags. Both hand-created HTML documents and tool-generated HTML documents tend to use common separator tags consistently. Our current list is **hr**, **td**, **tr**, **a**, **table**, **p**, **br**, **h4**, **h1**, **strong**, **b**, **i**.

- *SD* is the *Standard Deviation* heuristic. It assumes that records are similar in size. Candidate tags are ranked in order of the standard deviation of included plain text between identical tags.

- *RP* is the *Repeating Pattern* heuristic. It assumes that divisions between records are formed by several tags that consistently occur in the same order (e.g. a **br** followed immediately by an **hr**). If a tag pair <a><b> occurs at a record boundary and <a> is the record separator, then the count for the pair <a><b> should be about the same as the number of occurrences of <a> alone. *RP* ranks tags based on how close the total number of occurrences comes to the total number of occurrences of a particular pair in which the tag occurs.

- *OM* is the *Ontology Matching* heuristic. It is based on the ontological content of a record. Our ontology tells which fields of a record are expected to occur once and only once. Such fields are called *record-identifying fields*[3]. *OM* counts the number of such record-identifying fields on a page, and then computes X, the average of the number of occurrences of each record-identifying field. *OM* ranks the candidate tags by how closely their number of occurrences corresponds to X.

Each of these individual heuristics produces a ranking of the candidate tags. We adopt *Stanford certainty theory* [26] to obtain a consensus heuristic. Stanford certainty theory requires "certainty factors." We use the certainty factors of Table 1. These factors were

[3]E.g. because of a specializing constraint, *Deceased Name* in Figure 3 is a record-identifying field.

Heuristic Approach\Ranking	*1*	*2*	*3*	*4*
HT	49.0%	32.5%	16.5%	2.0%
IT	96.0%	4.0%	0.0%	0.0%
SD	65.5%	22.5%	12.0%	0.0%
RP	77.5%	12.5%	9.0%	1.0%
OM	84.5%	12.5%	2.0%	1.0%

Table 1: Certainty factors

obtained by running each of the individual heuristics against real data and checking the results by hand. The certainty factors in Table 1 assert, for example, that the *OM* heuristic ranked a correct separator first an average of 84.5% of the time, an incorrect separator first and a correct separator second an average of 12.5% of the time, and so forth.

The Stanford-certainty-theory formula for combining the five heuristics is

$$a + b + c + d + e - a(b + c + d + e) - b(c + d + e) - c(d + e) - de +$$
$$ab(c + d + e) + ac(d + e) + bc(d + e) + de(a + b + c) -$$
$$abcd - abce - abde - acde - bcde + abcde$$

where a, b, c, d, and e are the certainty factors, each determined by its corresponding individual heuristic.

Using the Stanford-certainty-theory formula with the certainty factors in Table 1, our algorithm determined a record separator correctly 100% of the time in the 120 Web pages we tried. The 120 pages include the pages used in the training set and those used in the test set[4].

Once a separator tag t is found, the system inserts "#####" surrounded by blank lines immediately after each occurrence of t. The system then removes all HTML tags, to yield the records of interest separated by five pound signs. Figure 7 shows the records produced for the sample HTML document of Figure 6(a). In Figure 7 the obituaries (the records of interest) occur between pairs of separators. Header and trailer information occurs before the first separator and after the last separator.

[4]The Web pages came from newspaper sites geographically distributed throughout the United States and contained records from several different application domains. See [18] for further details.

```
Classifieds


Funeral Notices
#####
Lemar K. Adamson ...
#####
...
#####
Brian Fielding Frost ...
#####
Leonard Kenneth Gunther ...
#####
...
#####


All material is copyrighted.
```

Figure 7: Obituaries extracted from the HTML document.

3.3 Database Record Generation

With the output of the ontology parser and the output of the record extractor the system proceeds to populate the database by applying two basic steps for each unstructured record. The first step produces a data-record table containing a set of descriptor/string/position tuples for those constants and keywords recognized in the unstructured record. The second step applies heuristics to this table to construct database tuples.

