2005-01-26

A Framework for Extraction Plans and Heuristics in an Ontology-Based Data-Extraction System

Alan E. Wessman
Brigham Young University - Provo

A FRAMEWORK FOR EXTRACTION PLANS AND HEURISTICS IN AN ONTOLOGY-BASED DATA-EXTRACTION SYSTEM

by

Alan Wessman

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

April 2005

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Alan Wessman

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

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

BRIGHAM YOUNG UNIVERSITY

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

Date	David W. Embley, Chair, Graduate Committee
Accepted for the Department	
	David W. Embley, Graduate Coordinator
Accepted for the College	
	G. Rex Bryce, Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

A FRAMEWORK FOR EXTRACTION PLANS AND HEURISTICS IN AN ONTOLOGY-BASED DATA-EXTRACTION SYSTEM

Alan Wessman

Department of Computer Science

Master of Science

Extraction of information from semi-structured or unstructured documents, such as Web pages, is a useful yet complex task. Research has demonstrated that ontologies may be used to achieve a high degree of accuracy in data extraction while maintaining resiliency in the face of document changes. Ontologies do not, however, diminish the complexity of a data-extraction system. As research in the field progresses, the need for a modular data-extraction system that de-couples the various functional processes involved continues to grow.

In this thesis we propose a framework for such a system. The nature of the framework allows new algorithms and ideas to be incorporated into a data extraction system without requiring wholesale rewrites of a large part of the system's source code. It also allows researchers to focus their attention on parts of the system relevant to their research without having to worry about introducing incompatibilities with the remaining

components. We demonstrate the value of the framework by providing a implementation of it, and we show that our implementation is capable of achieving accuracy in its extraction results comparable to that achieved by the legacy BYU-Ontos data-extraction system. We also suggest alternate ways in which the framework may be extended and implemented, and we supply documentation on the framework for future use by data-extraction researchers.

ACKNOWLEDGMENTS

I am indebted to many people for their assistance and support in completing this thesis. While I cannot mention all such individuals, I would like to express particular appreciation to the following people:

To my first advisor, the late Dr. Douglas M. Campbell, whose enthusiasm, intelligence, wit, and love for both the subject matter and students he taught inspired me to pursue my educational goals. He advised me during my efforts to understand the literature and search for thesis topics, but he became incapacitated by pulmonary fibrosis and passed away. I and many others miss him greatly.

To my current advisor, Dr. Stephen W. Liddle, who has made time in a very busy schedule to discuss ideas, review my designs and code, and provide invaluable suggestions regarding my work.

To Dr. David W. Embley, who leads the Data Extraction Group (DEG) and also serves as the graduate committee chair and chair of my thesis committee, for his time and attention spent on behalf of myself and the other students in the DEG.

To my fellow members of the DEG, for their encouragement and kind association, and particularly to those upon whose work I have built or whose work has inspired mine, including Kimball Hewitt (OntologyEditor), Tim Chartrand (various tools), Troy Walker (record separator), Cui Tao (extraction from HTML tables), and Helen Chen (extracting from the hidden Web).

To my parents and siblings, whose love is a source of great support.

To my children Sarah and Emily, for the bright smiles and cries of “Daddy!” when I come home tired of computers and offices.

Finally, to my wife Shannon, who has patiently endured her husband’s busy schedule, long hours at the computer, and procrastination of a lot of important things for several years now. I’ll get right on those just as soon as I fix this little bug…

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

1.1 Purpose of data extraction

Making sense of the vast amount of information available on the World Wide Web has become an increasingly important and lucrative endeavor. The success of Web search companies such as Google demonstrates the vitality of this industry. Yet Web search has much still to deliver. While traditional search engines can locate and retrieve documents of interest, they lack the capacity to make sense of the information those documents contain. They are also susceptible to inaccuracy and increasing manipulation by some Web site owners who seek to bolster their pages' relevancy rankings [McN04].

Traditional search engines occupy the domain of *information retrieval*, which is the task of identifying from a corpus of documents those that are most relevant to a particular user query. However, after the relevant documents have been identified and ranked, it is up to the user to browse the results and attempt to make use of their information. Often, the user's need is for highly specific information buried within the documents that the search engine returns. And at times the results may exclude relevant documents because the keyword-based algorithms of the search engine lack the sophistication to understand the user's ultimate objective.

For example, a query such as "list all used car sales advertisements for Ford Mustang cars with a sales price under $15,000, located in the Pacific Northwest" will fail to yield an acceptably accurate result set on a standard keyword-based Web search engine (Figure 1). In such a case, a keyword-based search engine may exclude a document with a relevant listing because the term "Pacific Northwest" is not found, despite the fact that the listed telephone number contains an area code for the Seattle region. Or, the engine

may include a page of Toyota car listings because they are being sold through a Ford dealership that advertises new Mustang models. Moreover, the engine may return a document with a single matching record within many similar but non-matching records, requiring the user to visually scan the document for the pertinent information.

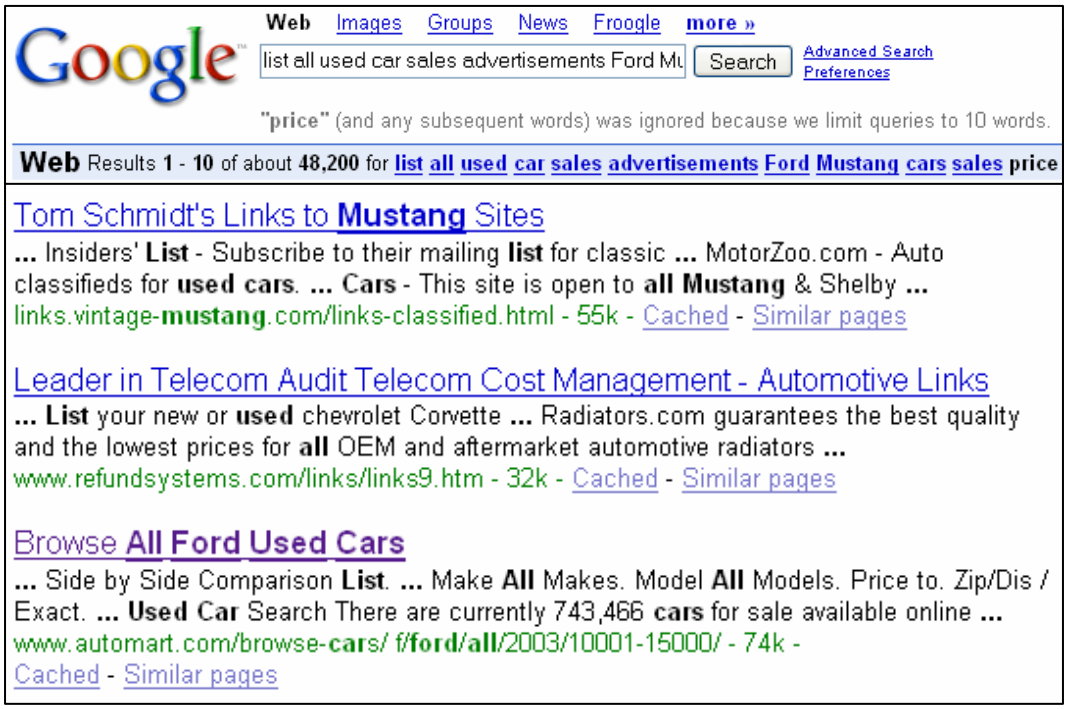

Figure 1: Google search results for a complex query.

From these examples, we see that keyword-based engines are limited by their inability to distinguish between relevant and irrelevant information on a page, or to detect key relationships between concepts in the page. Further, they cannot transform a set of relevant documents into a set of individual records representing the primary objects of interest, such as individual car listings. This prevents the user from being able to query the listings as one would a database.

The field of *data extraction* (also called *information extraction*) addresses many of the problems highlighted above. Data extraction is the activity of locating values of interest within electronic textual documents, and mapping those values to a target

conceptual schema [LR+02]. The conceptual schema may be as simple as slots in a template (a *wrapper*) used to locate relevant data within a web page, or it may be as complex as a large domain ontology that defines hierarchies of concepts and intricate relationships between those concepts. The conceptual schema is usually linked to a storage structure such as an XML file or a physical database model to permit users to query the extracted data. In this way, the meaning of a document is detected, captured, and made available to user queries or independent software programs.

Data extraction primarily focuses on locating values of interest within documents and associating the located values with a formal structure. Extraction is performed on *unstructured* or *semi-structured* documents, whose structure does not fully define the meaning of the data, as well as *structured* documents, which contain sufficient structure to allow unambiguous parsing and recognition of information [Abi97], but whose underlying schema is not fully known to the extraction process [ETL02]. A natural-language narrative is an example of an unstructured text; a Web page with sentence fragments and tables of values might be classified as semi-structured, and a comma-delimited file exported from a database is an instance of a structured document.

Data-extraction software is usually measured according to its accuracy in extracting data—minimizing the number of false or omitted mappings between value and schema, and maximizing the number of correct mappings—although other metrics, such as the amount of human interaction required to achieve a particular level of accuracy, may be applied as well [Eik99].

Much of the research in data extraction has aimed at developing more accurate wrappers while requiring less human intervention in the process. Other data-extraction

researchers have focused on the use of richer and more formal conceptual schemas (ontologies) to improve accuracy in data extraction. Among those involved in ontology-based data-extraction research are members of the Data Extraction Group (DEG) at Brigham Young University. This thesis is based on research conducted by the author and many other present and past participants of the Data Extraction Group.

1.2 Background and related work

Researchers have devised a number of different approaches to the problem of data extraction. We summarize a few of the most prominent below to establish the background for our work.

1.2.1 Wrappers

Perhaps the most common solution to the data-extraction problem is the construction and use of grammar-based wrappers [HGN+97] [KWD97] [AK97]. Wrappers use clues in the document's structure and syntax to locate useful data. The "holes" or "slots" in the wrapper, filled with data, are mapped to fields of a common semantic model, such as a data model in a relational database. Early on, wrappers were created manually [HGN+97]; subsequent research focused on automatic or semiautomatic generation of wrappers [CMM01]. Two useful surveys of wrapper research are [Eik99] and [LR+02].

The primary drawback from which most wrapper approaches suffer is their dependence upon the actual syntax of the document markup to detect the boundaries between what is and is not relevant data. This means that when the markup of a site (not the data) changes, the wrapper is invalidated. Automated generation of wrappers

alleviates this problem somewhat. Another problem is that a different wrapper is required for each unique document syntax, so thousands of wrappers may have to be generated and managed in order to adequately extract data for a particular subject domain.

1.2.2 Ontology-based extractors

Another class of data extractors (and the type of primary concern for this thesis) is the ontology-based extractors. These rely upon ontological descriptions of the subject domain as a basis for recognizing data of interest and for inferring the existence of objects and relationships that are not explicitly stated in the text.

Because an ontology describes a subject domain rather than a document, ontology-based data-extraction systems are resilient to changes in how source documents are formatted, and they can handle documents from various sources without impairing the accuracy of the extraction. This contrasts with wrappers, which merely describe the locations of data values in a particular set of similarly-formatted documents. Ontology-based extractors compare unfavorably to wrappers in one important way: considerably more human effort is required up front to construct a high-quality extraction ontology, while wrappers can be constructed more easily, even to the point of automation of much of the process.

Aside from the extraction system developed at BYU, some other notable ontology-based extraction systems have been developed at other institutions. [DMR02] uses an ontology language that distinguishes between nonlexical concepts and single- or multi-valued attributes. Identifier functions connected to the attribute definitions provide rules (such as regular expressions) that identify values from a document. Inference of

structural relationships between extracted values and nonlexical concepts is achieved by applying various heuristic algorithms based on the DOM tree of the web page. The system also employs unsupervised learning to identify concepts missed by the identifier functions but located in the same parts of the structure where identified values were found. The authors report good precision and recall results for two simple ontologies.

[SFM03] discusses HOWLIR, a framework that combines an ontology represented in DAML+OIL with the HAIRCUT information retrieval system to perform extraction, inference, annotation, and search capabilities. The system integrates a variety of technologies, most notably the AeroText™ text extraction tool and document structure analysis to extract the desired information, DAMLJessKB to make logical inferences over the extracted data, and HAIRCUT to index and retrieve the resulting semantic annotations. Being based on standards such as RDF and DAML+OIL, HOWLIR has some degree of flexibility and interoperability, but is not an entirely implementation-independent framework.

[Eng02] describes OntoWrapper, which targets semistructured sources, taking advantage of structural clues to improve accuracy. Extraction rules are represented in DAML+OIL and follow a script-like approach, referencing system and user-defined variables such as the URL of the current page or the running total of occurrences found for a particular concept. The user manually constructs a wrapper using these rules, although some aspects of the wrapper are general enough to be applicable across different document syntaxes. Still, this approach is significantly more dependent on document syntax than other ontology-based extraction systems.

1.2.3 OSM and Ontos

The data-extraction system developed by the DEG is called BYU-Ontos, or simply Ontos. It is an ontology-based engine whose present version accepts multi-record HTML documents, determines record boundaries within those documents, and extracts the data from each record. It generates SQL DDL (CREATE TABLE) statements for the model structure and stores the extracted information as DML (INSERT) statements. This facilitates querying of the results but also removes certain metadata (such as the original location of the data within the source document) attached to the data during the extraction process. This metadata may be important for learning algorithms or for further research.

The system is based on the Object-Oriented Systems Model (OSM) [EKW92]. OSM is a set-theoretic modeling approach founded upon first-order predicate logic, which enables it to express modeled concepts and constraints in terms of sets and relations. An OSM instance can serve as an ontology: concepts are represented by *object sets*, which group values (*objects*) that have similar characteristics; and connections between concepts are expressed via *relationship sets*, which group object tuples (*relationships*) that share common structure. Generalization-specialization is a special type of relation that expresses "is-a" relationships between object sets. In Ontos, OSM is expressed by OSML (OSM Language) [LEW00].

For use in extraction, OSM has been augmented by *data frames*, which serve to describe characteristics of objects, similar to an abstract data type [Emb80]. Data frames are attached to object sets, and provide a means to recognize lexical values that correspond to objects in the ontology.

OSM and data frames together provide the modeling power necessary for effective ontology-based data extraction. Experiments on small- to medium-size ontologies (two to twenty object sets) have demonstrated that Ontos exhibits a rather high degree of accuracy with the flexibility to maintain that accuracy even when the structure of the source records varies considerably [ECJ+99].

1.2.4 OntologyEditor

OntologyEditor is a predominantly WYSIWYG tool for editing OSM-based data-extraction ontologies [Hew00]. It performs no true data-extraction work itself, but provides a way for the user to preview the effect of the value recognition rules defined in the data frames of the ontology on a source document. As part of the preparation for this thesis, the author spent a significant amount of effort in refactoring OntologyEditor to accommodate new extraction ontology capabilities and to support a new ontology storage format.

1.2.5 Variations on Ontos

Considerable research touching upon Ontos and OSM has originated in the DEG. Much has focused on topics surrounding data extraction: record boundary detection [EJN99] [WE04], semiautomatic inference of data-extraction ontologies [Din03], extracting from tabular structures [ETL02] [Tao03] or from documents accessible only via HTML forms [Che04] [CEL04], and extraction into a target representation suitable for the Semantic Web [Cha03]. These various pursuits have served to exercise OSM and Ontos in ways that were difficult to predict when they were first developed.

1.3 Challenges in ontology-based data extraction

Much of the research conducted after the development of Ontos has, as a side effect, proven the system to be inflexible when certain fundamental operational parameters are changed. For instance, the system expects multiple-record documents as input, and thus performs poorly on single-record or tabular document structures. Research conducted on such document structures has required parallel versions of Ontos to be developed, or significant portions of Ontos code to be extracted and customized for newly developed systems, in order to perform effective experiments. Furthermore, Ontos is not easily adapted when new features are added to OSM or the data frames specification.

