A Conceptual-Modeling Approach to Extracting Data from the Web

D.W. Embley, D.M. Campbell, Y.S. Jiang, Y.-K. Ng, R.D. Smith
Department of Computer Science
Email: {embley,campbell,jiang,ng,smithr}@cs.byu.edu

S.W. Liddle*, D.W. Quass*
School of Accountancy and Information Systems
Email: {liddle,quass}@byu.edu

Brigham Young University, Provo, Utah 84602, U.S.A.
Phone: (801) 378-6470, Fax: (801) 378-7775

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 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. 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.

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

*Research funded in part by Faneuil Research Group

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 intuitive, but present severe limitations [Ape94]. 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, much more than the user can handle.

To retrieve data more efficiently from the Web, some researchers have resorted to ideas taken from databases techniques. Databases, however, require structured data and 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., [AM98]) and generating wrappers for Web pages (e.g., [AK97]).

In this paper, we discuss an approach to extracting and structuring data from documents posted on the Web that differs markedly from those previously suggested. Our proposed data extraction method is based on conceptual modeling, and, as such, we also believe that this approach represents a new direction for research in conceptual modeling.

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.

Our approach specifically focuses on 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 ontology. Neither of these definitions is exact, but they express the idea that the kinds of Web documents we are considering have many constant values and are narrow in the domain they cover.

As an example, the unstructured documents we have chosen for illustration in this paper are obituaries. Figure 1 shows an example[1]. An obituary is data rich, typically including several constants such as name, age, death date, and birth date of the deceased person; 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.

[1]To protect individual privacy, this obituary is not real. It based on an actual obituary, but it has been significantly changed so as not to reveal identities. Obituaries used in our experiment reported later in this paper are real, but only summary data and isolated occurrences of actual items of data are reported.

Specifically, our approach consists of the following steps. (1) We develop the ontological model instance over the 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 separates an unstructured Web document into individual record-size chunks, cleans them by removing markup-language tags, and presents them as individual unstructured 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 and relationships 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 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.

In an earlier paper [ECLS98], 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, essentially only allowing single constants or single sets of constants to be associated with a given item of interest (i.e., a car or a job).

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. 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 [ECLS98].

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. 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 making the information queryable, using query languages more powerful than keyword search.

One approach is to enhance traditional query languages to make them "Web aware" [AM98, KS95, LSS96, MMM96, MMM97], so that data in web pages can be queried directly. In general, these languages view the web as a graph and allow specification of queries to navigate the graph. [AM98] and [LSS96] also include the ability to query the structure of individual web pages.

Another approach is to extract the information contained within Web pages using "wrappers." The notion of using a wrapper to extract information from non-database sources has been around for several years in the area of data integration (e.g., [CGMH+94]). If the extracted information is well structured, say in the form of relational tuples, then it can be queried using a relational query language such as SQL. Otherwise, if the extracted information contains some structure, but does not conform entirely to a predefined schema, then special "semistructured" query languages may be used [ACC+97, AQM+97, BDHS96].

The approach we take in this paper to extract data uses wrappers. A wrapper for extracting data from a text-based information source generally consists of two parts: (1) extracting attribute values from the text, and (2) composing the extracted values for attributes into complex data structures. Wrappers have been written either fully manually [AM97, GHR97, HGMC+97], or with some degree of automation [Ade98, AK97, DEW97, KWD97, Sod97]. The work on automating wrapper writing focuses primarily on using syntactic clues, such as HTML tags, to identify and direct the composition of extraction of attribute values. Our work differs fundamentally from this approach to wrapper writing because it focuses on conceptual modeling to identify and direct extraction and composition (although we do use syntactic clues to distinguish between record boundaries in unstructured documents). In our approach, once the conceptual-model instance representing the application ontology has been written, wrapper generation is fully automatic.