As Figure 2 shows, the constant/keyword recognizer applies the generated matching rules to an unstructured document to produce a data-record table. Figure 8 gives the first several lines of the data-record table produced from the obituary of Figure 1. Each entry (a line in the table) describes either a constant or a keyword. Fields of an entry are separated by a bar (|). The first field is a descriptor. For constants, the descriptor is an object-set name to which the constant may belong. For keywords, the descriptor is *KEYWORD(x)*, where x is an object-set name to which the keyword may apply. The second field is the constant or keyword found in the document (transformed by substitution rules provided in the data frame). The third field is the starting position of the constant or keyword within the record. The fourth field is the ending position of the constant or keyword. To facilitate processing, the system sorts this table on the third and fourth fields.

We now describe the construction of database tuples from the data-record table. Figure 8 gives insights into (1) the recognition of constants and keywords, and (2) the processing

```
RelativeName|Brian Fielding Frost|1|20
DeceasedName|Brian Fielding Frost|1|20
RelativeName|Brian Fielding Frost|36|55
DeceasedName|Brian Fielding Frost|36|55
KEYWORD(Age)|age|58|60
Age|41|62|63
KEYWORD(DeathDate)|passed away|66|76
BirthDate|March 7, 1998|96|108
DeathDate|March 7, 1998|96|108
IntermentDate|March 7, 1998|96|108
FuneralDate|March 7, 1998|96|108
ViewingDate|March 7, 1998|96|108
KEYWORD(Relationship)|born August 4, 1956 in Salt Lake City, to|172|212
Relationship|parent|172|212
KEYWORD(BirthDate)|born|172|175
BirthDate|August 4, 1956|177|190
DeathDate|August 4, 1956|177|190
IntermentDate|August 4, 1956|177|190
FuneralDate|August 4, 1956|177|190
ViewingDate|August 4, 1956|177|190
RelativeName|Donald Fielding|214|228
DeceasedName|Donald Fielding|214|228
RelativeName|Helen Glade Frost|234|250
DeceasedName|Helen Glade Frost|234|250
KEYWORD(Relationship)|married|257|263
Relationship|spouse|257|263
...
```

Figure 8: Sample entries in a data-record table.

required by the database-instance generator. We give three examples.

Example 1. The first four lines of Figure 8 are the string "Brian Fielding Frost." At this point the string could be either the name of the decedent or the name of the decedent's relative. Since Figure 8 has no keyword for *Deceased Person*, no keyword directly resolves this conflict. The system uses the heuristic that an important item is almost always introduced at the beginning. Using this heuristic the system infers that "Brian Fielding Frost" is the name of the decedent, not the name of one of the decedent's relatives.

Example 2. Resolution of conflicts using keywords is common. In Figure 8 consider the resolution of "March 7, 1998" of lines 8 and 9. Is it a death date or a birth date? Since the various dates are all specializations of *Date*, a particular date could be any one of five possible kinds of date. Notice, however, that "passed away", a keyword for *DeathDate*, is only 20 characters away from the beginning of "March 7, 1998". Similarly, "born", a keyword for *BirthDate*, is within two characters of "August 4, 1956". The system uses keyword proximity to resolve these conflicts.

```
Object: DeceasedPerson;
Nonlexical: Viewing {ViewingDate, ViewingAddress, ...
Lexical: Date, BirthDate, DeathDate, ...
DeceasedPerson: DeceasedName [1];
DeceasedPerson: Age [0..1];
DeceasedPerson: BirthDate [0..1];
DeceasedPerson: DeathDate [0..1];
DeceasedPerson: Funeral [0..1];
DeceasedPerson: FuneralDate [0..1];
...
Viewing: DeceasedPerson [0..*] has Viewing [1];
Viewing: Viewing [0..1] has ViewingDate [1..*];
...
DeceasedPersonRelationshipRelativeName:
    DeceasedPerson [0..*] has Relationship [1..*] to RelativeName [1];
```

Figure 9: Part of the generated record-level description.