Finally, although there are many possible algorithms for extracting data based on an ontology, and it is not yet clear which are best under which circumstances, Ontos is heavily tailored to a single extraction algorithm and cannot readily be modified to execute a substantially different one. An illustration of this is the experience of a researcher who devised an extraction algorithm that involved inferring hidden Markov models from the ontology and using those to map values to concepts [Wes02]. The new algorithm could not be tied back into Ontos because the system was too strongly coupled with its original extraction algorithm, and the project was eventually abandoned.

This inflexibility makes it difficult to evaluate different ideas or approaches for data extraction. A higher number of hard-coded assumptions about operational parameters makes it more likely that reimplementation of the system is required when these assumptions are contradicted. In contrast, reducing the number of *a priori* assumptions encoded into the system should make it easier to experiment with or

improve certain aspects of the process while keeping the rest of the system constant. This allows us to make scientifically rigorous claims about the performance of the system and the impact of the changes made.

Sections 1.3.1 to 1.3.4 give a sample of the operational parameters for data extraction that we might expect a robust data-extraction system to handle flexibly without requiring extensive reimplementation.

1.3.1 Accommodating different ontology languages

One of the fundamental parameters of ontology-based extraction is the language used to represent the ontology. Different ontology languages can have different means of defining concepts, relationships, and extraction rules. They can also feature varying levels of support for modeling constructs such as generalization-specialization or incorporation of external ontologies by reference.

Ontos is written to support OSML, a declarative language with its own unique syntax for defining OSM ontologies and data frames. With new standards having been recently established by the World Wide Web Consortium (W3C) for Web-based ontology markup languages [W3C04], it is becoming increasingly important to accommodate different ontology languages without requiring completely new systems to be developed around each one. We recognize, however, that the data-extraction process can be highly dependent on the inherent limitations of the particular ontology language used, and thus we should not expect that interchanging languages would be a trivial task. Yet while some operational parameters might depend on the ontology language used, those that do not should be managed by the system so that varying the ontology language does not necessitate re-implementing the unrelated algorithms.

1.3.2 Handling various document types

Ontos is built with the assumption that the input text documents each contain a logical sequence of contiguous records, each of which represents a primary object of interest with its dependent objects. The system is able to detect record boundaries, separate the document into records, and extract the appropriate objects from the individual records with a significant degree of accuracy. Ontos is less effective at handling documents that do not match this description. These include documents with only a single primary object of interest (single-record documents), with records represented in a tabular form [Tao03], or with a complex hierarchy of records and sub-records (e.g., a university course catalog with a college-department-major-course listing hierarchy).

Ontos also relies upon a record separator module that assumes that the input documents are marked up with HTML. Other markup types such as plaintext, PDF, or XML cause the software to fail or perform poorly, since it attempts to locate HTML markup to guide its document processing logic.

A more flexible system might contain a module that can identify a document's type and choose the most appropriate routine for parsing it. It might also allow for new document parser modules to be implemented and added to the system easily in order to support new forms of markup.

1.3.3 Normalizing content

Because the record separator employed by Ontos assumes that its input documents are formatted in HTML, it strips out HTML tags from the records before extracting data. This usually makes sense because most HTML tags only format the

content, and do not by themselves lend meaning to a document. But there may be times that we wish to retain portions of the HTML markup. For example, `IMG` tags have a `SRC` attribute containing a URL to the image to be displayed; they also sometimes contain an `ALT` or `LONGDESC` attribute that presents a human-readable text description of the image [W3C99]. It may be useful to extract data from the content of these attributes rather than discarding them prior to extraction.

In addition to including content that would otherwise be excluded before processing, we may also find it useful to screen out stopwords or other content-free tokens (e.g. navigational hyperlinks, repeated whitespace characters) from the document when they would ordinarily be retained. Ontos does not provide a way to select either of these alternatives.

Ideally, a data-extraction system would allow various content filters to be interchanged as necessary in order to normalize the content before data is extracted from it.

1.3.4 Locating values of interest

Ontos locates values of interest in a document by applying rules specified in the data frame portion of an OSML ontology. These data frames support a particular set of rules governing how to recognize values, keywords, or contextual phrases. In the case of OSML, there are a number of features available for creating powerful recognition rules. These include the ability to match against terms listed in an external lexicon, substitution of symbolic macros within a recognition expression, and the specification of contextual phrases that must be found in order to accept a recognized value.

There are various aspects of value recognition that Ontos does not support. These include association of confidence measures with recognized values, use of previously extracted values as a "dynamic lexicon" for identifying values, and composite recognizers that independently locate values but act together to synthesize a canonical form of the desired data (such as a name recognizer that locates first and last names and then yields the name in "Last, First" format). Many other features can surely be invented to enhance the value recognition process.

Ontology languages other than OSML might support a different set of recognition features. Because ontology support for value recognition rules can differ across ontology languages, the code responsible for identifying values in the document based on those rules must have the same degree of modularity as the ontology processing code does. The Ontos code has some support for adding features to the current value recognition code, but the recognition code cannot easily be replaced entirely by an alternative implementation.

1.3.5 Deciding between conflicting claims on values

After identifying data values from the text through application of the recognition rules, Ontos maps those values to concepts in the ontology. Ontos also infers relationships between extracted values.

Ontos performs these actions by applying a set of heuristics that (a) decide between competing claims on the same region of text, (b) infer objects and relationships from the extracted values, and (c) generate and populate a relational structure implied by the ontology, using a modified Bernstein synthesis algorithm [Mai83].

In Ontos, the heuristics are rigidly defined and coupled to each other to such a degree that identifying each heuristic from the code is extremely difficult, and modifying or adding heuristics is even more so. Moreover, the mapping decisions it makes are binary and irreversible, prohibiting sophisticated inference techniques such as backtracking or confidence optimization.

We seek to improve Ontos to support two levels of value-mapping modularity: first, a decreased coupling between different heuristics along with ways for the heuristics to be extended, and second, an ability to replace the entire heuristics mapping module with other value-mapping implementations.

1.4 Toward solutions for data-extraction problems

Addressing the need for high flexibility, modularity, and extensibility in a data-extraction system, we propose a new framework for data extraction. We assert that such a framework will provide the support for customization and experimentation needed to efficiently conduct data-extraction research. We justify this assertion below by examining the role of frameworks in software engineering.

1.4.1 Value of frameworks and design patterns

Frameworks provide the means to address the macro-level problems of a system while allowing details to be implemented or varied after the initial design, so that components may be interchanged and system performance may be tuned. Formally, a framework is "a system of rules, ideas or beliefs that is used to plan or decide something" [Cam04]. A software framework (hereafter simply called "a framework") thus provides a skeletal implementation for a general problem, and establishes parameters for a set of

solutions based on that partial implementation. Solution providers can concentrate on satisfying those requirements that are unique to a particular approach, using the existing mechanisms provided by the framework to handle the rest of the system.

Frameworks are predominantly implemented in object-oriented languages such as C++ or Java due to their polymorphic capabilities. They consist of abstract classes and interfaces that define important concepts in the problem domain and how they interact on a general level. Frameworks may also be composed of declarative configuration files that are interpreted at runtime, allowing an implementation user to adjust operational parameters quickly without recompiling the system code.

The theory of design patterns has contributed to the rise of robust software frameworks. Design patterns are collections of templates and principles for designing code that solve commonly-encountered software design scenarios in an abstract manner. An example of a design pattern is the **Factory Method** [GHJV95], which provides a model for instantiating new objects whose actual classes are known only at runtime (dynamic binding) rather than at design time (static binding). These design patterns are often encountered in frameworks due to their highly generalized architecture, which enables frameworks to defer non-essential design decisions to framework implementations.

Frameworks are most often constructed to support complex systems whose behavior is subject to a large number of operational parameters. Examples of such systems include the Apache Struts framework for handling Web requests and responses [Apa04], the Java Collections framework that provides a library of abstract data structures (e.g., `List` or `Map`) [Sun04], or Oracle's Application Data Framework (ADF)

layer that stands between a relational data model and client- or Web-based user interfaces [Ora04]. We assert that the requirements of a generalized data-extraction system are sufficiently complex and variable to justify creation of a framework as a sensible approach to addressing these requirements.

1.4.2 Analyzing data-extraction systems to design a framework

As with any suitably powerful framework, realization of a data-extraction framework requires a substantial analytical effort. We begin the process by surveying existing data-extraction systems, noting the assumptions and dependencies inherent in each, and establishing what common functionality is necessary. Where differences in operation or operating assumptions exist, we generalize them in such a way that the framework can accommodate them. We do this either by specifying operational parameters to the execution API of the framework, or by deferring specification of the operational differences to implementations of the framework modules. Lastly, we establish a set of interfaces, design principles, and programming conventions to regulate how implementations fit into the framework.

This thesis reports on the result of an effort to construct an ontology-based data-extraction framework that supports a major new version of BYU's Ontos extraction system.

1.4.3 Thesis statement

A generalized framework of interfaces and abstract classes written in an object-oriented language (such as Java) can decouple each operational module from the rest of the engine to produce a highly flexible and configurable data-extraction system. The framework can be sufficiently flexible both to allow the current heuristics to be re-

implemented under the framework and to enable new heuristics and other modules to be created and incorporated into the Ontos system without requiring significant rewrites of unrelated code.

1.4.4 Contributions to computer science

The work performed for this thesis contributes the following to current data-extraction research.

Design and construction of a data-extraction framework

The primary work product for this thesis is an ontology-based data-extraction framework written in Java. This framework seeks to abstract or eliminate numerous operational assumptions on which individual data-extraction systems (most notably Ontos) have been based. The framework will allow orthogonal components of an implementation to be modified or replaced with minimal impact to other components of the system. This modularity will greatly aid the state of research in this area, as researchers will more easily be able to conduct well-controlled experiments focusing on specific problems in the data-extraction process without spending a great deal of effort on re-implementing aspects of the system that lie outside the focus of the research.

Concept of extraction plans

In the process of generalizing the functionality of a data-extraction system, we realized a new concept that can be introduced to this field of computer science: the *extraction plan*. This is a high-level algorithm that prescribes the execution of the data-extraction system. We propose that an extraction plan relates to a data-extraction system in much the same way that a SQL execution plan relates to a relational database system.

While extraction plans are presently defined only in compiled Java classes, we envision that they bear the potential of being defined declaratively or even computed like modern SQL execution plans are. Thus the concept of extraction plans opens up a further avenue of research.

OSMX

The complexity of maintaining and extending OSML together with its attendant parser and compiler prompted an initiative to define a new way to represent OSM and data frames, using XML as the basis for the representation. This new language, defined by an XML Schema [W3C01], is called OSMX. Porting the language to XML provides numerous benefits, including obviating the need for specialized parsers, natively enforcing constraints on how ontologies may be constructed, and allowing researchers to use open-source and other third-party tools to check their ontologies for compliance with the OSMX schema.

1.4.5 Re-implementation of Ontos under the framework

As both a demonstration of the practicality of the data-extraction framework as well as a much-needed upgrade to the core technology of the Data Extraction Group, this thesis effort contributes a re-implementation of Ontos under the new framework and based on OSMX ontologies. The new Ontos serves as a reference for future implementations of the framework. It retains support for all essential features of the old Ontos system but also adds many new capabilities. In addition to those implied by the flexibility of the framework, these new capabilities include:

- Fully transitive data frame inheritance via generalization-specialization,

- True lexicon substitution into recognizer expressions,
- Recursive macro substitution within recognizer expressions,
- Association of keyword recognition rules with both individual value recognition rules and the entire collection of value recognition rules for an object set,
- Support for confidence values for recognized and accepted matches,
- Rapid modification of the ontology traversal order for the value-mapping stage of processing,
- Retention of extracted data inline with the ontology to support later use of the data without sacrificing ontology-supplied semantic meaning, and
- Output of extracted data (objects and relationships) as nested lists in HTML, with support for other output formats via the framework.

2 A framework for performing ontology-based data extraction

This chapter explains the design and construction of the data-extraction framework. We describe each interface and abstract class, explain the essential contracts they define, and discuss how they might be implemented.

A graphical overview of the framework appears in Figure 2. Control begins at the engine at the top of the diagram, and passes to the extraction plan. The narrow boxes running down the right side represent modules involved in the extraction process.

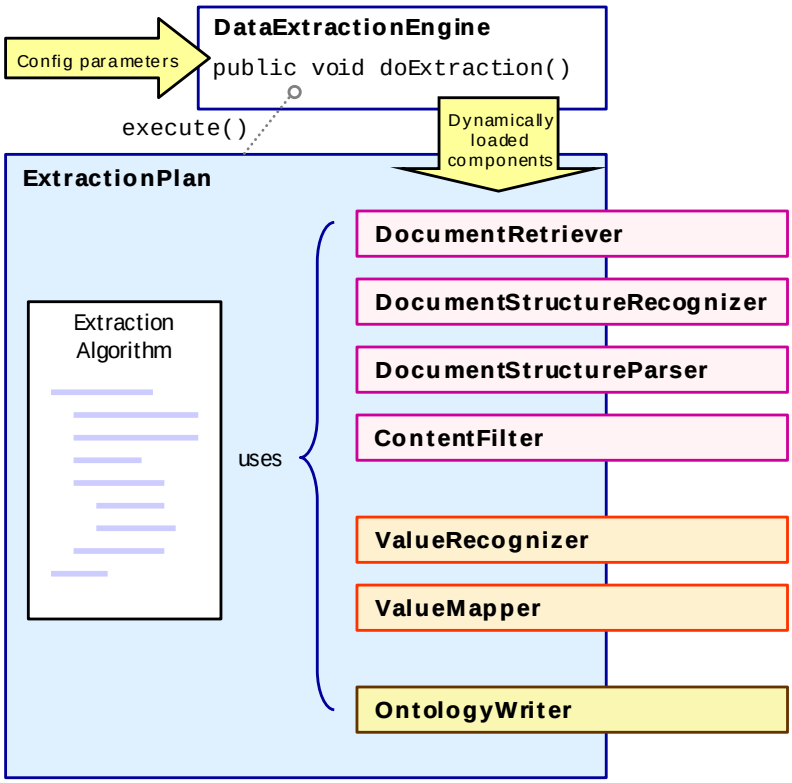

Figure 2: The data extraction framework.

2.1 Essential concepts

We first discuss the concepts central to the framework—those that touch almost every aspect of the extraction process. These interfaces and classes are as follows.

- `DataExtractionEngine`
- `Ontology`
- `Document`
- `ExtractionPlan`

We provide details on each below.

2.1.1 DataExtractionEngine

The `DataExtractionEngine` abstract class represents the overall data-extraction system. Its primary purpose is to accept operational parameters, locate and load the appropriate modules, perform any additional initialization steps, and initiate the extraction process. It then performs cleanup as necessary and terminates.

`DataExtractionEngine` follows the **Facade** design pattern [GHJV95]. A Facade is a simplified interface to a complex system; in this case, DataExtractionEngine defines simple methods for initializing the system and executing the extraction process. This pattern also allows clients such as the OntologyEditor to interact with the system at a very high level without being coupled to specific components of the system.

2.1.2 Ontology

The `Ontology` interface describes an in-memory representation of the extraction ontology. The interface is designed to be independent of the language the ontology is written in; thus we eliminate from the framework any assumption that the extraction ontology is built with OSM, DAML, OWL, or any other specific ontology language.

This decision presents us with a significant challenge. Without knowing the features of the ontology language, how can we define the capabilities of the `Ontology` interface? The answer is that we defer such details to implementation classes, and use `Ontology` as primarily a marker interface for ensuring that parameters and member variables representing the ontology are of the correct data type.

2.1.3 Document

`Document` is an interface that represents cohesive units of unstructured or semi-structured information that may be interspersed with data that is not of interest.