A large body of research exists in the area of information extraction using natural-language understanding techniques [CL96]. The goal of these natural-language techniques is to extract conceptual information from the text through the use of lexicons identifying important keywords combined with sentence analysis. In comparison, our work does not attempt to extract such a deep level of understanding of the text but also does not depend upon complete sentences, which their work does. We believe our approach to be more appropriate for web pages and classified ads, which often do not contain complete sentences.

The work closest to ours is [SL97]. In this work, the authors explain how they extract information from text-based data sources using a notion of "concept definition frames," which are similar to the object definitions ("data frames") in our conceptual model. An advantage of our approach is that our conceptual model is richer, including, for example, cardinality constraints, which we use in the heuristics for composing extracted attribute values into object structures.

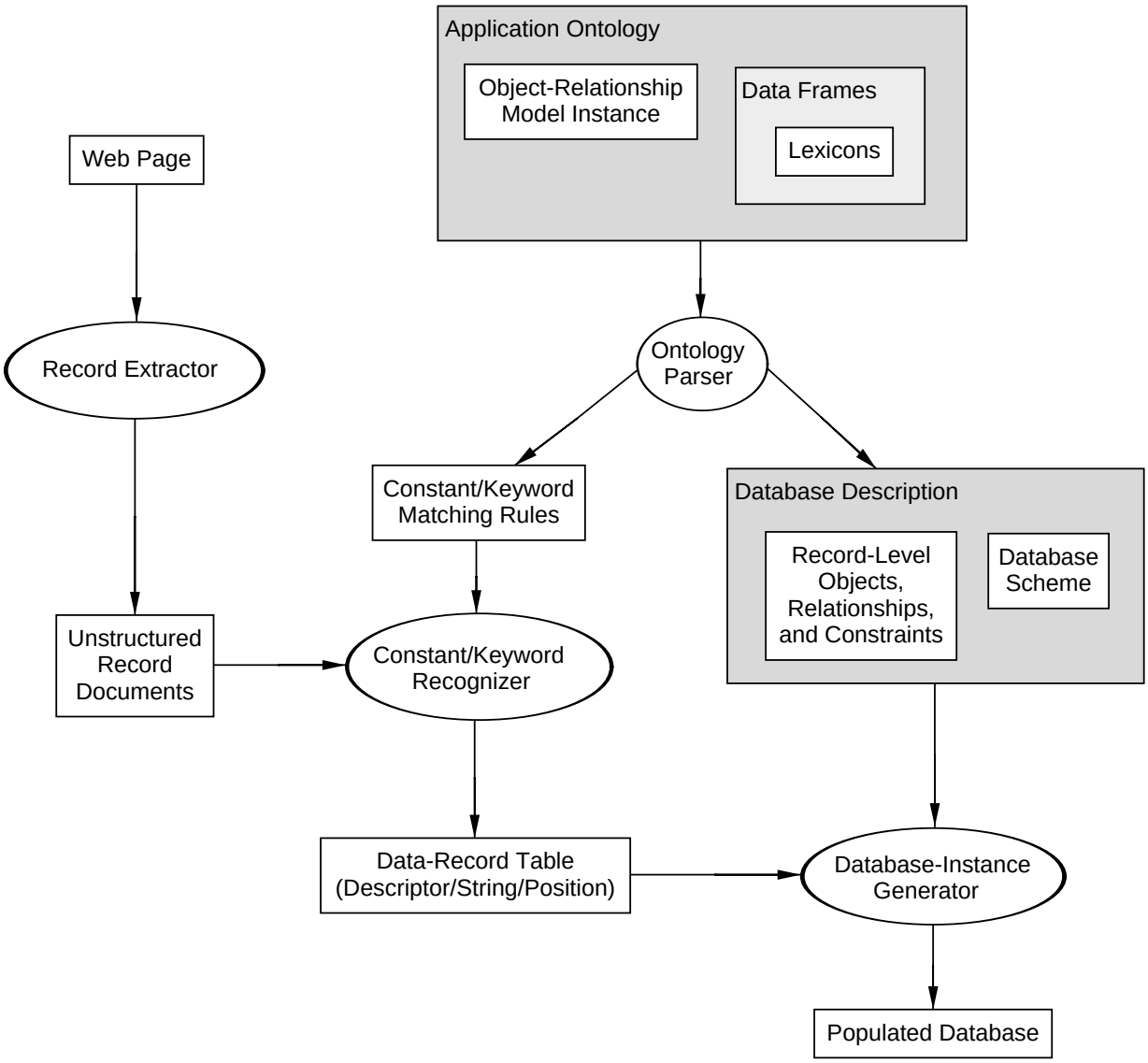


Figure 2: Data extraction and structuring process.

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. The figure also shows that the application ontology is an independent input. This ontology describes the application of interest. When we change applications, for example from car ads, to job ads, to obituaries, we change the ontology, and we apply the process to different Web pages. Significantly, everything else remains the same: the routines that extract records, parse the ontology, recognize constants and keywords, and generate the populated database instance do not change. In this way, we make the process 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 all this information as input and produces constant/keyword matching rules and a database description as output.

Figure 3 gives the object-relationship model instance for our obituary application in graphical form, and Figure 4 gives part of this model instance in an equivalent textual form. We use the Object-oriented Systems Model (OSM) [EKW92] to describe our ontology. In OSM rectangles represent sets of objects. Dotted rectangles represent lexical object sets (those such as *Age* and *Birth Date* whose objects are strings that represent themselves), and solid rectangles represent nonlexical object sets (those such as *Deceased Person* and *Viewing* whose objects are object identifiers that represent nonlexical real-world entities). 