Example 3. Consider the phrase of line 13 of Figure 8 "born August 4, 1956 in Salt Lake City, to". Line 13 of Figure 8 indicates that the recognizer tagged this phrase as a keyword for *Relationship*. Line 14 of Figure 8 indicates that the recognizer tagged this same phrase as a *Relationship* constant, with "parent" substituted for the longer phrase. The regular expression that the recognizer uses for "parent" matches "born to" with any number of intervening characters. The *Relationship* data frame states that "born to" is a keyword for a family relationship and is also a possible constant value for the *Relationship* object set, with the substitution "parent". Furthermore, the system observes that "parent" is only two characters before the beginning of the name *Donald Fielding* and twenty-two characters before the beginning of the name *Helen Glade Frost*. It therefore infers that these two people are the parents of the decedent.

The database-instance generator takes the data-record table as input along with a description of the database and constructs tuples for the extracted raw data. Figure 9 gives part of the generated record-level description, and Figure 10 gives part of the generated database scheme. The database-instance generator uses the record-level description to process a single obituary and generate appropriate insert statements for the database scheme.

We now describe the heuristics used by the database-instance generator. These heuristics are motivated by observations about the constraints in the record-level description. We classify constraint-based heuristics as singleton heuristics, functional-group heuristics, and nested-group heuristics.

- *Singleton Heuristics.* For a value that can occur at most once, we use keyword prox-

```
create table DeceasedPerson (
    DeceasedPerson integer,
    DeceasedName varchar(80),
    Age varchar(4),
    DeathDate varchar(16),
    ...
create table Viewing (
    Viewing integer,
    DeceasedPerson integer,
    ViewingDate varchar(80),
    ...
create table DeceasedPersonRelationshipRelativeName (
    DeceasedPerson integer,
    Relationship varchar(14),
    RelativeName varchar(80))
```

Figure 10: Part of the generated database scheme.

imity to find the best match (if any). example, for the generated data-record table in Figure 8 we match keyword *DeathDate* of line 7 with "March 7, 1998" and keyword *BirthDate* of line 15 with "August 4, 1956", as explained earlier.

For a value that must be present, when keyword proximity fails to find a match, the system chooses the first occurrence of a constant belonging to the corresponding object set. If no such value is present, then the system rejects the record. For our ontology, only the name of the decedent must be found.

Once the system associates a value with a singleton attribute for an object set S, it culls the data-record table by removing all other constant entries (1) whose descriptor is S or (2) whose position numbers overlap with the constant chosen for S. Thus, once the system chooses "March 7, 1998" of position 96 to 108 as the death date, it removes (1) other *DeathDate* entries and (2) all constant entries that overlap the character positions 96-108.

- *Functional-Group Heuristics.* An object set whose objects can occur several times, along with its functionally dependent object sets, constitutes a functional group. In our sample ontology *Viewing* and its functionally dependent attributes constitute such a group. Groups of such data generally occur as a unit within an unstructured document. Thus, the system looks for groups of values in close proximity within the boundaries of a context switch that pertain to the item of interest. Keywords that do not pertain to the item of interest provide boundaries for context switches. For the obituary of Figure 1, there is a *Funeral* context before the viewing information and an *Interment* context after the viewing information. Within this context the system therefore searches for *ViewingDate* / *ViewingAddress* / *BeginningTime* / *EndingTime* groups.