Two major questions to be resolved by the framework are 1) what constitutes data, and 2) what constitutes a document. For example, do we seek to extract information from graphical images, sound, video, or other forms of multimedia? Does graphical layout of a textual document according to, say, HTML markup play a role in the extraction process? Are some hyperlinked Web pages considered part of the same document as the page that linked to them?

We narrow the scope of our problem by choosing to focus only on textual data extraction. If an information source is not text-based originally but can be made to yield text data, it must be transformed into a text-based source before the extraction engine can make use of it. We do not attempt to decode images or other non-text sources of information.

The structure of a document is another problem. Structure can provide valuable clues to the extraction process; often it indicates contextual boundaries that correspond with distinct conceptual groups in the ontology. Although any text document can be expressed as a one-dimensional string of characters, we may find it more useful to

represent a complex document structure by a hierarchy of documents and sub-documents, for example. A matrix representation may be appropriate for tabular data. Hyperlinked documents would appear as nodes in a directed graph.

Due to the sheer number of possible document structures, it is infeasible to provide support for all of them. We choose the approach that seems most flexible (borne out by the success of XML), which is to represent a document as a sequence of (possibly zero-length) strings interleaved with sub-documents. This allows us, for instance, to represent a hyperlinked "drill-down" page of details as an inline sub-document, its contents taking the place of the hyperlink in the summary page.

A sub-document is itself a `Document`, and thus may contain other sub-documents. This definition implies a tree structure (cycles are not permitted) with one `Document` at the root, and we formally define the term *sub-document* to indicate any non-root node in a `Document` tree. We note that `Document` is an instance of the **Composite** design pattern [GHJV95], which allows objects to be composed into treelike hierarchies while providing a uniform interface both for interior nodes and leaves.

A Uniform Resource Identifier (URI) uniquely identifies each document or sub-document. We may use such references to track the origin of extracted data, which is an important element of evaluating data-extraction accuracy.

2.1.4 ExtractionPlan

The `ExtractionPlan` abstract class represents the overall algorithm for carrying out the extraction activity. In this way, it is similar to an SQL extraction plan in a relational database management system. `ExtractionPlan` eliminates assumptions about the order of operations for a data-extraction system: one implementation might proceed in

a linear fashion, from retrieval straight through to mapping and output, while another implementation might discover some new information (such as a relevant URL) during extraction and immediately branch into a recursive execution based on that data. In this sense, `ExtractionPlan` adheres to the **Strategy** design pattern [GHJV95], which encapsulates an algorithmic process and standardizes its invocation so that different algorithms may be interchanged.

`ExtractionPlan` is an abstract class, not an interface, because we require it to be initialized with a reference to the `DataExtractionEngine` that invokes it. This is because the `ExtractionPlan` needs to be able to access the engine's initialization parameters and system components in order to carry out its responsibilities.

2.2 Locating and preparing documents

Before actual extraction work can occur, the extraction engine must provide a way for documents to be located and prepared for extraction. The following interfaces define important aspects of this process.

- `DocumentRetriever`
- `DocumentStructureParser`
- `DocumentStructureRecognizer`
- `ContentFilter`

We describe each interface below.

2.2.1 DocumentRetriever

The `DocumentRetriever` interface (Figure 3) defines the module responsible for supplying the extraction engine with source documents. It may, for instance, represent a view of a local filesystem, or it may wrap the functionality of a Web crawler or even a

Web search engine such as Google. The module accepts a URI as input and produces a set of `Documents`. Since an extraction engine can do nothing useful without documents to process, the `DocumentRetriever` is a required component.

Figure 3: Operation of `DocumentRetriever`.

The interface is rather basic. It supports the creation and management of a list of locations (URIs) from which to retrieve the source documents. The primary function of `DocumentRetriever` is to retrieve the documents based on the locations stored in the list. We leave it to the implementations to decide whether to filter out documents based on some criteria (e.g., file extension or MIME type).

The `DocumentRetriever` will usually perform its functions at the beginning of the extraction process. However, a sophisticated implementation might use it to retrieve additional documents using extracted URLs from previously retrieved web pages, in an intelligent spidering process.

2.2.2 DocumentStructureRecognizer

`DocumentStructureRecognizer` (Figure 4) is an optional component of the system. Its role is to analyze a document to determine which available `DocumentStructureParser` is best suited to decompose the document into its structural components. This is useful, for example, when the extraction engine is operating on a mixture of source documents, wherein some may be single-record documents and others may have a multiple-record structure. In such a case, we would not want a

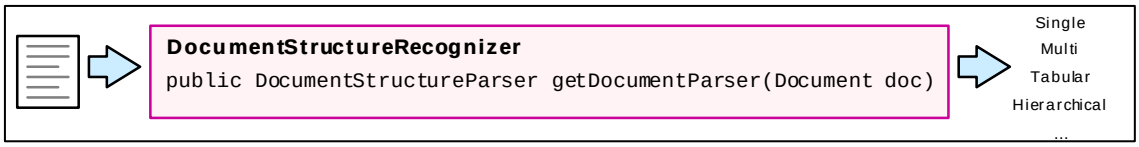

Figure 4: Operation of `DocumentStructureRecognizer`.

`DocumentStructureParser` designed for multiple-record extraction to attempt to break a single-record document into what it thinks are its constituent records. The `DocumentStructureRecognizer` intervenes to make sure that each `DocumentStructureParser` is given only those documents it is best suited to parse.

2.2.3 DocumentStructureParser

We may be interested in breaking up a document into sub-documents in order to make extraction easier or more accurate. For example, dividing a multi-record document into sub-documents, each constituting an individual record, allows us to process one record at a time without having to worry about missing a record boundary and extracting values from an adjacent record.

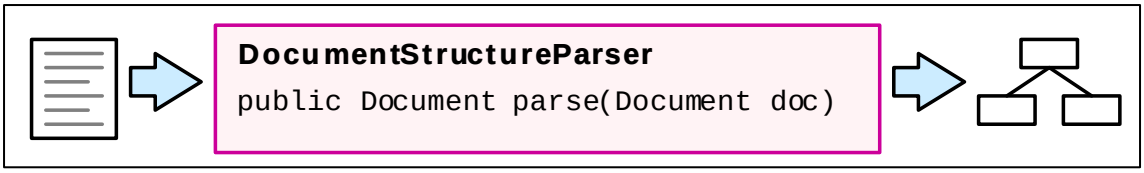

Figure 5: Operation of `DocumentStructureParser`.

We define the `DocumentStructureParser` interface (Figure 5) as a solution to this problem. It is an optional component of the system; left unspecified, the input document will be treated as an indivisible unit (and therefore a single-record document).

The interface defines one operation: `parse()`. It takes a `Document` as its single parameter and returns a `Document`. The returned document is the root node in a tree of `Document` objects. For a multiple-record document, the tree will be shallow and broad, with one root (representing the original document) and many leaves (each representing a single record). Other document structures might result in deeper trees, as in a thesis with chapters, sections within the chapters, and sub-sections below those. The document tree structure provides a natural representation for factored information: for example, a car dealer's name and contact information located at the top of a Web page could be stored in the root of the document tree, while the used-car ads associated with the dealer information would exist as individual records at the leaf level.

We generally recommend, but do not require, that implementations ensure that the contents of sibling documents within the tree do not overlap. This guards against extraction of the same value twice due to processing different sub-documents containing the same value. However, a `DocumentStructureParser` designer might find it useful to define overlapping sub-documents for some structures, such as tabular data.

2.2.4 ContentFilter

Most documents contain a combination of meaningful data and formatting information. An HTML document contains many tags that indicate how the document may be represented in a browser, but these tags usually do not lend additional meaning to the content. We find it convenient therefore to remove text that is exclusively for formatting purposes from the document before proceeding with extraction.

We define the `ContentFilter` interface to support this requirement. This is another optional component of a data-extraction system, as going without a filter simply means extracting from the document's original content.

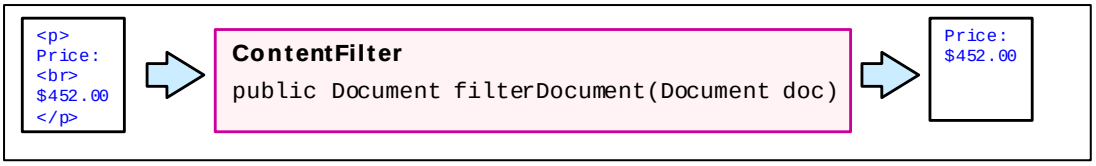

Figure 6: Operation of `ContentFilter`.

`ContentFilter` (Figure 6) is a simple interface, defining a single method `filterDocument()`. It accepts a `Document` as input and returns a `Document`, usually (but not necessarily) the same one. The general contract is that the output document's content is a filtered version of the input document's content. We note that since this contract is not enforceable at the interface level, it is merely implied by the name of the interface.

Filtering out the formatting data from a document is a process that demands a substantial degree of flexibility. For example, we may at times wish to strip all HTML tags from a document, leaving behind only the text content found between those tags. On the other hand, we might desire to preserve quasi-meaningful pieces of information found within certain HTML tags, such as the contents of the ALT attribute of an IMG tag. Consider the example of a series of icons used to depict amenities provided at a campground. The icons express information that we wish to extract into an ontology for campgrounds, and by extracting from the ALT attribute of those IMG tags we hope to glean the desired knowledge without having to attempt to decode the graphics themselves. By providing a flexible means to implement various filters, we allow

implementers to target those portions of the document that they deem most likely to yield useful data, while discarding data that simply gets in the way.

2.3 Extracting and mapping values

With the document retrieved, parsed, and filtered, we can perform the actual task of extracting values from the document and mapping them to the ontology. Two interfaces divide this work:

- `ValueRecognizer`
- `ValueMapper`

Our discussion of the functionality defined by these interfaces follows.

2.3.1 ValueRecognizer

`ValueRecognizer` (Figure 7) occupies a key role in a data-extraction system, and is a required component of the framework. Its responsibility is to apply the value-

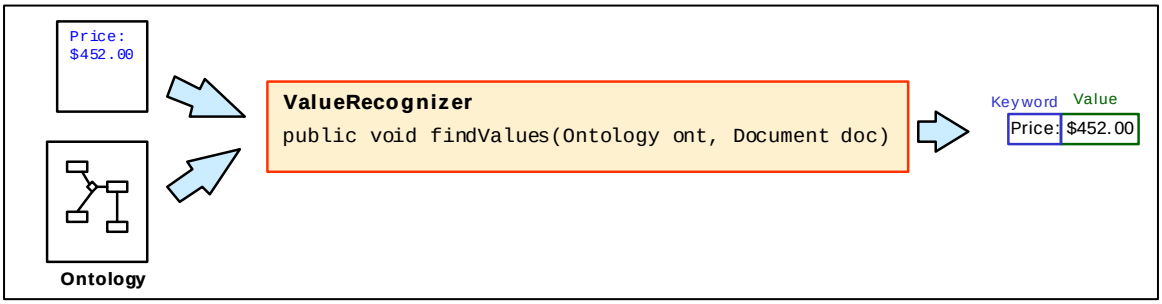

Figure 7: Operation of `ValueRecognizer`.

recognition rules associated with the extraction ontology to the input document, producing a set of candidate extractions. We say "candidate" because it does not resolve conflicts about which matched values belong to which parts of the ontology; it merely

identifies from the document the values we can find that *might* belong in the final data instance.

Locating and interpreting the extraction rules is a process that can differ according to the ontology language used, so at the framework level we do not restrict how this is done. Nor do we specify how the rules are to be applied to the document or how the matching results are to be stored. As we detail later, our reference implementation associates match values with the ontology through composition; but other methods of handling the match results (such as annotations inline with the document content) may be equally valid.

The `ValueRecognizer` also bears the responsibility of maintaining location information for each candidate value. This provides a traceable path back to the document content and also can supply useful data for the algorithms that resolve match conflicts and create mappings from candidate values to elements of the ontology. We do not specify a format for the location data, but <*start position, end position*> or <*start position, length*> pairs generally make the most sense for character-based text sources.

2.3.2 ValueMapper

Perhaps the most important and difficult part of the extraction system is the process that takes candidate value matches and uses them to build a data instance consistent with the constraints specified by the ontology. Since this process maps candidate values to elements of the ontology, we name this interface `ValueMapper`.

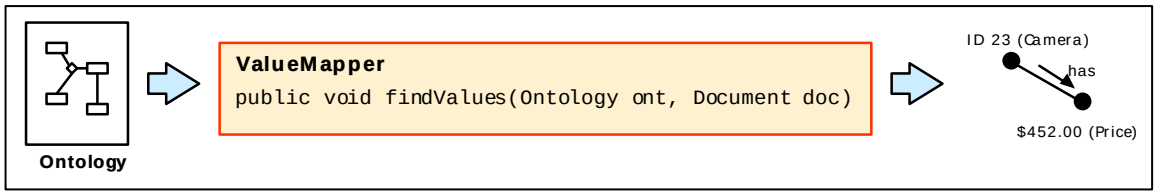

Figure 8: Operation of `ValueMapper`.

There are four tasks that a `ValueMapper` (Figure 8) must perform to transform candidate value matches into a data instance:

- Resolve conflicting claims that different elements of the ontology make upon the same matched value
- Transform lexical values into objects (instances of concepts defined in the ontology)
- Infer the existence of objects that have no direct lexical representation in the text
- Infer relationships between objects

The `ValueMapper`'s work yields a data instance: a collection of objects and relationships between those objects.

2.4 Generating output with OntologyWriter

When the `ValueMapper` process has finished, what do we do with the resulting data instance? The `OntologyWriter` abstract class (Figure 9) addresses this question. It provides a standard way for an implementation to export the objects and relationships to a useful storage format via the Java interface `Writer`. The particular storage format is up to the implementation to define.

`OntologyWriter` defines two primary methods: `writeModelInstance()` and `writeDataInstance()`. The former provides an export path for the structure (object

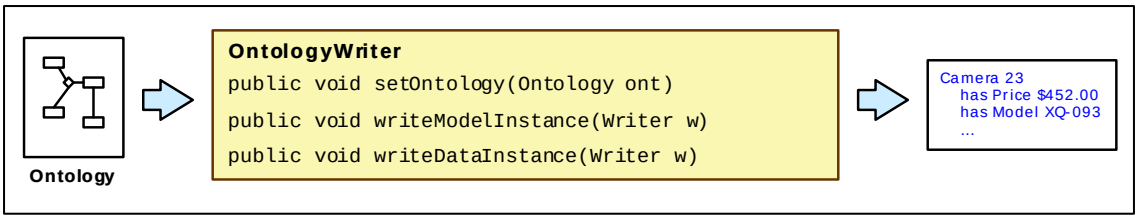

Figure 9: Operation of `OntologyWriter`.

sets, relationship sets, etc.) of the ontology, while the latter provides for the contents (the data instance, objects, and relationships) to be written to an output format.

3 Constructing extraction ontologies with OSMX

Fundamental to an implementation of the data-extraction framework is the language used to define the ontologies involved in the extraction process. The ontology language establishes the capability of the ontology to represent a given subject domain.

Our reference implementation depends upon OSM just as the legacy version of Ontos does. However, there are some key differences between the representation of OSM used by legacy Ontos (OSML) and the version used by our implementation (OSMX). Foremost of these are the syntax and grammar differences. OSML is represented by a customized declarative syntax; the new version of OSM is represented by standard XML syntax. The grammar differences are less severe; some capabilities have been added to the new representation that the old one lacked, and some lesser-used features of the old representation have been deprecated or left out of the new version, but most of the grammar productions remain essentially the same. Because of its XML syntax, we dub the new version OSMX.