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*), and n-ary relationships have a diamond and a full descriptive name that includes the names of its connected object sets. Participation constraints near connection points between object and relationship sets designate the minimum and maximum number of times an object in the set participates in the relationship. In OSM 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 set of 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 previously explained and specialize it for a particular obituary by imposing additional constraints. We denote these specializing constraints in our notation by a "becomes" arrow (->). In Figure 3, for example, the *Deceased Person* object set becomes a single object, as denoted by "-> •", and the *1:** participation constraint on both *Deceased Name* and *Relative Name* becomes *1*. We thus declare in our ontology that an obituary is for one deceased person and that a name either identifies the deceased person or the family relationship of a relative of the deceased person. From these specializing constraints, we can also derive other facts about individual obituaries, such as that there is only one funeral and one interment, although there may be several viewings and several relatives.

Since a model-equivalent language has already been defined for OSM [LEW95], we can faithfully write any OSM model instance in an equivalent textual form. We use the textual representation for parsing. Figure 4 shows part of the sample ontology written as text. *Deceased Person [-> object]* in Figure 3, for example, denotes the *Deceased Person* object set and the fact that it "becomes" an object in an individual obituary. *Deceased Person [1] has Deceased Name [1:* -> 1]* denotes the relationship set between *Deceased Person* and *Deceased Name* along with its participation constraints, including one that is specialized for an individual obituary. Figure 4 shows a few other examples including the ternary relationship set in our example and the specializations for the date object sets.

Whether an object set is lexical or nonlexical depends on whether its associated data frame [Emb80] describes a set of possible strings as objects for the object set. In general a *data frame* describes everything we wish to know about an object set. If the data frame is for a lexical object set, it describes the string patterns for its constants (member objects).

