Conceptual-model-based data extraction from multiple-record Web pages

D.W. Embley ¹, D.M. Campbell ², Y.S. Jiang, S.W. Liddle *,³, D.W. Lonsdale ⁴, Y.-K. Ng ⁵, R.D. Smith

Data Extraction Group, Brigham Young University, Provo, Utah, UT 84602, USA

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. © 1999 Elsevier Science B.V. All rights reserved.

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 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. Fig. 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

**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.

Fig. 1. A sample obituary.
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 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 [22]. 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 (NLP), semistructured data, and Web queries.

Junglee coined the phrase ‘virtual database technology’ [22]. 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’ is 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 TSIM-MIS 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 [6,23], (iii) page grammars [7], and (iv) concept definition frames [32]; these approaches are all similar. Wrappers have been written either fully manually [8,22,23], or with some degree of automation [3,6,13,26,33]. 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.

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 [19,20,24]. 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’ [33] 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’ [32] 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’ [5,25,27,30,31]. 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

Fig. 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. Fig. 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 Fig. 2.

3.1. Ontological specification

As Fig. 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.

Fig. 3 gives the object-relationship model instance for our obituary application in graphical form; Fig. 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 Fig. 3, by adding ‘→ † ’ to the *Deceased Person* object set, we specify that *Deceased Person* becomes a single object, meaning that in an obituary there is exactly one deceased person of interest. By adding ‘→ 1’ to the participation constraint on *Relative Name*, we specify that within an obituary, a name identifies a related person. From these specializing constraints the system derives facts about individual obituaries. For example, the system now knows that for a single obituary, the name of the deceased person (as well as several other fields) become identifiers for the decedent and that relative names functionally determine relationships.

Since a model-equivalent language has been defined already for OSM [28], we can faithfully convert any OSM model instance to an equivalent textual form. We use the textual representation for parsing. Fig. 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.

Fig. 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 Fig. 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 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 Fig. 5 is a specialization of the Name data frame. In many obituaries, spouse names of blood relatives appear parenthetically inside names. In Fig. 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 Fig. 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).

Ontology construction requires a significant manual effort by someone who is skilled with regular expressions and conceptual modeling. This process is much like software development: there are analysis, design, implementation, and testing steps. For each application domain of interest, a new ontology must be constructed. We begin by gathering examples of Web pages that contain multiple records for the given application domain. Then we build an ORM and data frames to accomplish the extraction. The ontology is tested against the examples and successively refined until the extracted data is sufficiently correct and complete. Most of the refinement involves modifying regular expressions within the data frames. For our obituary ontology, there are 19 object sets, 17 relationship sets, 21 data frames, and 219 unique regular expressions written for keyword, extract, context, substitute, filter, and except clauses. The effort involved to create and debug this application ontology was on the order of 1 or 2 man-weeks of effort (in contrast with top-level ontology development, which is much more complex and generally involves man-years of effort).

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 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. Fig. 6 gives part of a sample obituary HTML document, and Fig. 7 gives its corresponding tag tree. Fig. 6 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 \(h1, h4, hr, h4, \ldots\) 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 Fig. 7. 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.

```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> Leman 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>
```

Fig. 6. A sample obituary HTML document.
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 Fig. 7. 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. 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 (\( h_1 \), \( h_4 \), and \( hr \) in our example). We discard any candidate tags that occur relatively few times (\( h_1 \) 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 and tool-generated HTML documents tend to use common separator tags consistently. Our current list is \( hr, td, tr, a, table, p, br, h_4, h_1 \), 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 \( \langle a \rangle \langle b \rangle \) occurs at a record boundary and \( \langle a \rangle \) is the record separator, then the count for the pair \( \langle a \rangle \langle b \rangle \) should be about the same as the number of occurrences of \( \langle a \rangle \) 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.

---

6 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.
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. 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 [29] to obtain a consensus heuristic. Stanford certainty theory requires ‘certainty factors.’ We use the certainty factors of Table 1. These factors were 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

\[
\begin{align*}
    a + b + c + d + e &= a(b + c + d + e) - b(c + d + e) - c(d + e) - de + ab(c + d + e) \\
    &+ ac(d + e) + bc(d + e) + de(a + b + c) - abcd - abce - abde - acde - bcde + abcde,
\end{align*}
\]

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.