The official OSMX specification is defined by an XML Schema document [Wes04]. This document defines the standards for creating a well-formed and valid OSMX document. We use the Java Architecture for XML Binding (JAXB) technology [JAXB03] to generate, from the OSMX specification, Java classes and interfaces that represent OSMX constructs. Modifying the OSMX definition is generally a straightforward process: we adjust the definitions in the XML Schema document, and then execute a JAXB program that rebuilds the classes and interfaces automatically. We use these classes and interfaces to access and manipulate portions of an ontology from within the data-extraction framework reference implementation.

This chapter explains the constructs supported by OSMX as a prelude to discussion of the data-extraction framework reference implementation in the following chapter. Because many of these constructs were introduced in the original OSM research [EKW92], we do not go beyond a general discussion of these except where OSMX has introduced new aspects or capabilities. We provide more detailed discussion for concepts that originated in OSMX.

3.1 Object sets

OSM is built upon formal mathematical principles, and at its core is a representation of first-order predicate logic. It therefore deals with sets and relations. Perhaps the most important construct of OSM is the *object set*.

3.1.1 General concept

An object set represents a concept or classification in an OSM ontology, similar to an abstract data type. As the name implies, it is treated as a set of *objects*, which represent concrete instances or values. For example, *Car* might be the name for an object set, while the automobile with a particular Vehicle Identification Number would be an object belonging to that set.

An object set has a *Name* property used for identification purposes, although the name is not necessarily unique among all object set names. In the original OSM definition, object sets were permitted to have multiple names, but for simplicity OSMX limits them to at most one name per object set. (This is not a significant limitation because multiple aliases can be encoded into a single name string if desired.) To enable

unambiguous identification of an object set, OSMX introduces an *ID* property that uniquely identifies an object set within an ontology. An object set has exactly one ID.

Because an object set's name is often used to label the concept associated with the object set (such as *Car*), one can use it in data-extraction processes to help match the ontological concept with terms found in the source documents. However, this technique is complicated by the potential use of synonymy, abbreviation, or inaccuracy in the choice of the name. The data-extraction framework reference implementation we describe in this thesis does not employ object set name matching techniques due to their complexity, but nothing in the framework rules out use of this approach in other implementations.

3.1.2 Lexicality

Common to both OSML and OSMX is the property of object set *lexicality*. The ontology designer designates an object set as lexical or nonlexical. In our reference implementation, we assume that nonlexical objects do not directly represent textual values, but rather represent the things in the real world that the lexical objects describe. Thus "James Smith" is a lexical object representing a person's name, which serves to describe the actual individual represented by the nonlexical object. Since under this assumption we cannot express the nonlexical object in a meaningful textual manner, we instead express the object with an arbitrary identifier such as "Person549."

3.1.3 Primary object set

OSM supports the designation of one object set in the ontology as the top-level primary object set of interest. This creates a constraint on the object set with respect to a particular record (in the case of a multiple-record document) or an entire document (in

the case of a single-record document). Essentially, we constrain the primary object set to produce exactly one object from the input record or document. Thus, while the global view of the ontology (across all records or documents) might specify an unbounded set cardinality for the object set, in the context of a single record or document its cardinality is one.

3.2 Data frames

In general, a *data frame* describes the relevant knowledge about a data item, such as an object set [Emb80]. For extraction purposes, a data frame defines value recognition rules for the concept associated with the corresponding object set. If the object set is lexical, the corresponding data frame will describe the set of values that may identify objects in the set. For both lexical and nonlexical object sets, a data frame can specify keywords and other contextual clues that help indicate the existence of an object within the text.

In OSMX, a data frame can specify an internal representation for a lexical object's value. This allows us to interpret the value as a particular data type, such as `String` or `Double`. We may also designate a *canonicalization method* that converts extracted values into a canonical format compatible with the internal representation.

This functionality, though, is largely peripheral to the data-extraction process. Much more important are the extraction rules themselves; these are specified by constructions of regular expressions, macro and lexicon references, and value and keyword phrases. We discuss each of these elements below.

3.2.1 Matching expressions, macros, and lexicons

The most basic element of an extraction rule for a data frame is a matching expression. This is an augmented form of a Perl-5 compatible regular expression. The level of regular expression support is defined by the Java regular expression package `java.util.regex`. We augment regular expressions by allowing the rule designer to embed macro and lexicon references within the expression itself.

A macro defines a literal substitution rule similar to the `#define` macro-substitution feature of the C programming language. For example, we might define a macro named "DayOfWeek" with the substitution value of:

```
((Sun|Mon|Tues|Wednes|Thurs|Fri|Satur)(day)?)
```

We can then construct a regular expression that references this macro, such as:

```
((from|on|starting|beginning) {DayOfWeek})
```

When this expression is applied to a text, the macro reference first expands into the substitution value, and the resulting regular expression is matched against the text.

Macro references are fully recursive in our reference implementation, but cyclical references are forced to terminate at the first recurrence of a previously expanded macro, so that infinite recursion does not occur.

A somewhat similar process occurs with lexicon references. A lexicon is an external list of text strings that allow matching against finite domains of specific values rather than general patterns. For example, a lexicon might contain entries for the U.S. states and their standard abbreviations. When we encounter a lexicon reference within an expression, we substitute the disjunction of the lexicon entries. For example:

```
({CityPattern}, {State} {Zip})
```

could be an expression used to identify the last part of a U.S. postal address. If the `{State}` reference were for a lexicon, it might expand into the following:

```
(alabama|alaska|arizona|...|al|ak|az|...)
```

With the tools of regular expressions and macro and lexicon substitutions, we can create powerful text matching rules and combine them in different ways to extract values and contextual keywords. We next discuss the use of matching expressions in extracting lexical values.

3.2.2 Value phrases

OSM allows an ontology designer to associate zero to many *value phrases* with a data frame for a lexical object set. A value phrase is an extraction rule that uses potentially several matching expressions to locate the correct strings from the input text.

The most basic requirement of a value phrase is that it contain a matching expression for the value to be extracted. It may also contain the following expressions:

- **Required context**: the value must be found entirely within the string matched by this expression.
- **Left context**: the string matched by this expression must be immediately adjacent to the left of the extracted value.
- **Right context**: the string matched by this expression must be immediately adjacent to the right of the extracted value.
- **Exception**: the extracted value must not contain the pattern specified by this expression.

- **Substitute from/to**: the "substitute to" string replaces substrings of the extracted value that match the "substitute from" expression. This provides a limited alternative to canonicalization.

A value phrase can also specify a *confidence value* that gives weight to the matches for the phrase. This can be useful when a data frame contains several value phrases and some are more likely than others to yield accurate matches due to their higher specificity.

The ontology designer can assign a label (or *hint*) to a value phrase. This can help the designer remember the purpose or idea behind the expression without having to decode the expression itself. The hint also plays a role in data frame inheritance, which we discuss in more detail in Section 3.4.

3.2.3 Keyword phrases

A *keyword phrase* is a recognition rule for contextual indicators (*keywords*) that help provide evidence for the existence of lexical or nonlexical objects. Keyword phrases lack the contextual, exception, and substitution expression features of a value phrase. Instead, they simply contain one expression that targets matching keywords in the text. Keyword phrases can, however, define confidence values for matches, and they can also have hints, just as value phrases can.

We may define keyword phrases for the entire data frame, or for individual value phrases. Associating keyword phrases with specific value phrases enables us to give preference to value phrase matches that have a nearby associated keyword phrase match, over value phrase matches that lack an associated keyword. Keyword phrase definitions at the data frame level similarly lend more credence to the value matches for that data

frame in comparison to other data frames' value phrase matches where no keyword matches are found.

3.3 Relationship sets

A *relationship set* defines a relation between object sets in an ontology. Its members are *relationships*, tuples whose elements are objects that are members of the related object sets.

Relationship sets define how concepts are linked together. They are therefore a central part of a data-extraction ontology: for example, we not only wish to know that "July 24, 1998" is a date, but that it links to a particular library book ("Moby Dick") as its overdue date. This example would involve the Library Book object set connecting to the Overdue Date object set via a relationship set. The relationship set would have a member relationship that connects "Moby Dick" to "July 24, 1998."

Relationship sets must have a name; in the example above, an appropriate name might be "Library Book is overdue on Overdue Date." We can shorten this to "is overdue on" if we provide a reading arrow with the relationship set to indicate which object set "is overdue on" the other. Many binary relationship sets in data-extraction ontologies express an "X has Y" relation between object sets, and so for convenience we assign "has" as the default binary relationship set name if none is specified by the user. The name is read as "<nonlexical> has <lexical>" since that is typically correct; if this assumption is wrong, the ontology designer may override the name or reading direction to compensate.

We also assign an ID to each relationship set because, like object sets, the relationship set's name is not guaranteed to be unique within the ontology.

The method of linking object sets to each other via a relationship set is a matter of some complexity. We address this issue below.

3.3.1 Connections

A relationship set defines a relation and consequently a structure for tuples that belong to the relation. Each slot in the tuple structure corresponds to a connection between the relationship set and an object set. We permit an object set to connect to the same relationship set in multiple ways, so for example we might have a tuple structure like <*Person, Person*>. We distinguish each slot in the tuple structure from the others (e.g., <$Person_{Child}$, $Person_{Parent}$>, so that <Person1, Person2> is regarded as a different tuple than <Person2, Person1>, provided the order of the slots is held constant.

In OSMX, we define each slot as a *relationship set connection*. Because identifying an object set does not uniquely identify a single connection for a relationship set, we furnish an ID property for relationship set connections that guarantees uniqueness, allowing us to distinguish between different connections to the same object set by a single relationship set.

The number of connections belonging to a relationship set defines the set's *arity*. In practice, ontology designers use binary relationships extensively, and *n*-ary relationships ($n > 2$) are comparatively rare.

3.3.2 Participation constraints

The ontology designer may affix a *participation constraint* to a relationship set connection. The most basic type of participation constraint prescribes the minimum and maximum number of times that an object may appear in that slot in the set of tuples that belong to the relationship set. We represent the concept of unbounded maximum

participation by the asterisk (`*`) symbol. Our format for writing basic participation constraints is `min:max`, with one exception: when the minimum and maximum participation constraints are the same value, we allow (but do not require) the user to write the constraint as a single value, e.g. "1". In this thesis, we term participation constraints with maximum constraints of 1 as *max-one* constraints, and those with unbounded maximum constraints we term *max-many* constraints.

Let us suppose, for illustration purposes, that a relationship set connection defines a participation constraint of `0:1` for its associated lexical object set, *SerialNumber*. This means that at most one relationship in the set can contain "0X947JP" in the *SerialNumber* slot corresponding to the relationship set connection. However, because the minimum participation constraint is zero, we satisfy the constraint even if no relationships have the "0X947JP" object in the *SerialNumber* slot.

We can add more sophistication to participation constraints. One way is to allow for an "expected" or "average" participation value. This would indicate how many times we expect to find a given object appearing in the slot for the relationship tuples. For example, a participation constraint of `0:1.2:*` would indicate that while an object may appear zero or more times, on average it appears in 1.2 tuples in the relationship set. An expected constraint can provide good information for a `ValueMapper` to make its decisions, such as whether to infer a particular relationship. Although OSMX supports the use of expected participation values, it does not require it.

Another extension to participation constraints is constraint refinement. This is expressed with an arrow notation after the main participation constraint: `0:*->0:1` is a typical example. The `0:1` indicates that the constrained object's participation is limited

within the set of relationships connected to a primary object of interest. Refined participation constraints are useful for relationship sets with an arity of 3 or more.

3.4 Generalization and specialization

OSM permits us to define generalization-specialization relationships between object sets. These express the "is-a" notion: a *Cat* is a *Mammal*. In this example, *Mammal* is the generalization and *Cat* is the specialization. In terms of set theory, the specialization set of objects is a subset of the generalization set of objects.

An object set may have multiple generalizations, multiple specializations, or both. OSMX prohibits circular generalization-specialization, since object sets involved in such a structure are by definition identical sets, and can thus be modeled as a single object set.

Specializations inherit the data frame definitions of their generalizations. They can also add their own value and keyword phrases. Furthermore, we allow specializations to override named value and keyword phrases inherited from ancestor object sets. This is done simply by creating a new phrase with the same hint as the phrase to be overridden, and providing a new phrase definition. We can also suppress an inherited phrase by overriding it with an empty definition. We do not, however, support mutation of an inherited phrase into a new definition based on the old; all inherited phrases are either overridden entirely or retained as-is.

While the old Ontos system had some support for data frame inheritance, it was limited to the immediate parent of the specialized object set. Our new reference implementation provides full inheritance from all ancestors of an object set.

3.5 Data instances

OSMX provides support for a *data instance*, a collection of objects and relationships that fit the ontology. We use data instances to store the results of data extraction alongside the ontology structure. This enables us to refer back to those objects and relationships during subsequent extraction activities and use those values and associations to further aid the extraction and mapping process.

OSMX defines not only relationships but also relationship connections, which link relationships to objects in much the same way as relationship set connections link relationship sets to objects sets.

OSMX also supports associating confidence values with objects and relationships. Algorithms might use these values to establish threshold criteria for acceptance of objects and relationships into a data instance. How these values are calculated is left up to implementations of the framework to define.

4 OntosEngine: an OSMX-based implementation

This chapter discusses the design and implementation of a working data-extraction system that adheres to the architecture defined by the data-extraction framework. We mean for this new system to serve as a point of reference for future implementations or enhancements, so we refer to the system as the *reference implementation* of the framework. This term should not be confused with the term *framework definition*, which is the set of classes, interfaces, and other architectural components that establish the parameters of the framework itself, without respect to any particular implementation.

We do not intend this chapter as a comprehensive explanation of how the reference implementation is designed, but we do dwell in detail on some of the more interesting and complex parts of the system. We pay particular attention to the implementation of `ValueMapper`, as this is where the central knowledge inference occurs. First, however, we consider the workings of the system's other modules.

The implementation we describe below is written in Java, compatible with version 1.4 of the Java virtual machine.

4.1 General implementation details

We start with the highest level of functionality, the `DataExtractionEngine`. Figure 10 shows a high-level flow diagram of the reference implementation. We extend this abstract class, creating the `OntosEngine` class, in order to implement the `doExtraction()` method. We also implement initialization code that allows command-

line parameters or entries in a configuration file to specify the implementation modules' subclasses at runtime.

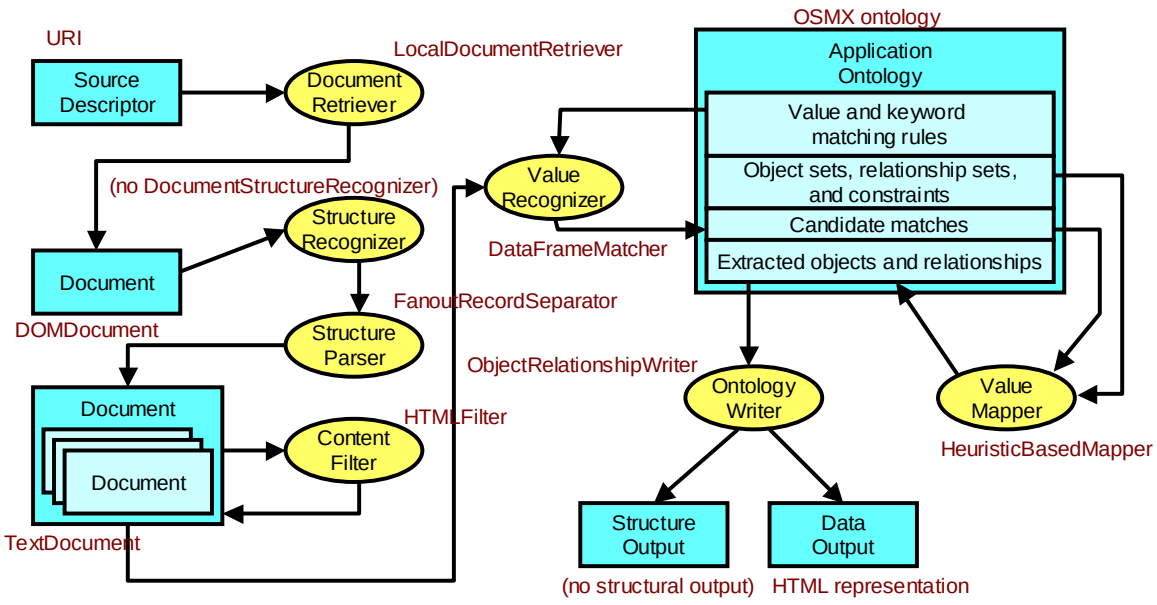

Figure 10: Architecture of the new Ontos system under the framework.