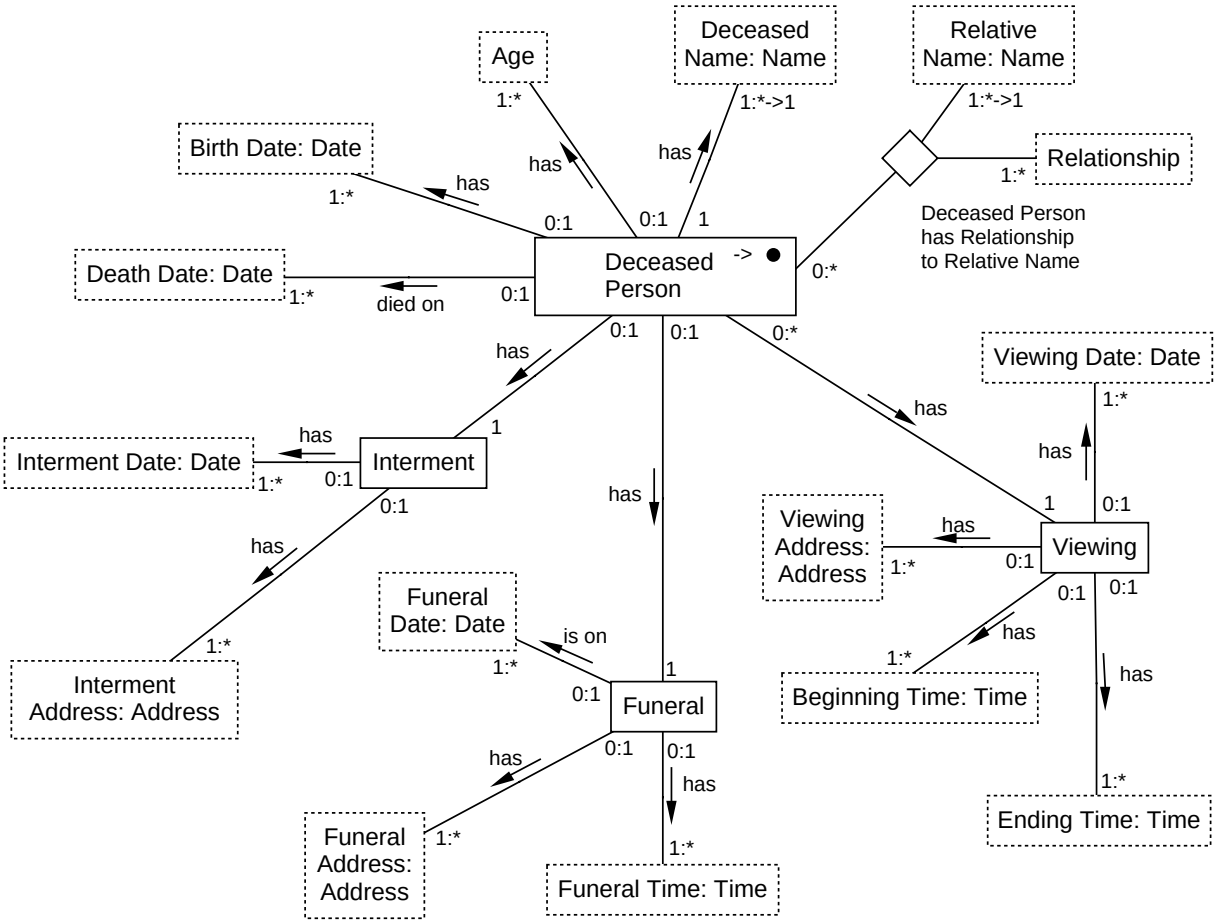


Figure 3: Sample object-relationship model instance.

```
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.

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, we may have "died" or "passed away" as context keywords for for *Death Date* and "buried" as a context keyword for *Interment*. A data frame for lexical object sets also defines conversion routines to and from a common representation and other applicable operations, but our main emphasis here is on recognizing constants and context keywords.

In Figure 5 we show as examples part of the data frames for *Name*, *Relative Name*, and *Relationship*. A number in brackets designates the longest expected constant for the data frame; we use this number to generate upper-bounds for "varchar" declarations in our database scheme. Inside a data frame we declare constant patterns, keyword patterns, and lexicons of constants. We can declare patterns to be case sensitive/insensitive and switch back and forth as needed. We write all our patterns 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". We use these lexicons in patterns by referring to them respectively as *First* and *Last*. Thus for example the first constant pattern in *Name* matches any one of the names in the first-name lexicon, followed by one or more white-space characters, followed by any one of the names in the last-name lexicon. The other pattern matches a string of letters starting with a capital letter (i.e., a first name, not necessarily in the lexicon), followed by white space, optionally followed by a capital-letter/period pair (a middle initial) and more white space, and finally a name 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 deceased, 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 parenthesized name, if any, when it extracts a possible name of a relative. Besides *extract* and *substitute*, a data frame may also have *context* and *filter* clauses, which we illustrate in the *Relationship* data frame. The *context* and *filter* clauses respectively tell us what context we must have for an extraction and what we filter out when we do the extraction. Thus, for example, as the third rule in the *Relationship* data frame in Figure 5 shows, if we see "... step-father ...", we extract "step-father" and filter it to "stepfather".

3.2 Unstructured Record Extraction

Obtaining pages of interest from the Web requires two steps: (1) classify pages as being of interest or not of interest with respect to the given application ontology, and (2) separate the information within the pages of interest into records, chunks of data that represent one instance of the main item specified in the ontology. For our obituary application, we need to find pages containing obituaries and separate the obituaries found within these pages into individual obituaries. We have not yet considered the problem of classifying pages. We believe that an approach to classification that uses the ontology to find pages of potential interest has merit, but for now we leave this as future work. We have considered some aspects of the problem of separating chunks of data into records. To extract records, we