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. Fig. 8 shows the records produced for the sample HTML document of Fig. 6. In Fig. 8 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.

\[\text{Table 1} \]
\begin{tabular}{|c|c|c|c|}
\hline
Heuristic approach/ranking & 1 (%) & 2 (%) & 3 (%) & 4 (%) \\
\hline
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 \\
\hline
\end{tabular}

\( ^{7} \) E.g., because of a specializing constraint, Deceased Name in Fig. 3 is a record-identifying field.

\( ^{8} \) The Web pages came from newspaper sites geographically distributed throughout the US and contained records from several different application domains. See [18] for further details.
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 Fig. 2 shows, the constant/keyword recognizer applies the generated matching rules to an unstructured document to produce a data-record table. Fig. 9 gives the first several lines of the data-record table produced from the obituary of Fig. 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 $\text{KEYWORD}(x)$, 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. Fig. 9 gives insights into (1) the recognition of constants and keywords, and (2) the processing required by the database-instance generator. We give three examples.

Example 1. The first four lines of Fig. 9 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 Fig. 9 has no keyword for $\text{Deceased Person}$, 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 Fig. 9 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.

**Example 3.** Consider the phrase of line 13 of Fig. 9 ‘born August 4, 1956 in Salt Lake City, to’. Line 13 of Fig. 9 indicates that the recognizer tagged this phrase as a keyword for `Relationship`. Line 14 of Fig. 9 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 22 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. Fig. 10 gives part of the generated record-level description, and Fig. 11 gives part of the generated database scheme. The

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];

Fig. 10. Part of the generated record-level description.

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))

Fig. 11. Part of the generated database scheme.
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 proximity to find the best match (if any). For example, for the generated data-record table in Fig. 9 we match the 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 96–108 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 Fig. 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 Fig. 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 Fig. 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 Fig. 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 Fig. 1, our extraction process generated the insert statements of Fig. 12. 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
insert into DeceasedPerson
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")

Fig. 12. Generated SQL insert statements.

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.
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. 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 [21]. 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 $C/N$, and the precision ratio is $C/(C+I)$.

<table>
<thead>
<tr>
<th>Table 2</th>
<th><em>Salt Lake Tribune</em> obituaries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of facts in source</td>
</tr>
<tr>
<td>DeceasedPerson</td>
<td>38</td>
</tr>
<tr>
<td>DeceasedName</td>
<td>38</td>
</tr>
<tr>
<td>Age</td>
<td>22</td>
</tr>
<tr>
<td>BirthDate</td>
<td>30</td>
</tr>
<tr>
<td>DeathDate</td>
<td>33</td>
</tr>
<tr>
<td>FuneralDate</td>
<td>24</td>
</tr>
<tr>
<td>FuneralAddress</td>
<td>25</td>
</tr>
<tr>
<td>FuneralTime</td>
<td>29</td>
</tr>
<tr>
<td>IntermentDate</td>
<td>0</td>
</tr>
<tr>
<td>IntermentAddress</td>
<td>4</td>
</tr>
<tr>
<td>Viewing</td>
<td>29</td>
</tr>
<tr>
<td>ViewingDate</td>
<td>10</td>
</tr>
<tr>
<td>ViewingAddress</td>
<td>17</td>
</tr>
<tr>
<td>BeginningTime</td>
<td>32</td>
</tr>
<tr>
<td>EndingTime</td>
<td>29</td>
</tr>
<tr>
<td>Relationship</td>
<td>453</td>
</tr>
<tr>
<td>RelativeName</td>
<td>453</td>
</tr>
</tbody>
</table>
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 ‘6:00 p.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

<table>
<thead>
<tr>
<th>Table 3</th>
<th>Arizona Daily Star obituaries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of facts in source</td>
</tr>
<tr>
<td>DeceasedPerson</td>
<td>90</td>
</tr>
<tr>
<td>DeceasedName</td>
<td>90</td>
</tr>
<tr>
<td>Age</td>
<td>73</td>
</tr>
<tr>
<td>Birthdate</td>
<td>26</td>
</tr>
<tr>
<td>DeathDate</td>
<td>86</td>
</tr>
<tr>
<td>FuneralDate</td>
<td>45</td>
</tr>
<tr>
<td>FuneralAddress</td>
<td>33</td>
</tr>
<tr>
<td>FuneralTime</td>
<td>50</td>
</tr>
<tr>
<td>IntermentDate</td>
<td>1</td>
</tr>
<tr>
<td>IntermentAddress</td>
<td>0</td>
</tr>
<tr>
<td>Viewing</td>
<td>29</td>
</tr>
<tr>
<td>ViewingDate</td>
<td>25</td>
</tr>
<tr>
<td>ViewingAddress</td>
<td>21</td>
</tr>
<tr>
<td>BeginningTime</td>
<td>29</td>
</tr>
<tr>
<td>EndingTime</td>
<td>22</td>
</tr>
<tr>
<td>Relationship</td>
<td>626</td>
</tr>
<tr>
<td>RelativeName</td>
<td>626</td>
</tr>
</tbody>
</table>
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, we undertook a second experiment of greater quantitative and qualitative scope.