After loading all necessary modules, the `doExtraction()` method instantiates an `OntosExtractionPlan` and invokes its `execute()` method. As its name suggests, `OntosExtractionPlan` extends the `ExtractionPlan` abstract class. For our implementation of `ExtractionPlan`, we define a straightforward algorithm for performing the extraction. Iterating over each `Document` returned by the `DocumentRetriever` module, we obtain a document tree from the available `DocumentStructureParser` or retain the original `Document` if no parser module was specified in the initialization phase. For each `Document` in the document tree, we remove markup with the available `ContentFilter` and then perform value recognition and

mapping with the appropriate components. Finally, we write the results to a human-readable HTML file.

We now examine each of these general steps in a bit more detail.

4.1.1 Retrieval, parsing, and filtering of documents

`LocalDocumentRetriever` is a `DocumentRetriever` that locates and retrieves documents from a specified directory in the local filesystem. From the directory, `LocalDocumentRetriever` can retrieve files whose names match a regular expression specified as an initialization parameter to `OntosEngine`. Our reference implementation does not attempt any retrieval from online sources such as Web search engines. Figure 11 shows part of a Web page that was saved locally and passed to the engine by the `LocalDocumentRetriever`.

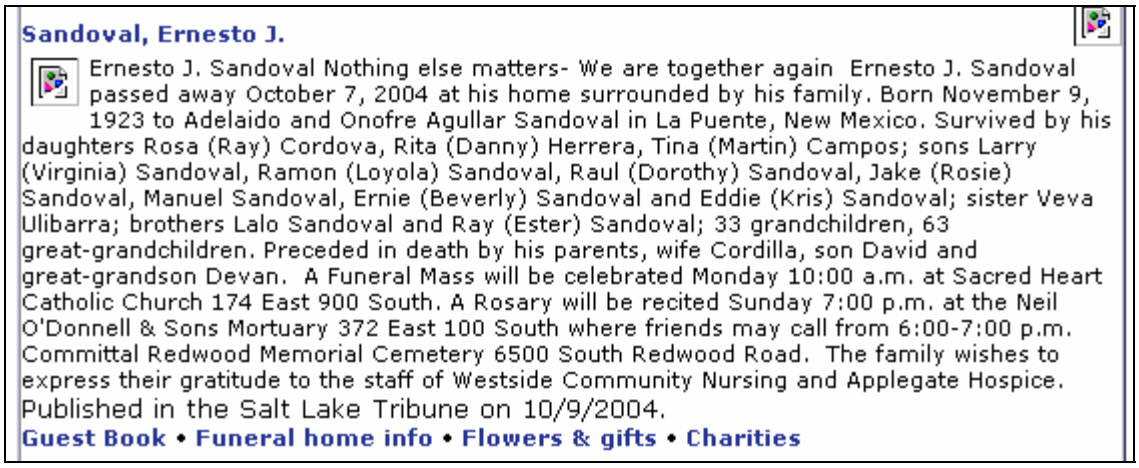

Figure 11: Sample source obituary (names have been changed for privacy).

Because our reference implementation mimics legacy Ontos in its focus on extraction from single- and multiple-record documents, we only depend on one `DocumentStructureParser` implementation, so no implementation of `DocumentStructureRecognizer` is necessary. Indeed, even an implementation created

merely for reference purposes would be too trivial to be instructive. In any case, the `DocumentStructureRecognizer` interface is small enough (defining only two methods) and well enough documented that a competent programmer will have no problem implementing the interface despite the lack of a reference implementation.

A fellow research group member furnished our `DocumentStructureParser` reference implementation, `FanoutRecordSeparator`. True to its name, the parser anticipates that the document has a shallow multi-record structure, and divides the document into sub-documents accordingly. It is possible for the `FanoutRecordSeparator` to return a single-node document tree, in which case the engine treats the tree as a single-record document rather than a multi-record document with no records [WE04].

Our present focus for extraction is on HTML documents, which contain a considerable amount of markup extraneous to the data-extraction process. We therefore provide an `HTMLFilter` implementation of `ContentFilter`. It essentially serves as an **`Adapter`** [GHJV95] for a preexisting DEG utility that strips out unwanted HTML. This exemplifies how we can incorporate existing code into the framework as well as writing original implementations.

In total, these modules (`LocalDocumentRetriever`, `FanoutRecordSeparator`, and `HTMLFilter`) prepare a document for value recognition and mapping to the ontology.

4.1.2 Recognizing values

At this point in processing, the framework requires an ontology. We use the ontology showing in Figure 12, representing the domain of obituaries. (Arrows indicating how the relationship set names should be read are omitted to simplify the diagram; the direction is outward from the primary object set.)

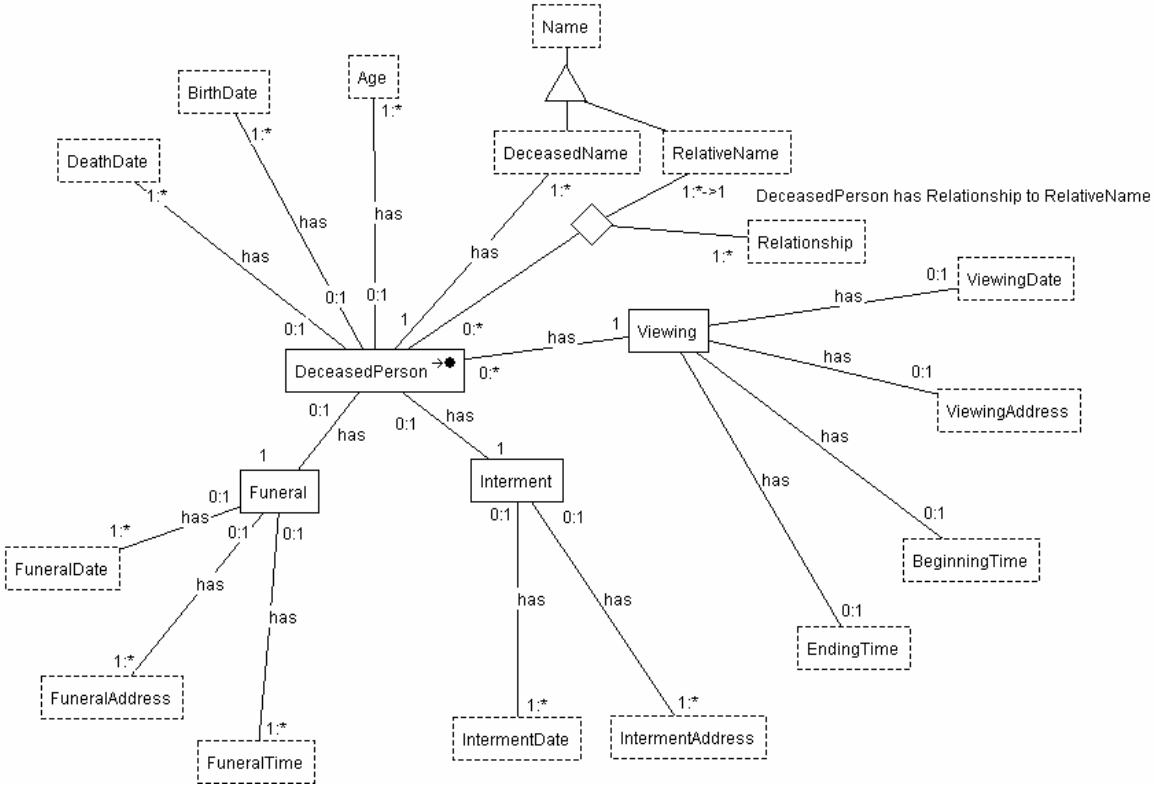

Figure 12: Obituaries ontology.

Our reference implementation of `ValueRecognizer` is called `DataFrameMatcher`, reflecting our choice of OSMX (with its data frames) as the ontology language. This module locates the recognition rules specified by each data frame and applies them to the input text. The matcher identifies all substrings in the text that match the recognition rules and, for each such substring, constructs a `MatchedText` object that records the matched substring, its starting and ending character positions (in the context of the

filtered document), and the URI of the `Document` from which it was extracted. A `MatchedText` object also maintains a status attribute, indicating whether the match has been accepted, rejected, or not yet processed by the `ValueMapper`. These `MatchedText` objects are stored with the rule that they matched, preserving information vital to the operation of the `ValueMapper`. Figure 13 depicts the functionality of `DataFrameMatcher`.

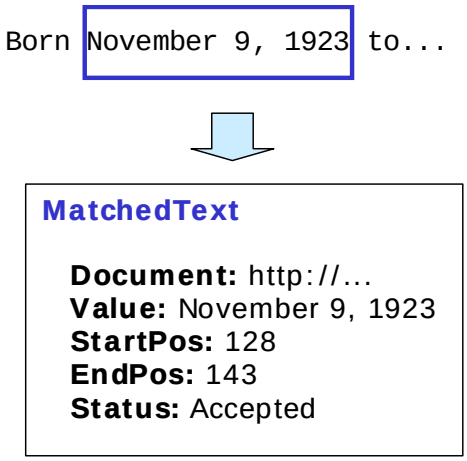

Figure 13: Operation of `DataFrameMatcher`.

4.1.3 Producing output of data instance

We provide an `OntologyWriter` subclass called `ObjectRelationshipWriter` that produces a human-readable hierarchical list of objects and relationships in each data instance stored with the input ontology. The output format is HTML, which suits our present purposes since we intend humans and not computers to process the results in our reference implementation. Figure 14 is an example of the output our implementation generates. Despite its orientation towards human readability, the logic in

`ObjectRelationshipWriter` can easily be adapted to generate machine-friendly XML output.

Note that the legacy Ontos system produces its output in the form of SQL statements: DDL (CREATE TABLE) for generating the model structure, and DML (INSERT) for generating the data instance. Although this is not essential functionality for our reference implementation, the flexibility of our framework makes it a straightforward matter to create an `OntologyWriter` implementation built to emit the model structure and data in SQL.

DeceasedPerson *osmx3113*
- has **DeceasedName** *Sandoval, Ernesto J.*
- has **DeathDate** *October 7, 2004*
- has **BirthDate** *November 9, 1923*
- has **Age** *63*
- DeceasedPerson has Relationship to RelativeName
  - **RelativeName** *Agullar Sandoval*
  - **Relationship** *daughter*
- DeceasedPerson has Relationship to RelativeName
  - **RelativeName** *Lalo Sandoval*
  - **Relationship** *brother*
- DeceasedPerson has Relationship to RelativeName
  - **RelativeName** *Rosa Cordova*
  - **Relationship** *daughter*
- DeceasedPerson has Relationship to RelativeName
  - **RelativeName** *Ernesto J. Sandoval*
  - **Relationship** *daughter*
- DeceasedPerson has Relationship to RelativeName
  - **RelativeName** *Larry Sandoval*
  - **Relationship** *son*
- has **Funeral** *osmx3131*
  - has **FuneralTime** *7:00*
  - has **FuneralAddress** *174 East 900 South*
  - has **FuneralDate** *10/9/2004*
- has **Interment** *osmx3147*
  - has **IntermentAddress** *La Puente, New Mexico*
- has **Viewing** *osmx3261*
  - has **BeginningTime** *6:00*
  - has **EndingTime** *10:00*
  - has **ViewingAddress** *372 East 100 South*

Figure 14: Sample output of `OntologyWriter`.

4.2 ValueMapper implementation

This section discusses our implementation of `ValueMapper`, which is the most complex part of the extraction system (and a primary reason for creating this new framework, due to the difficulty of extending the legacy version of Ontos). The module's task is to infer a set of mappings between extracted values (the `MatchedText` objects) and object sets in the ontology. Each mapping is realized as a lexical object. `ValueMapper` must also infer the existence of nonlexical objects and relationships between objects.

Not every `MatchedText` object will necessarily become a lexical object; nor will the `ValueMapper` implementation infer an object or relationship wherever an object set or relationship set exists. In fact, a key problem for the `ValueMapper` to solve is how to decide when to generate an object or relationship. Our implementation employs various heuristics to solve this problem.

4.2.1 HeuristicBasedMapper

Because a set of heuristics lies at the core of the `ValueMapper` implementation, we name the module `HeuristicBasedMapper`. This module delegates much of the mapping work to a set of auxiliary classes that implement the heuristics, but it retains some processing logic. Most notably, before it applies any of the other heuristics, it first invokes one fundamental heuristic: elimination of subsumed and overlapping matches.

The algorithm at the heart of this heuristic considers subsumption or overlap to occur only if the matches involved belong to the same data frame, since within the data frame itself we have all the information we need to make a determination of whether to retain a conflicting match. We do not consider subsumption or overlap to occur across data frames, since these matches represent equally plausible claims on the same substring of the text, and we do not yet have enough information about the object sets owning the data frames to decide which match should prevail.

When the algorithm determines that matches overlap, it is a simple matter to decide which match to prefer: accept the longer match. When the matches are of the same length, we prefer the earlier occurring match. For example:

- “embankment” is preferred over the subsumed “bank” since it is longer;

- “green house” is shorter than “house plants” (overlapping on the substring “house”), so we prefer the latter;
- “green house” occurs earlier than “house plant” (which is the same length), so we prefer the former.

This introduces an inference bias toward longer matches, but we believe that in most data-extraction applications this bias is justified since it preserves more information. Besides, our framework provides a means to override this functionality if one should wish to test a different bias.

After the application of this preliminary heuristic, `HeuristicBasedMapper` invokes a mapping algorithm centered in the `ContextualHeuristic` class, which extends the abstract class `MappingHeuristic`.

4.2.2 MappingHeuristic

`MappingHeuristic` is a base class for the various heuristic classes involved in our mapping algorithm. Methods `addObject()` and `addRelationship()` handle the association of objects and relationships, respectively, with the appropriate source document reference within the current data instance. Another method, `bindObject()`, creates an object’s connection to a relationship based on the corresponding relationship set connection. Finally, `setOntologyIndex()` and `getOntologyIndex()` manage references to a cached and indexed view of the ontology to maintain reasonable performance in terms of both memory footprint and processing speed.