```
...
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  "-"; },
     ...
    keyword "\bspouse\b",
            "\bmarried\b",
    ...
end;
...
```

Figure 5: Sample data frames.

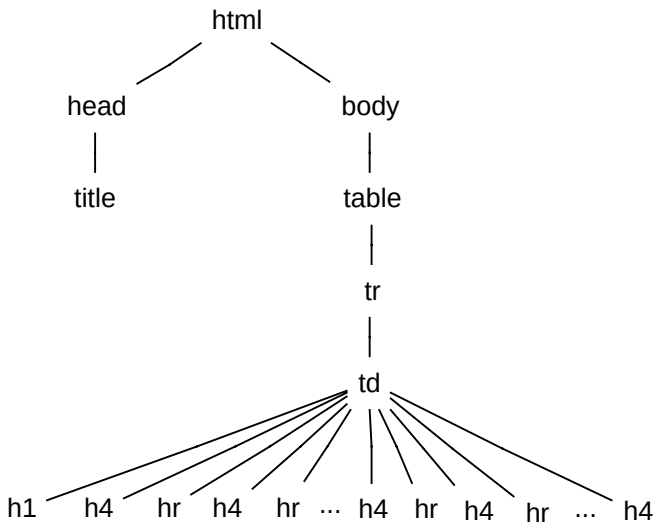
first observe that in many cases each record is already in a separate page. Although we have not yet automated this case, we believe that this is easier than the case when the records are all on the same page. When multiple records are on the same page, it is usually easy for a human to recognize boundaries that separate these records, but how do we have the system recognize these boundaries automatically?

We report here on our implementation of one possible approach to the problem of record extraction when multiple records are in a single HTML (or XML) Web page. The approach we take builds a tree of the page's structure, heuristically searches the tree for the subtree most likely to contain the records, and then heuristically finds the most likely separator among the siblings in this subtree of records. We explain the details in succeeding paragraphs. There are other approaches that may work as well (e.g., we can preclassify particular HTML tags as likely separators or match the given ontology against probable records), but we leave these for future work.

HTML tags define discrete regions within an HTML document. Some start-tags have corresponding end-tags that together determine the boundary of a region in an HTML document. Between a start-tag/end-tag pair, other tags can be nested. Based on the

```
<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.

nested structure of start- and end-tags, we build a tree called a *tag-tree*. Figure 6(a) gives part of a sample obituary HTML document, and Figure 6(b) gives its corresponding tag-tree. As Figure 6(a) shows, the tag-pair *<html>-</html>* surrounds the entire document and thus *html* becomes the root of the tag-tree. Similarly, we have *title* nested within *head*, which is nested within *html*, and as a sibling of *head* we have *body* with its nested structure. The leaves nested within the *<td>-</td>* pair are the ordered sequence of sibling of nodes *h1*, *h4*, *hr*, *h4*, ... A node in a tag-tree has two fields: (1) the first tag of each start-tag/end-tag pair or a lone tag (when there is no closing tag), and (2) the associated text. We do not show the text in Figure 6(b), but, for example, the text field for the *title* node is "Classifieds" and the text field for the first *h4* field following the first ellipsis in the leaves is the obituary for Brian Fielding Frost.