- *Nested-Group Heuristics.* We use nested-group heuristics to process n-ary relationship sets (for $n > 2$). Cardinality constraints can simplify the nesting and can suggest potential orders. The ontology of Figure 3 has an n-ary relationship set for capturing family relationships. This particular n-ary relationship set has strong cardinality constraints that heuristically guide the nesting. Observe in Figure 3 that for a single obituary, the n-ary relationship set connects the decedent to a relative, and *Relative Name* functionally determines the *Relationship*. This suggests that the nesting is likely to be implicit for the decedent and be otherwise hierarchical, with family relationships as roots of subtrees of information. Indeed, the obituaries we considered consistently follow this pattern. In Figure 1 we see "sons" followed by "Jordan", "Travis", and "Bryce"; "brothers" followed by "Donald", "Kenneth", and "Alex"; and "sisters" followed by "Anne" and "Sally".

The result of applying these heuristics to an unstructured obituary record is a set of generated SQL insert statements. For the obituary of Figure 1, our extraction process generated the insert statements of Figure 11. The values extracted are quite accurate, but not perfect, as illustrated by three examples. Example 1: we missed the second viewing address, which happens to have been correctly inserted as the funeral address, but not inserted as the viewing address for the second viewing. Our implementation currently does not allow constants to be inserted in two different places, but we plan to have future implementations allow for this possibility. Example 2: we obtained neither of the viewing dates, both of which can be inferred from "Thursday" and "Friday" in the obituary. Example 3: we did not obtain the full name for sons of the decedent, which can be inferred by common rules for family names. Our implementation currently finds only constants that actually occur in the document. We plan to add procedures to our data frames to do the calculations and inferences needed to obtain better results.

```
insert into DeceasedPerson
    values(1001, "Brian Fielding Frost", "41", "March 7, 1998",
    "August 4, 1956", 5001, "March 13, 1998", "350 South 1600 East",
    "12 noon", 0, "", "")
insert into Viewing
    values(7001, 1001, "", "3401 S. Highland Drive", "5", "7 p.m .")
insert into Viewing
    values(9001, 1001, "", "", "10:45", "11:45 a.m.")
insert into DeceasedPersonRelationshipRelativeName
    values(1001, "parent", "Donald Fielding")
insert into DeceasedPersonRelationshipRelativeName
    values(1001, "parent", "Helen Glade Frost")
insert into DeceasedPersonRelationshipRelativeName
    values(1001, "spouse", "Susan Fox")
insert into DeceasedPersonRelationshipRelativeName
    values(1001, "son", "Jordan")
insert into DeceasedPersonRelationshipRelativeName
    values(1001, "son", "Travis")
insert into DeceasedPersonRelationshipRelativeName
    values(1001, "son", "Bryce")
insert into DeceasedPersonRelationshipRelativeName
    values(1001, "brother", "Donald Glade")
insert into DeceasedPersonRelationshipRelativeName
    values(1001, "brother", "Kenneth Wesley")
insert into DeceasedPersonRelationshipRelativeName
    values(1001, "brother", "Alex Reed")
insert into DeceasedPersonRelationshipRelativeName
    values(1001, "sister", "Anne Elkins")
insert into DeceasedPersonRelationshipRelativeName
    values(1001, "sister", "Sally Britton")
insert into DeceasedPersonRelationshipRelativeName
    values(1001, "son", "Michael Brian Frost")
```

Figure 11: Generated SQL insert statements.

4 Results

For our test data, we took 38 obituaries from a Web page provided by the *Salt Lake Tribune* (www.sltrib.com) and 90 obituaries from a Web page provided by the *Arizona Daily Star* (www.azstarnet.com). Before running our extraction processor on these obituaries, we trained the system on several dozen other obituaries from these two newspapers. Based on this training set, we made a number of adjustments to the ontology. We then applied our developed application ontology to the test sets and obtained the results in Table 2 for the *Salt Lake Tribune* and in Table 3 for the *Arizona Daily Star*.

As Tables 2 and 3 show, we counted the number of facts (attribute values) in the test-set documents. Consistent with our implementation, which only extracts explicit constants, we counted a string as being correct if we extracted the constant as it occurred in the text.