4.2.3 ContextualHeuristic

`ContextualHeuristic` embodies the bulk of the logic for the mapping algorithm. Its primary method is `processContextualGroup()`. The method operates on an object set and a *matching context*, which is a contiguous region of text from the document represented by a set of candidate value and keyword matches associated with the data frames of the ontology (see Figure 15). (This is similar to the data record table concept

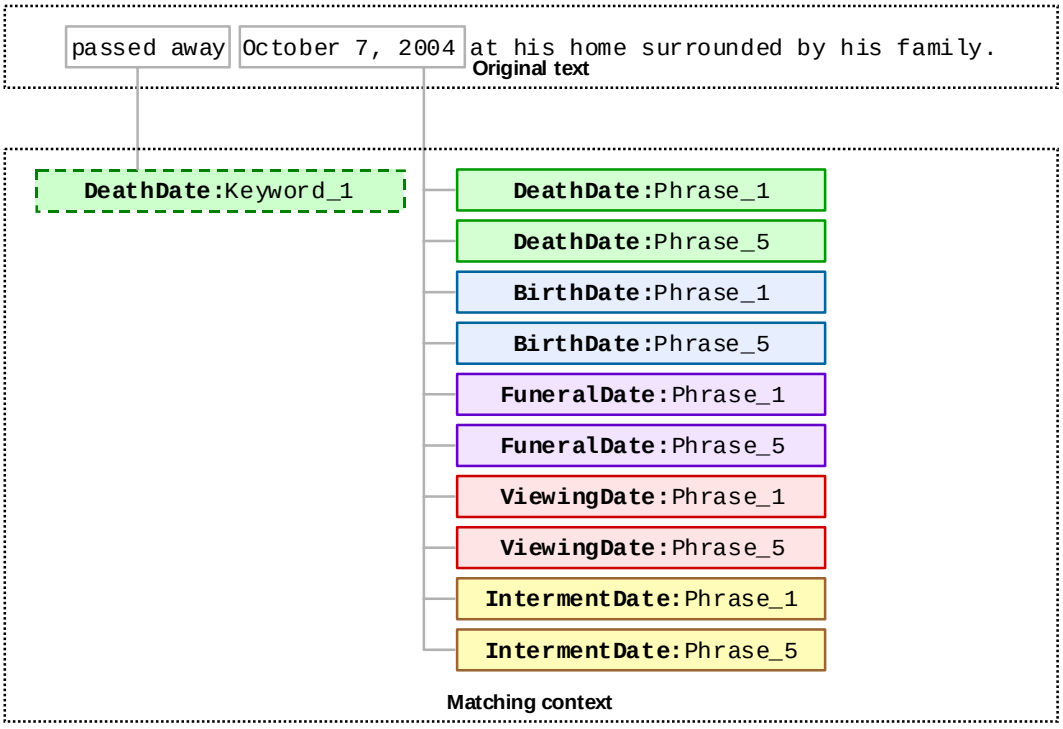

Figure 15: Visualization of a matching context.

of legacy Ontos.) The overall contract of the method is to generate an object for the object set together with its child relationships and objects, based on the structure defined by the ontology and using the evidence provided by the matching context.

When invoked with the top-level primary object set and a matching context spanning the entire document or record, the result of `processContextualGroup()` is a

primary object with all of its associated properties, as represented by subordinate objects connected directly or indirectly (through other connected objects) to the primary object via relationships. If constraints defined in the ontology are not satisfied, however, the method discards the generated object and returns a null value.

Figure 16 shows a representation of the object tree that results from invoking `processContextualGroup()`. The inputs to the method are the *DeceasedPerson* object set for an obituaries ontology, and a matching context containing keyword and value phrase matches recognized by the data frames of the ontology.

DeceasedPerson *osmx3113*
- has **DeceasedName** *Sandoval, Ernesto J.*
- has **DeathDate** *October 7, 2004*
- has **BirthDate** *November 9, 1923*
- has **Age** *63*
- DeceasedPerson has Relationship to RelativeName
  - **RelativeName** *Agullar Sandoval*
  - **Relationship** *daughter*
- DeceasedPerson has Relationship to RelativeName
  - **RelativeName** *Lalo Sandoval*
  - **Relationship** *brother*

Figure 16: Result of `processContextualGroup()` for *DeceasedPerson* object set.

A key insight here is that in each invocation of this method, we focus on a particular object set that, for the current context, acts as a primary object set at the local level—even if it is not designated as the top-level primary object set for the entire document. This paradigm enables us to traverse the ontology recursively, starting with

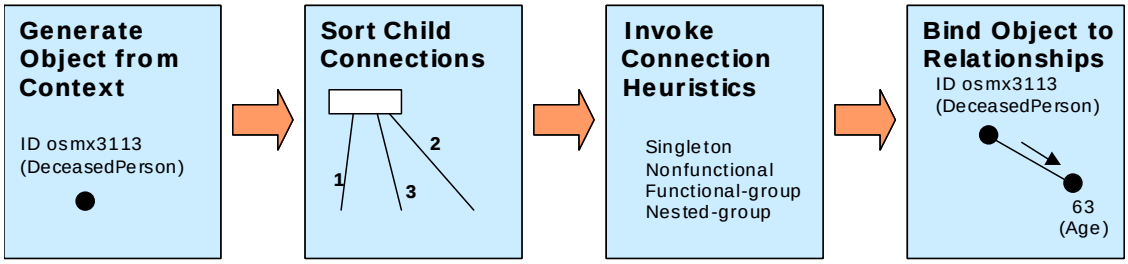

Figure 17: Steps of `processContextualGroup()`.

the top-level primary object set and working outward, invoking `processContextualGroup()` each time we seek to generate an object for another object set. (To avoid infinite recursion, no relationship set may be traversed more than once.) This recursion also allows our algorithm to traverse an arbitrarily complex ontology with a relatively small amount (a few hundred lines) of implementation code. Figure 17 shows the essential steps of the algorithm.

The general flow of the method is:

1. Generate an object for the object set (but do not add it to the data instance yet).
2. If object set is lexical:
    a. Find the best value phrase match for this object set within the current matching context.
    b. If a value phrase is found in the matching context:
        i. Store information about the match (such as the matched text and location) with the object.
        ii. Add the object to the data instance and return it.
    c. Otherwise, discard the generated object and return null.
3. Determine which object sets to process next.
    a. Find all relationship set connections for this object set, except those already visited for this object (prevents infinite cycle).
    b. Determine an order for processing the connections.
4. Process the child object sets.
    a. Invoke the appropriate heuristic (singleton, functional group, nested group, nonfunctional), based on the nature of the relationship set and the participation constraint on the connection, to obtain one or more relationships.
    b. If relationships were generated, bind the object to each relationship.
5. If object is nonlexical:
    a. If not bound to any child relationships, discard it.
    b. Otherwise add it to the data instance.
6. Return the generated object if it was not discarded.

We emphasize three important points of this algorithm. First, to determine the best value phrase match in Step 2a, we define a method called `findBestLexicalMatch()`. This method is designed to be easily overridden if one wishes to redefine what "best match" means. Our reference implementation defines the best match to be the one closest to a keyword if keywords are present; otherwise, it is the earliest occurring match. Second, we define a method for Step 3b called `compareRelSets()` that provides an ordering for the relationship set connections associated with the local primary object set to guide ontology traversal. We implement it

as a protected method in order to provide an easy way to override the ordering, should the existing order prove to be sub-optimal. Third, we note that Step 4a is the recursive step; the four heuristics based on the relationship set connection all invoke the `processContextualGroup()` method of `ContextualHeuristic` at some point in their code.

We now examine the heuristics based on the connecting relationship set's structure and participation constraints. `ContextualHeuristic` invokes these to obtain relationships to bind to the generated object. Figure 18 depicts the different types of relationship set connection structures that correspond to these heuristics.

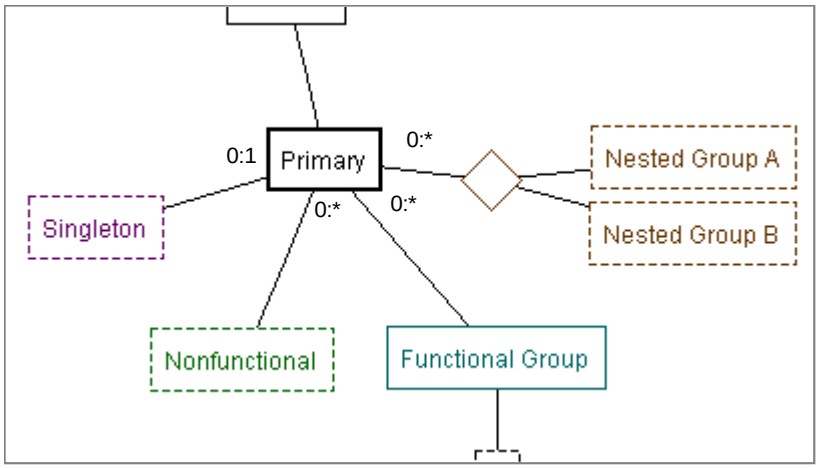

Figure 18: Relationship set connection types: singleton, nonfunctional, functional-group, nested-group.

SingletonHeuristic

For binary relationship set connections where the local primary object set participation constraint has a maximum of one, `ContextualHeuristic` invokes `SingletonHeuristic`. In this case, we expect to infer at most a single relationship for the primary object within the present matching context. (As a notational aside, we

observe that the terms *singleton*, *nonfunctional*, *functional-group*, and *nested-group* strictly describe only relationship set connections within the context of a local primary object set. However, we take the liberty of applying these terms to the connected relationship set and relationships belonging to it, recognizing that this use is meaningful only in terms of the local primary object set.) In relational terms, a singleton relationship usually represents a simple attribute of the primary object, such as *DateOfBirth* for a particular *Person*. Singletons may, however, include complex objects that themselves have child objects.

`SingletonHeuristic` is perhaps the simplest of the connection-based heuristics. Since we only expect to generate at most one binary relationship for the connection, we simply base our decision to generate a relationship on whether the object set at the opposite end of the relationship set generates an object from the matching context. By recursively invoking the `ContextualHeuristic` on the opposite object set using the same matching context, we obtain an object for the best candidate match if one is to be found, and either generate the relationship or exit without generating it depending on the result of the call. Figure 19 shows the operation of `SingletonHeuristic` on a matching context with several candidate matches. The candidate matches (depicted in the top part of the diagram) become accepted (circled value at bottom) or rejected (marked by an "X" in the diagram) for the singleton mapping. At most one object is chosen from the available candidates within the relevant context.

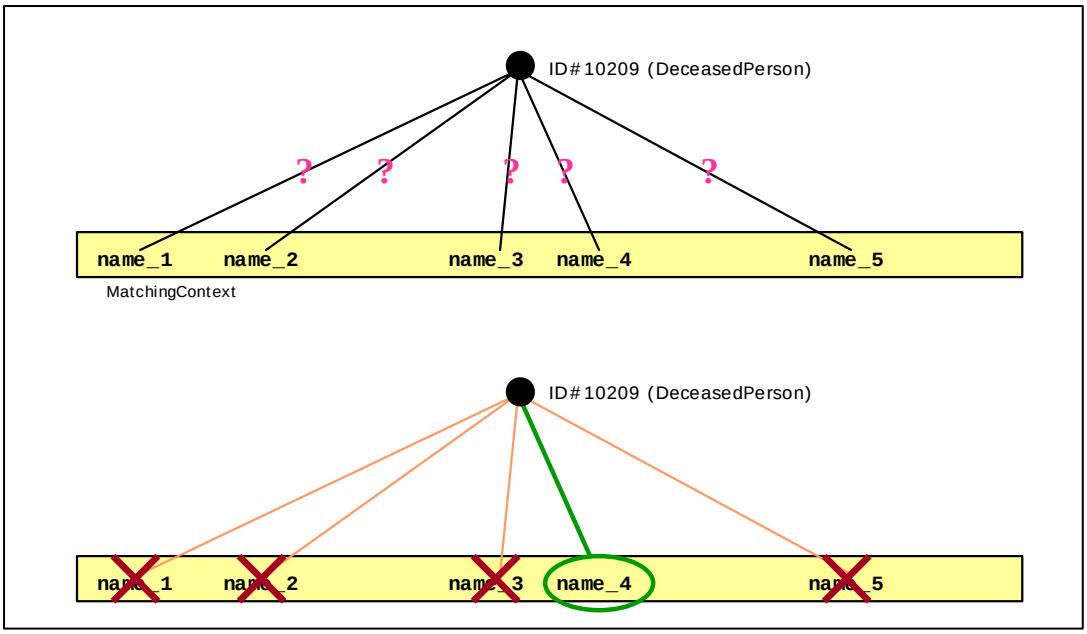

Figure 19: Operation of `SingletonHeuristic`.

NonfunctionalHeuristic

`NonfunctionalHeuristic` infers relationships for connections that qualify as *nonfunctional*. A nonfunctional relationship set connection links the local primary object set to a binary relationship set with a max-many participation constraint. Furthermore, the opposite object set must have no other connections to the ontology than the nonfunctional relationship set, and therefore must be lexical (otherwise it would not generate an object from the text, having no evidence to justify generating one). In relational-model terms, think of a nonfunctional relationship set as defining a simple multi-valued attribute for the local primary object set.

Because of the max-many participation constraint, we can generate multiple relationships for the local primary object, even within a single matching context. Because we are limited to binary relationships, the max-many constraint implies that multiple objects can be generated from the opposite object set. Thus our

`NonfunctionalHeuristic` code repeatedly invokes `ContextualHeuristic` on the opposite object set until it can generate no more objects from the matching context. The code then creates a relationship for each generated object, binds the opposite object to its corresponding relationship, and returns the set of relationships. Figure 20 shows how `NonfunctionalHeuristic` operates. The top portion of the diagram shows the state of the system before the heuristic is invoked; the job is to map the candidate values (indicated by question marks) to the local primary object if the values are deemed acceptable to participate in such a mapping. This determination is made by the recursive call to `ContextualHeuristic`.

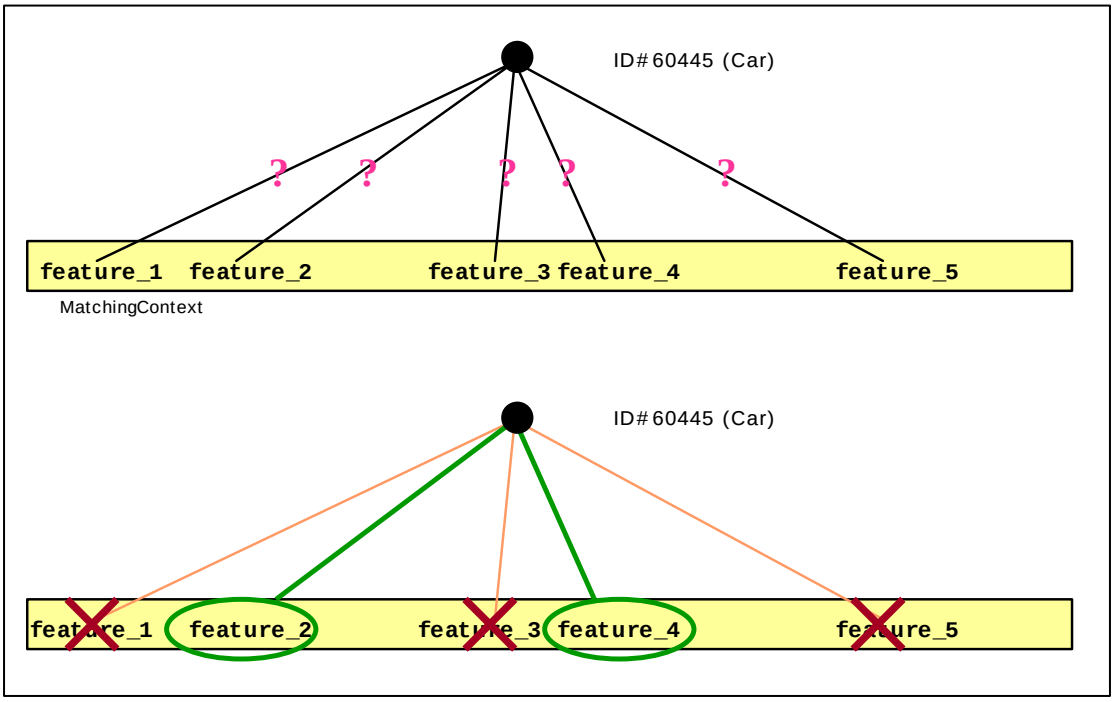

Figure 20: Operation of NonfunctionalHeuristic.