Using the tag-tree, we attempt to detect the region containing the records of interest by searching for the subtree with the largest fan-out—*td* in Figure 6(b). For documents with many records of interest, the subtree with the largest fan-out should contain these records; other subtrees represent global headers or trailers.

To find the record separators within the highest fan-out subtree, we begin by counting the number of appearances of each sibling tag below the root node of the subtree (the number of appearances of *h1*, *h4*, and *hr* for our example). We ignore tags with relatively few appearances (*h1* in our example) and concentrate on *dominant tags*, tags with many appearances (*h4* and *hr* in our example). For the dominant tags, we apply two heuristics: a Most-Appearance (MA) heuristic and a Standard-Deviation (SD) heuristic. If there is only one dominant tag, the MA heuristic selects it as the separator. If there are several dominant tags, the MA heuristic checks to see whether the dominant tags all have the same number of appearances or are within one of having the same number of appearances.

```
Classifieds

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

All material is copyrighted.
```

Figure 7: Obituaries extracted from the HTML document.

If so, the MA heuristic selects any one of the dominant tags as the separator. If not, we apply the SD heuristic. For the SD heuristic, we first find the length of each text segment between identical dominant tags (e.g., the lengths of the text segments between each successive pair of *hr* tags and between each successive pair of *h4* tags). We then calculate the standard deviation of these lengths for each tag. Since the records of interest often all have approximately the same length, we choose the tags with the least standard deviation to be the separator.

When the separator is found, we insert "#####" immediately after each tag chosen as the separator. We then remove all tags and obtain the records of interest separated by five pound signs. Figure 7 shows result for the sample HTML document in Figure 6(a). In Figure 7 the obituaries (which are the records of interest) appear between pairs of separators. Header and trailer information appears before the first and after the last separator.

3.3 Database Record Generation

With the output of the ontology parser and the record extractor in hand, we proceed with the problem of populating the database. To populate the database, we iterate over two basic steps for each unstructured record document. (1) We produce a descriptor/string/position table consisting of constants and keywords recognized in the unstructured record. (2) Based on this table, we match attributes with values and 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 our sample obituary in Figure 1. Each entry (a line in the table) describes either a constant or a keyword. We separate the fields of an entry by a bar (|). The first field is a descriptor: for constants the descriptor is an object-set name to which the constant may belong, and 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, possibly transformed as it is extracted according to substitution rules provided in a data frame. The last two fields

```
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.

give the position as the beginning and ending character count for the first and last symbol of the recognized constant or keyword. To facilitate later processing, we sort this table on the third field, the beginning character position of the recognized constant or keyword.

A careful consideration of Figure 8 reveals some interesting insights into the recognition of constants and keywords and also into the processing required by the database-instance generator. Notice in the first four lines, for example, that the string "Brian Fielding Frost" is the same and that it could either be the name of the deceased or the name of a relative of the deceased. To determine which one, we must heuristically resolve this conflict. Since there is no keyword here for *Deceased Person*, no keyword directly resolves this conflict for us. However, we know that the important item in a record is almost always introduced at the beginning, a strong indication that the name is the name of the deceased, not the name of one of the deceased's relatives. More formally, since the constraints on *DeceasedName* in Figure 9 require a one-to-one correspondence between *DeceasedName* and *DeceasedPerson* (the central item of the record) and since *DeceasedName* is not optional, the first name that appears is almost assuredly the name of the deceased person.

Keyword resolution of conflicts is common. In Figure 8, for example, consider the resolution of the death date and the birth date. Since the various dates are all specializations of *Date*, a particular date, without context, could be any one of the different dates (e.g., "March 7, 1998" might 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

```
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: Generated description of record-level objects, relationships, and constraints.