Table 2: *Salt Lake Tribune* Obituaries

	Number of Facts in Source	Number of Facts Declared Correctly + Partially Correct	Number of Facts Declared Incorrectly	Recall Ratio	Precision Ratio
DeceasedPerson	38	38	0	100%	100%
DeceasedName	38	23+15	0	100%	100%
Age	22	20	1	91%	95%
BirthDate	30	30	1	100%	97%
DeathDate	33	31	0	94%	100%
FuneralDate	24	22	0	92%	100%
FuneralAddress	25	24	1	96%	96%
FuneralTime	29	28	0	97%	100%
IntermentDate	0	0	0	NA	NA
IntermentAddress	4	4	0	100%	100%
Viewing	29	27	1	93%	96%
ViewingDate	10	7	0	70%	100%
ViewingAddress	17	13	0	76%	100%
BeginningTime	32	28	0	88%	100%
EndingTime	29	26	0	90%	100%
Relationship	453	359+9	29	81%	93%
RelativeName	453	322+75	159	88%	71%

Table 3: *Arizona Daily Star* Obituaries

	Number of Facts in Source	Number of Facts Declared Correctly + Partially Correct	Number of Facts Declared Incorrectly	Recall Ratio	Precision Ratio
DeceasedPerson	90	90	0	100%	100%
DeceasedName	90	80+10	0	100%	100%
Age	73	63	1	86%	98%
Birthdate	26	25	1	96%	96%
DeathDate	86	72	1	84%	99%
FuneralDate	45	43	3	96%	93%
FuneralAddress	33	27	6	82%	82%
FuneralTime	50	46	7	92%	87%
IntermentDate	1	1	0	100%	100%
IntermentAddress	0	0	1	NA	0%
Viewing	29	28	0	97%	100%
ViewingDate	25	25	0	100%	100%
ViewingAddress	21	20	0	95%	100%
BeginningTime	29	27	1	93%	96%
EndingTime	22	21	0	95%	100%
Relationship	626	566+11	20	92%	97%
RelativeName	626	446+150	211	95%	74%

With this understanding, counting was basically straightforward. For names, however, we often only obtained partial names, even when more of the name was present. Because our name lexicon was incomplete and because our name-extraction expressions were not rich, we sometimes missed part of a name or split a single name into two. We list the count for these cases after the + in the *Declared Correctly* column. Partial names also caused most of the problem for the large number of incorrectly identified relatives. With a more accurate and complete lexicon and with richer name-extraction expressions, we believe that we could achieve much higher precision.

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 retrieved [19]. We compute our recall and precision ratios replacing "documents" by "facts." 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, then the recall ratio is $\frac{C}{N}$, and the precision ratio is $\frac{C}{C+I}$.

Several comments about our results are in order. Nonlexical object sets are generally easier to identify correctly because they are represented by surrogate identifiers generated by the system. For example, a *Deceased Person* is always generated for an obituary record, and consequently the results for that set are 100% precision and recall. For other nonlexical object sets such as *Viewing*, the system generates a surrogate nonlexical object when it infers the presence of one of the associated lexical objects (e.g. *Viewing Date*, *Viewing Address*, *Beginning Time*, or *Ending Time*). Thus, misses in the nonlexical object sets result from misses in the associated lexical object sets. If the system infers a viewing date, it must also infer a viewing, so if the viewing date is incorrect, the system may infer the presence of a viewing that does not exist. Likewise, if the system incorrectly fails to infer all the lexical viewing information in an obituary (i.e. viewing date, address, starting time, and ending time), it will also fail to infer a viewing that actually does exist.

Lexical object sets are divided into bounded and unbounded sets. Bounded sets include *Age*, various dates (e.g. *Birth Date*, *Death Date*), several times (e.g. *Funeral Time*), and *Relationship*. Even though these sets could theoretically be infinite, they are practically quite finite and also predictable. In today's world, an age in years will probably be an integer from 1 to 100, and perhaps up to 120, but certainly no more than 140. Date and time objects are richer than age objects, but they usually appear in well-defined forms, such as "March 23, 1998" or "6:00p.m." We can specify regular expressions to match bounded sets with a high degree of accuracy.