FunctionalGroupHeuristic

The `FunctionalGroupHeuristic` operates on binary relationship sets that strike a balance between the singleton and nonfunctional structures. Like nonfunctionals, it features a max-many participation constraint on the local primary object set side; like singleton, it places no restriction on the connectivity of the opposite relationship set. The algorithm is similar to that of `NonfunctionalHeuristic`, since the max-many constraint governs its operation, so we will not revisit that logic here. We observe, however, that functional groups are complex objects that themselves contain child objects, and therefore the extracted objects are nonlexical. Since this structural difference can influence an extraction strategy, we distinguish this heuristic from `NonfunctionalHeuristic`. Figure 21 illustrates `FunctionalGroupHeuristic`. In this diagram, the nonlexical Address objects appear above the lexical objects (Street, City, State, Zip, Country) that denote their presence within the text. Recursive calls to `ContextualHeuristic` determine whether the lexical objects provide enough evidence to support generating a nonlexical object. If sufficient evidence exists, the recursive call generates and returns the nonlexical object, and `FunctionalHeuristic` maps it to the local primary object.

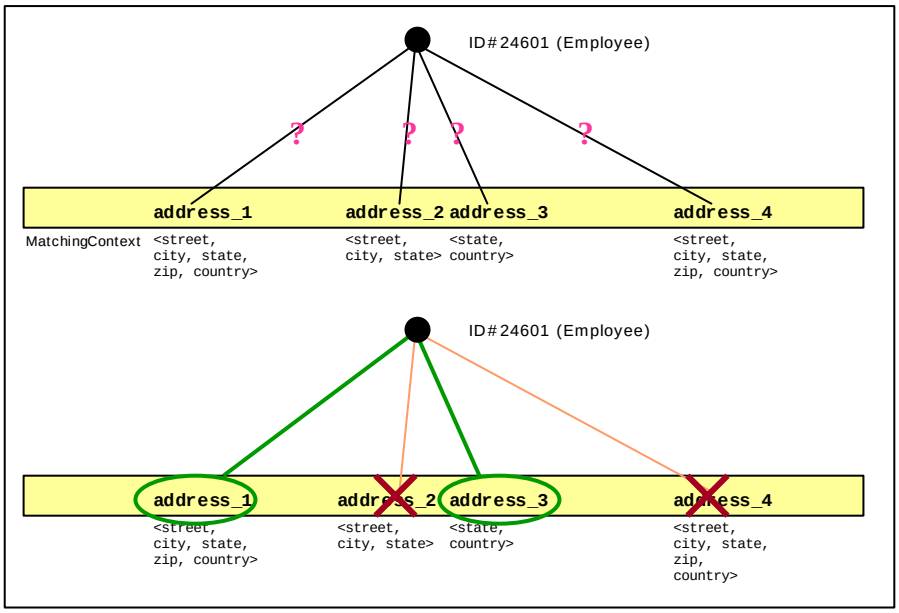

Figure 21: Operation of `FunctionalGroupHeuristic`.

NestedGroupHeuristic

`NestedGroupHeuristic` operates on all relationship sets with an arity greater than two. It includes those for which the primary-side participation constraint is max-one or max-many.

`NestedGroupHeuristic` is the most complicated of the connection-based heuristics. We divide its operation into two main cases: first, when the primary-side constraint is max-one; and second, when it is max-many.

Max-one scenario. The max-one constraint on the local primary object set side allows us to disregard participation cardinalities on the other connections in the relationship set, because we can only discover at most one relationship. In this case, we invoke `ContextualHeuristic` once on each opposite object set for the input context, and generate a relationship if and only if each invocation resulted in a generated object.

Because the order in which the opposite object sets are passed to `processContextualGroup()` may be significant, the algorithm for determining this order is encapsulated in an auxiliary class so it may be overridden if needed.

Max-many scenario. We now consider the more complex case where the local primary object set connection has a max-many participation constraint. An example of such a structure is shown in Figure 22.

The depicted structure shows the top-level primary object set, *DeceasedPerson*, connected to a ternary relationship set. Since the relationship set arity is greater than two, the structure is a nested group.

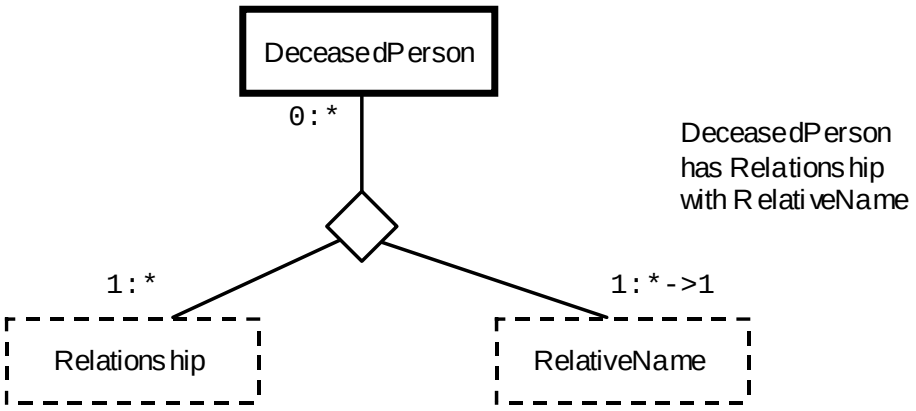

Figure 22: Example of a max-many nested group structure.

The participation constraint on the top-level primary object set connection is 0:*, indicating that a *DeceasedPerson* object may appear in multiple relationship tuples for this nested group relationship set. In other words, a *DeceasedPerson* object may be related to potentially many combinations of *Relationship* and *RelativeName* objects.

The constraint on the *Relationship* connection indicates that an extracted *Relationship* must participate in the relationship set, but may do so many times. Hence, a *DeceasedPerson* may be related to multiple *RelativeNames* under the same *Relationship*.

The constraint on the *RelativeName* connection indicates that at the model level, a *RelativeName* may appear potentially many times in connection with *DeceasedPerson* and *Relationship* objects, but at a local or refined level (i.e., for a particular *DeceasedPerson* object), it may only appear once.

With a max-many constraint on the local primary object set connection, it is possible to create multiple relationships. The important question is, how many do we create? We generally do not desire to create a relationship for every possible *n*-way binding among the extracted objects. Instead, we assume a contextual nesting structure (hence the *nested group* name). In this approach, each object in the relationship is

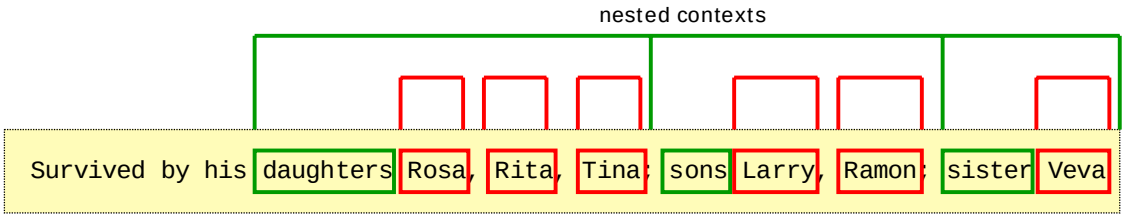

Figure 22: Example of nested contexts.

associated with a region of the text; this region is the object's context. Objects that are permitted to participate in multiple relationships have broader contexts, within which the max-one objects' contexts are nested. See Figure 23 for an example of this idea, based on the sample nested group structure depicted earlier.

The diagram shows part of the context for a *DeceasedPerson* object. Within this context are three nested sub-contexts, one for each *Relationship* object found (daughter, son, sister). Inside each of these contexts are contexts pertaining to the max-one *RelativeName* objects: Rosa, Rita, Tina, Larry, Ramon, and Veva. If we think of each context as a node in a tree, with nested contexts appearing as child nodes, then we have a tree with the local primary object as the root and the max-one objects as leaves. Each distinct path from root to leaf yields a relationship for the nested-group relationship set. Using the example from the above diagram, the nested contexts determine the tree shown in Figure 24. The tree in turn defines relationships that can be generated, which are shown in Figure 25.

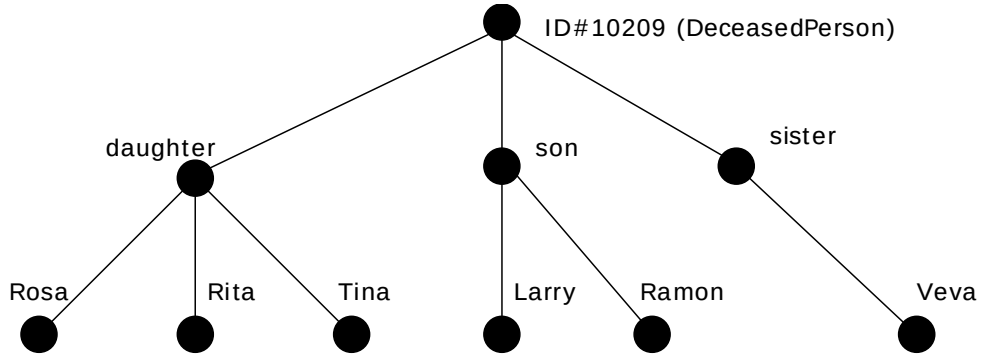

Figure 23: Tree implied by nesting contexts.

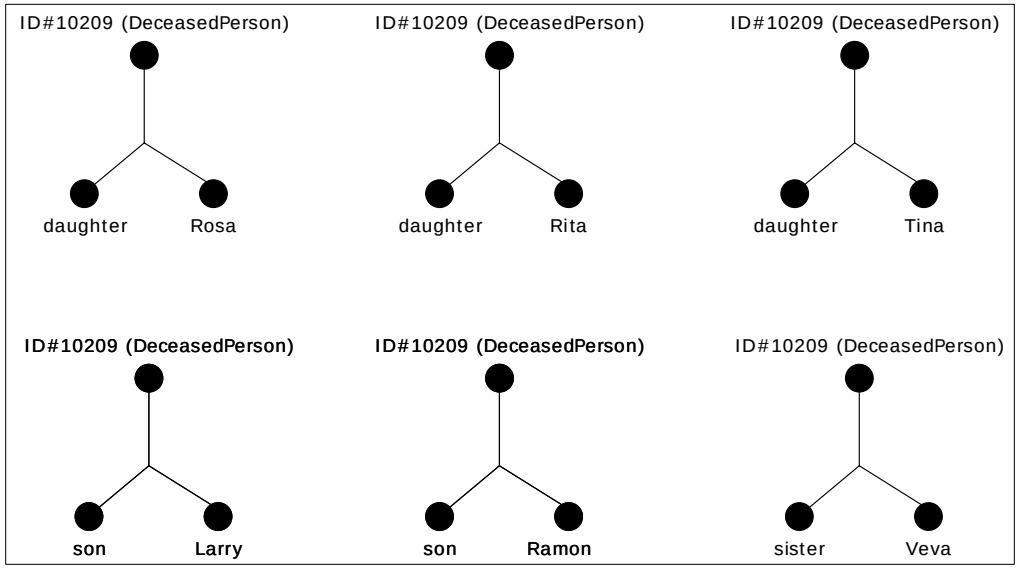

Figure 24: Relationships generated for nested group.

We assume that each object's context begins from the location of the extracted object and continues until the next extracted object of the same object set, or the end of the parent context. To determine which object set defines the next deeper nesting level, we search for the earliest appearing extractable object belonging to those max-many object sets that have not yet defined a nesting level. This is a heuristic for which one can imagine various exceptions (such as if the max-many object tends to appear at the end of a context rather than the beginning), yet it performs well for expressions where earlier concepts factor over a subsequent list of concepts, as in the "sons Larry and Ramon; sister Veva" example.

5 Evaluation of OntosEngine

We have claimed that our framework provides a flexible approach to building a data-extraction system, one that permits a wide variety of strategies and algorithms to be tested and compared while reducing the amount of coding required to achieve this. Although such a claim is not fully testable within the scope of this thesis, as it would involve creating a large number of different implementations to explore the limits of the framework and attempting to measure the increase in programming efficiency gained by use of the framework, we can assess whether the framework is sufficient for supporting data extraction, and leave the question of how flexible it is to future experience. Moreover, past experience with other frameworks gives us a high degree of confidence in our approach. Consequently, we have provided a reference implementation of the framework to demonstrate that it is complete enough to cover the main processes of data extraction. This gives us a point of comparison with legacy Ontos. It is important to show that use of the framework does not negatively impact extraction results, and our reference implementation of the framework does indeed achieve comparable, if not better, results for the extraction process.

We wish to show that the framework allows implementation of a system that is capable of achieving at least the same degree of accuracy in extraction as the legacy Ontos system. We demonstrate this by performing extraction using the same set of documents and the same ontology on both our new implementation and the legacy system, then comparing the results.

The subject domain for this experiment is obituary listings such as those commonly found in local newspapers. Our corpus is a set of recent obituaries from two

different newspapers—the *Salt Lake Tribune* and the *Arizona Daily Star*—containing a total of 25 individual obituaries. The ontology used for both experiments is the same, with the exception that for our implementation we represent the ontology with OSMX, and for the legacy system we represent it with the OSML language. The representational differences between these languages do not influence the accuracy of the extraction. We do, however, make use of keyword affinity, which is solely supported in OSMX and thus affects only the accuracy of the new system. We do this to demonstrate one of the new features of OSMX.

In our experiment, we measure precision and recall with respect to the lexical object sets from the ontology. Our basis for comparison is a set of manually derived extraction results based on the judgment of the human experimenter. We score correct (exact and partial) matches, false positives (incorrect mappings), and false negatives (missed mappings). Exact matches are defined as identical values appearing in the same mapping for both automatic and manual results; for partial matches, the automatically extracted value may be a substring of the manual result, or it could subsume or overlap the manual result. For example, a partial match might contain only "July 21," where the human would extract "July 21, 1993," or vice versa. In contrast, an exact match would contain "July 21, 1993" for both automatic and manual extraction processes. If the values are entirely different, such as "August 23, 1979" and "July 21, 1993," we consider them to be different mappings and record a false positive. We record as a false negative the failure to extract any sort of mapping (exact, partial, or incorrect) to correspond with one extracted manually.

Counting matches for nonlexical object sets entails a bit more difficulty, since there is no inherent lexical representation to use as a basis of comparison between mappings. Thus we simply count the occurrences of nonlexical objects in the test mappings and compare that number to the mappings in the control instance. If numbers are equal, precision and recall for that object set are both 100%. If they are unequal, one of precision or recall will be 100% but not both. It may seem counterintuitive to say that two nonlexical objects match each other even if the associated lexical objects wholly fail to match, but this is the only way to match nonlexical objects to each other without over-counting lexical mismatches. For example, if we compare two Funeral objects, one with a FuneralDate of "6 June 1999" and the other with a FuneralDate of "8 Sept 1999," we count a mismatch on FuneralDate. If we also count a mismatch on Funeral because the objects' lexical children disagree, the FuneralDate mismatch has effectively been counted twice, skewing the results. Therefore, to avoid disproportionately counting mismatches, we allow any nonlexical object to be identified for counting purposes with any other nonlexical object of the same object set. In the example above, we consider the two Funeral objects to match, but count the mismatch at the FuneralDate level.

However, we now have a problem when counting matches for lexical object sets that are functionally dependent on a nonlexical object set. For example, the lexical DeceasedName object set is functionally dependent on the nonlexical DeceasedPerson object set. When counting matches for DeceasedName, do we include those that were extracted for a false-positive DeceasedPerson object? For our experiment, we have chosen to omit functionally dependent mappings from our calculations when a nonlexical object is falsely generated. We do this because we perceive a strong correlation of the

nonlexical-lexical relationship with the rules for implication in propositional logic: the nonlexical mapping corresponds to the antecedent, and the lexical object to the consequent. If the antecedent is false, no statement can be made about the falsity of the consequent; similarly, a false mapping at the nonlexical level prevents us from making a determination as to the quality of the dependent lexical mapping. We therefore exclude the lexical mappings from our tally when they are extracted due to a falsely generated nonlexical object.