In the second experiment we collected a new corpus of obituaries which exemplified wider variability in style and content. First, we identified sources (in this case online newspapers) throughout the world that published local obituaries in English. We chose six widely-dispersed American sources and four sources from other countries. From each of these 10 sources 11 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 the Indian

---

9 and at the suggestion of one of the anonymous referees
10 The second corpus and associated Web links can be obtained by contacting the authors.
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 information would help us in adjusting our ex-
traction 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


David W. Embley received a B.A. in Mathematics (1970) and an M.S. in Computer Science (1972), both from the University of Utah. In 1976 he earned his Ph.D. in Computer Science from the University of Illinois. From 1976 to 1982 he was a faculty member in the Department of Computer Science at the University of Nebraska, where he was tenured in 1982. Since then he has been a faculty member in the Department of Computer Science at Brigham Young University. He is the co-director of the Object-oriented Systems Modeling (OSM) research group and the Data Extraction research group. His research interests include database systems, model-driven software development, and information extraction from data-rich unstructured documents. He is the author of Object Database Development: Concepts and Principles, Addison-Wesley, Reading, Massachusetts, 1998, and a co-author of Object-oriented Systems Analysis: A Model-driven Approach, Prentice-Hall, Englewood Cliffs, New Jersey, 1992. He is currently a member of the steering committee for the International Conferences on Conceptual Modeling (the ER Conferences), and he will chair ER 2000 to be held in Salt Lake City, Utah. He is a member of ACM and is affiliated with SIGMOD.

Douglas M. Campbell, a Phi Beta Kappa graduate of Harvard in 1967, has been a professor at Brigham Young University for 28 years. In addition to 67 refereed papers in Mathematics and Computer Science, he has written three expository books and many general interest articles. His research interests include data extraction, lexicon building, grammar checking, algorithmic analysis, and digital libraries.
Yuan Jiang received B.S. and M.S. degrees in Computer Science from Brigham Young University in 1996 and 1998, respectively. He currently works as a software engineer for Novell, in Provo, Utah. His current research interests include data mining and data extraction.

Stephen W. Liddle is a member of the Information Systems faculty of Brigham Young University’s Marriott School of Management. He graduated summa cum laude in Computer Science at BYU in 1989, and then received his Ph.D. in Computer Science from BYU in 1995. He is a member of the Object-oriented Systems Modeling and Data Extraction research groups at BYU. His current interests include data and information extraction from unstructured and semistructured sources, Web-based commerce, model-driven and object-oriented software development, and knowledge management. Stephen has helped organize several workshops, and serves on the program committees of the ER and CIKM conferences. He is a member of ACM and is an affiliate of the IEEE Computer Society.

Deryle Lonsdale received a B.Sc. from the University of Alberta in 1981. From 1981 to 1990 he worked as a computational linguist and project manager at Automated Language Processing Systems (ALPNET) in Canada and the U.S. Between 1990 and 1997 he attended graduate school at Carnegie Mellon University, receiving degrees in computational linguistics (M.Sc. in 1992 and Ph.D. in 1997). While at CMU he worked on the Kant knowledge-based machine translation system and on the NL-Soar cognitive processing system for natural language processing. In January 1998 he joined the BYU Linguistics faculty where he teaches graduate and undergraduate courses in linguistics and NLP. His interests include computational syntax and semantics, cognitive modeling of language use, and language/task integrations.

Yiu-Kai Ng is an associate professor of Computer Science at Brigham Young University. He received the B.S. and M.S. degrees in Computer Science at BYU in 1982 and 1984, respectively, and the Ph.D. degree in Computer Science at Kansas State University in 1991. He has been with the Computer Science Department at BYU since 1991. His current research interests are in Web data retrieval, semistructured data, and distributed databases. He is currently the director of the Advanced Database Applications Lab at BYU, and is actively involved in the Data Extraction research group at BYU. Yiu-Kai Ng is a member of ACM.

Randy Smith received B.S. degrees in both Mathematics and Computer Science from Brigham Young University in 1997. He received an M.S. degree in Computer Science from Brigham Young University in 1999. His primary research interests include natural language understanding and computational linguistics.