Errors in the bounded sets sometimes resulted from insufficiently general regular expressions. (Based on our current obituary ontology, we would miss times expressed on a 24-hour clock such as "19:30".) Such omissions are easily corrected by editing the ontology. Other errors in the bounded sets resulted from incorrectly categorizing a constant that was correctly extracted. For example, due to incomplete understanding of context, a funeral time was sometimes confused with a viewing time. Correcting this kind of error requires a careful analysis of context keywords. *Relationship* is a particularly interesting bounded set. Consider the phrase "was a member of the Daughter of the Nile and Eastern Star," which appeared in one of the obituaries. Our system inferred a parent relationship from the phrase "Daughter of," and it found "Star" in the name lexicon and thus inferred that "Star" was a parent of the decedent. Using our current architecture, there is no easy way to avoid this kind of incorrect inference.

Unbounded sets include names and addresses; it is much more difficult to specify regular expressions for unbounded sets. For this reason we incorporated lexicons into our architecture. However, as we show below, lexicons are subject to local variation, are difficult to specify completely, and still are subject to overlap and ambiguity (e.g. "May" is a month, a first name, a last name, and a place name). Names and addresses were the most difficult elements for our architecture to extract correctly, in part because of the considerable overlap between domains. For example, "Lombard Junior High" led to the inference that "Lombard" was a grandson of the decedent. We also encountered considerable variety in the way person names were expressed. Larger training sets would help solve the format variety problem. Improved context information would reduce misclassification of place names and person names; this is a more difficult problem. Improved results for unbounded sets would require larger training sets and a greater ontology-tuning effort.

This first experiment applied the ontology to a limited corpus of test obituaries from two different sources. In order to demonstrate the robustness of the approach and the general applicability of this ontology[5], we undertook a second experiment of greater quantitative and qualitative scope.

In the second experiment we collected a new corpus of obituaries which exemplified

[5]and at the suggestion of one of the anonymous referees

wider variability in style and content. First, we identified sources (in this case online newspapers) throughout the world that published local obituaries in English[6]. We chose six widely-dispersed American sources and four sources from other countries. From each of these ten sources eleven obituaries were randomly selected. Sometimes the same obituary happened to be selected more than once, so duplicates were discarded. The result was a corpus of 97 obituaries.

Without modifying the program or the ontology to treat this new corpus, and without editing the content of the individual obituaries, we submitted the new corpus to the extraction processor. The results for key facts are presented in Table 4. To summarize, performance in identifying crucial items (i.e. the decedent's name, age, death date and birth date) compared favorably with the first experiment's results.

Some interesting results emerged from this second experiment. Given the fact that the second corpus reflected a wider cultural context, there were key words and patterns that the ontology was not prepared to handle: terms for ceremonial events such as "Tenth Day Kriya" (India), "obsequies" and "cortege" (Sri Lanka), various date and address formats, the use of parenthetical or quoted nicknames and maiden names for the decedent (e.g. Edward J. "Butch" O'Hotto), and the use of different euphemisms for death ("expired", "at rest"). More problematic was the fact that the name dictionary was inadequate to handle the Maori, Sri Lankan, Hispanic, and Indian names that the extractor encountered. There were also editing-related stylistic idiosyncrasies (e.g. the use of "w/o" for "wife of") that were evident in the new corpus.

It is important to note, however, that all of these shortcomings can be addressed within our current approach. The solution involves extending the ontology's stock of patterns, specifying the new constructions in the regular-expression formalism we currently use, and extending the name lexicon. This effort reduces to an exercise in cultural localization for the domain in question.