"March 7, 1998", giving a strong indication that it is the death date. Similarly, "born", a keyword for *BirthDate*, is within two characters of "August 4, 1956". Keyword proximity easily resolves these conflicts for us.

Continuing with one more example, consider the phrase "born August 4, 1956 in Salt Lake City, to", which is particularly interesting. Observe in Figure 8 that the recognizer tags this phrase as a keyword for *Relationship* and also in the next line as constant for *Relationship*, with "parent" substituted for the longer phrase. The regular expression that the recognizer uses for this phrase matches "born to" with any number of intervening characters. Since we have specified in our *Relationship* data frame 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", we emit both lines as shown in Figure 8. Observe further that we have "parent" close by (two characters away from) the beginning of the name *Donald Fielding* and close by (twenty-two characters away from) the beginning of the name *Helen Glade Frost*, which are indeed the parents of the deceased.

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.

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

- *Singleton Heuristics.* For values that should appear at most once, we use keyword proximity to find the best match, if any, for the value. For example, for the generated data-record table in Figure 8 we match *DeathDate* with "March 7, 1998" and *BirthDate* with "August 4, 1956" as explained earlier. For values that must appear at least once, if keyword proximity fails to find a match, we choose the first appearance of a constant belonging to the object set whose value must appear. If no such value appears, we reject the record. For our ontology, only the name of the deceased must

```
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: Generated database scheme.

be found. Once we declare that a value associates with a singleton attribute for an object set S, we cull the data-record table by removing all other constant entries whose descriptor is S or whose position numbers overlap with the constant chosen for S. Thus, once we declare that "March 7, 1998" is the death date, we remove all other *DeathDate* entries and all constant entries that overlap the character positions 96-108.

- *Functional-Group Heuristics.* An object set whose objects can appear several times, along with its functionally dependent object sets constitutes a functional group. In our sample ontology *Viewing* and its functionally dependent attributes constitutes such a group. Groups of such data generally appear as a unit within an unstructured document; for human readability, other possibilities would necessitate repeated identification. We heuristically make use of this observation by looking 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 our example (see Figure 1), we have a *Funeral* context before the viewing information and an *Interment* context after the viewing information. Within this context we search for *ViewingDate* / *ViewingAddress* / *BeginningTime* / *EndingTime* groups.

- *Nested-Group Heuristics.* We use nested-group heuristics to process n-ary relationship sets (for $n > 2$). Writers often produce these groups by a nesting structure in which one value is given followed by its associated values, which may be nested, and so forth. Cardinality constraints can simplify the nesting and suggest potential orders. For our sample ontology, we have one n-ary relationship set for capturing family relationships. This particular n-ary relationship set has some strong cardinality constraints that heuristically guide the nesting. Observe in Figure 3 that for a single obituary, the n-ary relationship set connects the deceased person, and *Relative Name* functionally determines the *Relationship*. This heuristically indicates that the nesting is likely to be implicit for the deceased person 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 in Figure 1, our extraction process generated the insert statements in Figure 11. Observe that the values extracted are quite accurate, but not perfect. For example, we missed the second viewing address, which happens to have been correctly inserted as the funeral address, but not also 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. Also, we obtained neither of the viewing dates, both of which can be inferred from "Thursday" and "Friday" in the obituary. We also did not obtain the full name for some of the relatives, such as sons of the deceased, which can be inferred by common rules for family names. At this point our implementation only finds constants that actually appear in the document. In future implementations, we would like to add procedures to our data frames to do the calculations and inferences needed to obtain better results.

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 tried it on several dozen other obituaries from these two newspapers. Based on this training set, we made a number of adjustments. We then applied our developed application ontology to the test sets and obtained the results in Table 1 for the *Salt Lake Tribune* and in Table 2 for the *Arizona Daily Star*.

As Tables 1 and 2 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 appeared in the text. With this understanding, counting was basically straightforward. For names, however, we often only obtained partial names. Because our name lexicon was incomplete and our name-extraction expressions were not as rich as possible, 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. We noted that this also caused most of the problem of 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 [FBY92]. To compute our recall and precision ratios, we let *facts* be *documents*. If N is the number of facts in the

Table 1: *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 2: *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	100%
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%

```
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.