As a final note, we observe that since our precision and recall numbers only measure individual fields, in the future we should also work on a metric for extracted data quality in the aggregate. For example, how complete are the extracted records in total, rather than just individual fields? Did the extraction problems tend to cluster within certain obituaries, or were they evenly spread out among all records? More aggregate quality

[6] The second corpus and associated Web links can be obtained by contacting the authors.

Table 4: Selected results from processing the second corpus of obituaries.

Location	Name	Age	Deathdate	Birthdate
New Zealand	4-5-0	6-0-3	6-1-2	7-0-2
Ohio	7-4-0	11-0-0	11-0-0	8-0-3
New Hampshire	5-6-0	10-0-1	10-0-1	9-0-2
New Mexico	3-7-0	10-0-0	10-0-0	6-0-4
Sri Lanka	0-7-3	5-0-5	7-0-3	9-0-1
Maine	4-6-0	10-0-0	10-0-0	9-0-1
Ireland	6-4-0	9-0-1	9-0-1	10-0-0
Minnesota	0-9-0	8-0-1	9-0-0	9-0-0
Florida	8-1-1	10-0-0	10-0-0	10-0-0
India	0-1-6	4-0-3	0-0-7	4-0-3
Totals	37-50-10	82-1-14	82-1-14	81-0-16

Each cell reports (in sequence) the number of correct, partial, and incorrect matches.

information would help us in adjusting our extraction heuristics.

5 Conclusions

We have described a conceptual-modeling approach to extract and to structure data from the Web. The approach described fully automates wrapper generation for Web documents that are rich in data, narrow in ontological breadth, and have multiple records on a single page. Instead of using page structure or HTML format as a guide to extracting data, we used a predefined ontological model instance for the chosen application. An application ontology provided the relationships among the objects of interest, the cardinality constraints for these relationships, a description of the possible strings that can populate various sets of objects, and possible context keywords to help match values with object sets. To prepare unstructured documents for comparison with the ontology, we provided a means to identify the records of interest on a Web page. With the ontology and record extractor in place, we automatically extracted records and fed them to a processor that heuristically matched them with the ontology to populate a database with the extracted data.

The results we obtained for our obituary example are encouraging. Because of the richness of the ontology, we had initially expected much lower recall and precision ratios. Achieving about 90% recall and 75% precision for names and 95% precision elsewhere was a pleasant surprise.

Although we have accomplished our goal of showing that our conceptual-modeling approach to information extraction has promise, much remains to be done. We mention seven items of future work: (1) using an ontological approach to find and classify Web pages of interest for a given application ontology, (2) strengthening our heuristics for unstructured record identification, (3) using the application ontology to design more sophisticated ways to identify records of interest both within a page or on a set of related pages, (4) improving our heuristics to identify attribute-value pairs and to construct database tuples, (5) adding richer data conversions to our data frames, (6) providing a means to do inferencing so that inferred data, as well as extracted data, can be inserted into the database, and (7) using more extensive quality metrics.

Our tools and related papers are available from our Web site, `www.deg.byu.edu`. Our current implementation uses a mixture of Perl, Java, and C++ code. Future implementation work will be done primarily in Java.

References

[1] Abiteboul, S., Cluet, S., Christophides, V., Milo, T., Moerkotte, G., Siméon, J.: Querying documents in object databases. International Journal on Digital Libraries **1** (1997) 5–9

[2] Abiteboul, S., Quass, D., McHugh, J., Widom, J., Wiener, J.: The Lorel query language for semistructured data. International Journal on Digital Libraries **1** (1997) 66–88

[3] Adelberg, B.: NoDoSE—a tool for semi-automatically extracting structured and semistructured data from text documents. Proc. 1998 ACM SIGMOD International Conference on Management of Data. (1998) 283–294

[4] Apers, P.: Identifying internet-related database research. Proc. 2nd International East-West Database Workshop (1994) 183–193