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 $\frac{C}{C+I}$.

5 Conclusions

We described a conceptual-modeling approach to extracting and structuring data from the Web. The approach described fully automates wrapper generation for documents that are rich in data, narrow in ontological breadth, and appear on a Web page with limited structure. Instead of using the page structure as a guide to extracting data, we use a predefined ontological model instance for the chosen application. An application ontology provides 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 expected to help match values with object sets. To prepare unstructured documents for comparison with the ontology, we also provide a means to identify the records of interest on a Web page. With the ontology and record extractor in

place, we can automatically extract records and feed them one at a time to a processor that heuristically matches them with the ontology and populates 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 have already mentioned several items of future work, including (1) using an ontological approach to find and classify individual pages and sets of pages as being of interest for a given application ontology, (2) strengthening our heuristics for unstructured record identification, (3) using the application ontology as a means 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 for identifying attribute-value pairs and constructing database tuples, (5) adding richer data conversions and additional constraints to our data frames, and (6) providing a means to do inferencing so that inferred data as well as extracted data can be inserted into the database.

References

[ACC+97] 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.

[Ade98] B. Adelberg. Nodose - a tool for semi-automatically extracting structured and semistructured data from text documents. To appear in Proceedings of SIGMOD'98, 1998.

[AK97] N. Ashish and C. Knoblock. Wrapper generation for semi-structured internet sources. *SIGMOD Record*, 26(4):8–15, December 1997.

[AM97] P. Atzeni and G. Mecca. Cut and paste. In *Proceedings of the PODS'97*, 1997.

[AM98] G.O. Arocena and A.O. Mendelzon. Weboql: Restructuring documents, databases and webs. In *Proceedings of the Fourteen International Conference on Data Engineering*, February 1998.

[Ape94] P. M. G. Apers. Identifying internet-related database research. In *Proceedings of the 2nd International East-West Database Workshop*, pages 183–193, Klagenfurt, 1994. Springer-Verlag.

[AQM+97] 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.

[BDHS96] 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.

[CGMH+94] 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.

[CL96] J. Cowie and W. Lehnert. Information extraction. *Communications of the ACM*, 39(1):80–91, January 1996.

[DEW97] 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.

[ECLS98] D.W. Embley, D.M. Campbell, S.W. Liddle, and R.D. Smith. Ontology-Based Extraction and Structuring of Information from Data-Rich Unstructured Documents. Submitted for Publication, February 1998.

[EKW92] 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.

[Emb80] D.W. Embley. Programming with data frames for everyday data items. In *Proceedings of the 1980 National Computer Conference*, pages 301–305, Anaheim, California, May 1980.

[FBY92] W.B. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures & Algorithms*. Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[GHR97] A. Gupta, V. Harinarayan, and A. Rajaraman. Virtual database technology. *SIGMOD Record*, 26(4):57–61, December 1997.

[HGMC+97] 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.

[KS95] D. Konopnicki and O. Shmueli. W3QS: 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.

[KWD97] N. Kushmerick, D.S. Weld, and Doorenbos. Wrapper induction for information extraction. In *Proceedings of the 1997 International Joint Conference on Artificial Intelligence*, 1997.

[LEW95] 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.

[LSS96] 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.

[MMM96] A. Mendelzon, G. Mihaila, and T. Milo. Querying the world wide web. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems (PDIS'96)*, 1996.

[MMM97] 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.

[SL97] 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.

[Sod97] 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.