[5] Arocena, G., Mendelzon, A.: WebOQL: restructuring documents, databases and webs. Proc. Fourteenth International Conference on Data Engineering (1998)

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

[7] Atzeni, P., Mecca, G., Merialdo, P.: To weave the Web. Proc. Twenty-third International Conference on Very Large Data Bases (1997) 206–215

[8] Atzeni, P., Mecca, G.: Cut and paste. Proc. PODS'97 (1997) 144–153

[9] Buneman, P., Davidson, S., Hillebrand, G., Suciu, D.: A query language and optimization techniques for unstructured data. Proc. SIGMOD'96 (1996) 505–516

[10] Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., Widom, J.: The TSIMMIS project: integration of heterogeneous information sources. IPSJ Conference (1994) 7–18

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

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

[13] Doorenbos, R., Etzioni, O., Weld, D.: A scalable comparison-shopping agent for the World-Wide Web. Proc. First International Conference on Autonomous Agents (1997) 39–48

[14] Embley, D.: Programming with data frames for everyday data items. Proc. 1980 National Computer Conference (1980) 301–305

[15] Embley, D., Kurtz, B., Woodfield, S.: Object-oriented Systems Analysis: A Model-Driven Approach. (Prentice Hall, 1992)

[16] Embley, D., Campbell, D., Smith, R., Liddle, S.: Ontology-based extraction and structuring of information from data-rich unstructured documents. Proc. Conference on Information and Knowledge Management (CIKM'98) (1998) 52–59

[17] Embley, D., Campbell, D., Smith, R., Liddle, S.: A Conceptual-modeling approach to extracting data from the Web. Proc. 17th International Conference on Conceptual Modeling (ER'98) (1998) 78–91

[18] Embley, D., Jiang, S., Ng, Y.-K.: Record-boundary discovery in Web documents. Proc. 1999 ACM SIGMOD International Conference on Management of Data. (1999), to appear.

[19] Frakes, W., Baeza-Yates, R.: Information Retrieval: Data Structures & Algorithms. (Prentice Hall, 1992)

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

[21] Hammer, J., Garcia-Molina, H., Cho, J., Aranha, R., Crespo, A.: Extracting semistructured information from the Web. Proc. Workshop on Management of Semistructured Data (1997)

[22] Konopnicki, D., Shmueli, O.: W3QS: a query system for the world-wide Web. Proc. Twenty-first International Conference on Very Large Data Bases (1995) 54–65

[23] Kushmerick, N., Weld, D., Doorenbos, R.: Wrapper induction for information extraction. Proc. 1997 International Joint Conference on Artificial Intelligence (1997) 729–735

[24] Lakshmanan, L., Sadri, F., Subramanian, I.: A declarative language for querying and restructuring the Web. Proc. 6th international workshop on research issues in data engineering, RIDE'96 (1996)

[25] Liddle, S., Embley, D., Woodfield, S.: Unifying modeling and programming through an active, object-oriented, model-equivalent programming language. Proc. Fourteenth International Conference on Object-Oriented and Entity-Relationship Modeling (1995) 55–64

[26] Luger, G.F., Stubblefield, W.A.: Artificial Intelligence: Structures and Strategies for Complex Problem Solving, Third Edition. Addison Wesley Longman, Inc., (1998)

[27] Mendelzon, A., Mihaila, G., Milo, T.: Querying the World Wide Web. Proc. First International Conference on Parallel and Distributed Information Systems (PDIS'96) (1996)

[28] Mendelzon, A., Mihaila, G., Milo, T.: Querying the World Wide Web. International Journal on Digital Libraries **1** (1997) 54–67

[29] Smith, D., Lopez, M.: Information extraction for semi-structured documents. Proc. Workshop on Management of Semistructured Data (1997)

[30] Soderland, S.: Learning to extract text-based information from the World Wide Web. Proc. Third International Conference on Knowledge Discovery and Data Mining (1997) 251–254