Our results appear in Table 1. The left-hand column lists the object sets of the ontology. The following columns list the total correct matches (with partial matches in parentheses), the total false positives, total false negatives, precision, and recall for each of the two newspapers from which the input obituaries came. The rightmost columns give the overall precision and recall numbers. Precision and recall figures count both exact and partial matches as correct matches. Each row of the table gives the results for the legacy Ontos system (topmost numbers, in normal type) and the new Ontos system (bottommost numbers, in boldface).

For half of the object sets, the new system performs as well as or better than the legacy system. Of the remaining nine object sets, the new system performs worse for seven, and for the other two it equals or excels the legacy system's performance in either recall or precision, but not both.

	Salt Lake Tribune 9-Oct-2004					*Arizona Daily Star 9-Oct-2004*					Overall	
Object Set	C(P)	FP	FN	R%	P%	C(P)	FP	FN	R%	P%	R%	P%
DeceasedPerson	15	2	0	100	88	9	0	1	90	100	96	92
	15	**2**	**0**	**100**	**88**	**10**	**0**	**0**	**100**	**100**	**100**	**93**
DeceasedName	8 (7)	0	0	100	100	5 (4)	0	0	100	100	100	100
	8 (7)	**0**	**0**	**100**	**100**	**6 (4)**	**0**	**0**	**100**	**100**	**100**	**100**
Age	5	7	0	100	42	3	6	6	33	33	57	38
	5	**4**	**0**	**100**	**56**	**4**	**5**	**6**	**40**	**44**	**60**	**50**
DeathDate	12	3	3	80	80	8	1	1	89	89	83	83
	11	**4**	**4**	**73**	**73**	**9**	**1**	**1**	**90**	**90**	**80**	**80**
BirthDate	13	2	2	87	87	2	5	1	67	29	83	68
	11	**4**	**4**	**73**	**73**	**2**	**6**	**1**	**67**	**25**	**72**	**57**
Funeral	11	4	0	100	73	8	2	0	100	80	100	76
	11	**4**	**0**	**100**	**73**	**8**	**2**	**0**	**100**	**80**	**100**	**76**
FuneralDate	8	3	3	73	73	4	1	4	50	80	63	75
	7	**4**	**4**	**64**	**64**	**6**	**0**	**2**	**75**	**100**	**68**	**76**
FuneralTime	7	3	4	64	70	4	3	4	50	57	58	65
	5	**5**	**6**	**46**	**50**	**3**	**5**	**5**	**38**	**38**	**42**	**44**
FuneralAddress	5 (2)	4	3	70	64	2	5	2	50	29	64	50
	6 (2)	**3**	**2**	**80**	**73**	**1**	**7**	**3**	**25**	**13**	**64**	**47**
Interment	1	12	0	100	8	0	7	0	100	0	100	5
	1	**10**	**0**	**100**	**9**	**0**	**9**	**0**	**100**	**0**	**100**	**5**
IntermentDate	1	0	0	100	100	-	-	-	-	-	100	100
	1	**0**	**0**	**100**	**100**	-	-	-	-	-	**100**	**100**
IntermentAddress	1	0	0	100	100	-	-	-	-	-	100	100
	1	**0**	**0**	**100**	**100**	-	-	-	-	-	**100**	**100**
Viewing	10	47	0	100	18	4	18	1	80	18	93	18
	7	**5**	**3**	**70**	**58**	**5**	**2**	**0**	**100**	**71**	**80**	**63**
ViewingDate	1	4	0	100	20	0	0	5	0	0	17	25
	0	**6**	**1**	**0**	**0**	**0**	**1**	**5**	**0**	**0**	**0**	**0**
BeginningTime	5	1	5	50	83	4	0	1	80	100	60	90
	4	**2**	**6**	**40**	**67**	**4**	**1**	**1**	**80**	**80**	**53**	**73**
EndingTime	6	0	4	60	100	4	0	1	80	100	67	100
	2	**2**	**8**	**20**	**50**	**3**	**1**	**2**	**60**	**75**	**33**	**63**
ViewingAddress	2	1	2	50	67	0	3	4	0	0	25	33
	0	**4**	**4**	**0**	**0**	**1**	**3**	**3**	**25**	**25**	**14**	**14**
Relationship/	75 (32)	196	28	77	35	58 (35)	94	45	67	50	73	41
RelativeName	**67 (31)**	**191**	**38**	**72**	**34**	**69 (37)**	**78**	**32**	**77**	**58**	**74**	**43**

Table 1: Results of extraction of obituaries from two newspapers. Boldface numbers indicate results from the new Ontos system; normal font indicates legacy Ontos results. Columns are: C(P) = Correct (Partial); FP = False Positive; FN = False Negative; R% = Percent recall; P% = Percent precision.

By examining the operation of the systems, we have identified reasons why for some object sets our new system underperforms in comparison to the legacy system. The results for DeathDate and BirthDate illustrate one such reason. The new system is designed to follow the heuristic of choosing the value match that is closest to a corresponding keyword. This heuristic was enunciated in [ECJ+99], but it turns out that the legacy system does not quite accomplish this, instead selecting the value match

nearest to the earliest found keyword—even if a later keyword-value pair were closer to each other than those found. In some of the obituaries used for our experiment, we find phrasing such as "died at home in Park City, Utah, early on the morning of Thursday, October 7, 2004," while later in the obituary might be a reference to a relative who "died on July 7, 1991." In such a case, the legacy system will serendipitously make a correct decision to choose the earlier keyword-value pair, while the new system in strict adherence to the heuristic will incorrectly choose the later pair. We observe that because this heuristic is modularly implemented in the `ContextualHeuristic.getBestLexicalMatch()` method, it is easy to change the new system to follow the behavior of the legacy system in this regard, if one should wish to favor that behavior over the heuristic specification.

Another reason for the new system's underperformance is evident in the numbers for the Viewing object set and its dependent object sets, ViewingDate, BeginningTime, EndingTime, and ViewingAddress. The legacy system significantly outperforms the new system in recall and precision for all of these object sets, except for the precision of the nonlexical Viewing object set. Essentially, the legacy system is extracting so many Viewing objects that its precision suffers but its recall is high. Further, in counting matches for the dependent object sets, we have excluded those that belong to false-positive Viewing objects. This causes the legacy system to show much better results, since it has more chances to pick up the correct values in connection with a Viewing than the new system does. In contrast, the new system has much better precision with respect to Viewing (63% compared to 18%) than the legacy system does, but suffers from only having a few chances to correctly link the dependent lexical objects with the nonlexical

Viewing objects. This behavior too can be modified, at the expense of the high precision it currently enjoys.

In summary, the two systems perform at roughly the same level, with the major differences stemming from a few intentional departures by the new system from the rules followed by the legacy system. Some of these differences are magnified due to the counting methods employed that exclude from the count lexical objects that are dependent on false-positive nonlexicals. Generally, though, the performance between the two systems is remarkably similar, considering the very different algorithms at the heart of the systems. Where performance degrades in the new system, the code is modular enough to allow poorly performing code to be optimized or replaced without impacting the rest of the system. This bears out our claim that the framework is sufficient to support the task of data extraction at the same level as the legacy system, while providing a much more capable supporting code infrastructure.

6 Future work

Our efforts offer three dimensions of possibilities for future improvement. One dimension is the wide variety of customized modules that can be developed and plugged into the framework. For example, one might write a `DocumentRetriever` that invokes the Google APIs to take advantage of the search engine's existing capabilities for retrieval from the Internet. Another possibility is a `ValueMapper` that uses natural-language processing techniques to aid its decision making processes. Or, we might envision an `ExtractionPlan` that dynamically governs the flow of the extraction process based on the number and type of documents retrieved, the data found, and the degree to which the extracted data satisfies the constraints of the ontology. These are only a few of the many variations that can be developed under our framework. In this chapter, we examine how the framework can be extended in this manner.

The second dimension is the improvements that might be made to the specific implementation we have provided as a proof of the framework's applicability. These are enhancements that would build upon the existing Ontos approach rather than introducing entirely new ways of performing data extraction.

The third dimension is the avenues of future research made practical by our work. Two such avenues are declarative extraction plans and automated precision and recall computation.

6.1 Extending the framework

This section examines how the current framework code can serve as the basis for development of new modules for a data-extraction system.

6.1.1 Creating implementations and subclasses

Two primary ways exist for new modules to be written for the framework. First, entirely new classes can be created, implementing the interfaces defined by the framework. An example of this is our Ontos implementation written for the framework: all of the modules were written "from scratch" under the constraints imposed by the interface definitions. As another example, if one wished to develop a means to store the extracted data in a database, one could provide a new implementation of `OntologyWriter` to create the data model and generate the appropriate INSERT statements.

The second way to provide new code is to inherit the functionality of existing modules and override only those methods that are necessary to accomplish the desired result. This strategy is best if the functionality to be added represents a variation on existing capabilities, rather than a wholly different approach. For example, one might simply subclass `HTMLFilter` in order to adjust the types of markup that are stripped out, rather than re-implement the entire class.

Of course, one can directly modify an existing module's code to add functionality, if one has access to the source code. However, taking this approach is a much more permanent modification to the module and prevents one from switching back to the original feature set in order to perform side-by-side comparisons of performance or to establish a basis of reference for a series of experiments.

6.1.2 Plugging into existing components

A developer who has created new modules to be plugged into the framework must be able to instruct the system to use the new code, hopefully without having to modify

the system source code or overwrite the original module class files. Our framework provides a way to specify which classes supply the module implementations at runtime, via a text-based configuration file containing attribute-value pairs in the format *attribute=value*. Switching module implementations is as straightforward as modifying an entry in this file. Further, the location of this file can be provided to the main engine class as an argument, so a developer can also switch implementations by simply pointing the engine to an alternate configuration file. Figure 26 shows an example of the contents of the configuration file. So long as the specified classes correctly implement the appropriate framework interfaces, they can in this manner be incorporated into an existing data-extraction system subscribing to the framework with minimal effort required.

```
documentRetriever=edu.byu.deg.ontos.LocalDocumentRetriever
ontology=URL_to_OSMX_file
valueRecognizer=edu.byu.deg.ontos.DataFrameMatcher
retrievalPath=URL_to_input_document_directory
retrievalPattern=obituaries-test.html
valueMapper=edu.byu.deg.ontos.HeuristicBasedMapper
outputFile=pathname_of_output_OSMX_file
ontologyWriterOutputFile=pathname_of_output_HTML_file
ontologyWriter=edu.byu.deg.ontos.ObjectRelationshipWriter
documentStructureParser=edu.byu.deg.framework.DSP.fanout.FanoutRecordSeparator
extractionPlan=edu.byu.deg.ontos.OntosExtractionPlan
```

Figure 25: Sample `DataExtractionEngine` configuration file.

6.2 Enhancements to the new Ontos implementation

In addition to developing new modules to plug into the framework, another area of possible future work is enhancement of the Ontos implementation provided for this thesis. This implementation, while capable of the same performance as the old Ontos system, can be improved in many ways. We discuss a number of these possibilities below.

6.2.1 Additional OSM support

One area in which Ontos can improve is in its adherence to the OSM specification. While Ontos supports many of the OSM constructs, there are certain features that it does not yet support, and there are some constraints that it ignores. We discuss a few of the possible ways that Ontos can better support the OSM specification.

Model-wide constraints

Ontos attempts to satisfy certain constraints specified in the input ontology, particularly the participation constraints. However, it does so only at a localized level; the constraints are only honored within the context of a local primary object. Participation constraints are not necessarily honored across a data instance; an object (particularly a lexical object) might participate in several relationships across the data instance even though the relationship set's maximum participation constraint is one. For example, nothing prevents a single SocialSecurityNumber lexical value from appearing in relationships with two different USCitizen primary objects, even if the model constrains its maximum participation cardinality to one.

Furthermore, model-wide constraints such as object set cardinality or generalization-specialization constraints are not repected by Ontos, and there is no validation logic to make sure that the data instance generated by the mapping algorithm is valid with respect to the constraints of the ontology. Implementation of such logic and enforcement of these constraints during the extraction phase can enable researchers who wish to reason about the data to rely upon the ontology as a fully controlling specification for the data instances.

Canonicalization

The ability to reason about the data extracted by Ontos is affected by the data type of the extracted data. Currently, Ontos represents all extracted data as strings, but OSMX allows data frames to specify an alternate data type for the extracted values, such as Integer, URL, or Date. Since Ontos extracts only text strings, in order to use extracted values as any other data type we must provide a means to convert the values from their string representation.

OSMX provides support for this via the `defaultCanonicalizationMethod` attribute of a data frame. This attribute gives the name of a method that takes a string as input and returns a value of the data type specified by the data frame. For instance, a method `toInteger(String)` might be appropriate for canonicalizing string values that represent integers. OSMX also supports associating individual value phrases with different canonicalization methods, in case the phrases match different units of measure or representations that are better handled by different conversion methods. The reference implementation does not use this feature, but we foresee situations where canonicalization can be potentially useful.

6.2.2 Improvements to heuristics

Another way to improve Ontos is to enhance the capabilities of the heuristic-based mapping algorithm. We suggest three specific features that can provide the algorithm with more information to make its decisions: use of confidence values, support for average or expected participation constraint cardinalities, and employment of negative-indicator keywords. This is by no means an exhaustive list.

Confidence values

Currently, the heuristic-based mapping algorithm makes binary decisions about whether to accept or reject a candidate value. It might be valuable to assign to each proposed mapping a number that represents the likelihood or confidence that the mapping is correct. This would allow for mappings to be compared and would also allow for probabilistic calculation of the overall likelihood of a data instance's correctness.

OSMX supports assignment of confidence values to value phrases and keyword phrases. For example, consider a `Date` object set for which two value phrases are defined. The first matches dates in `Month DD, YYYY` format; the second matches dates in `YY` format. Using confidence values, we could assign a confidence of 0.99 (on a scale from 0.0 to 1.0) to the first phrase since matches are highly likely to be actual date values, while the second phrase might merit only a confidence of 0.3 since a two-digit number might only represent a date value 30% of the time.

Using confidence values in this manner can open the way to employing machine-learning algorithms as well: assigning confidence values to keyword and value phrases based on actual ratios from previously extracted data completes the feedback cycle that is essential to machine learning. This is a potentially rich area of future research.

Expected participation cardinality

One of the key sub-problems addressed in the heuristic-based algorithm arises when attempting to infer relationships for an extracted object. In this situation, the problem is deciding which of the relationship set connections that attach to the current object set should be followed next. This is an important problem because choosing the

wrong processing order can lead to values being mapped to the wrong object set because it was processed earlier than the correct one.

With expected participation constraint values, we can implement a heuristic that chooses the relationship set connection with the greatest difference between the expected participation value and the actual number of objects extracted for that relationship set connection so far. For example, if we have two connections with expected participation values of 2.3 and 3.8, and have extracted one object for the first and three objects for the second, we would choose the first connection to process next since we are farther away from satisfying the expectation stated by the first connection than the second.

Similar to confidence values, expected participation cardinalities can be used in machine-learning algorithms, since they can be dynamically calculated based on the participation cardinalities encountered in previously extracted data instances.

Negative-indicator keywords

Keyword matches are used in the extraction algorithm as positive indicators of the correctness of a value match. For example, the keyword "born" is a strong indicator of the correctness of the mapping of the nearby "January 24, 1955" value match to the object set `BirthDate`.

It is logical to propose a counterpoint to this feature, whereby keyword matches may serve as negative indicators of the correctness of a value match. An example is the negative keyword "fax" for a `HomePhone` object set, since we would wish to exclude fax numbers from being mapped to the object set. Combined with confidence values, negative-indicator keywords would serve to bolster the accuracy of the mapping

algorithm by screening out matches that are obviously inappropriate for the object set in question.

6.3 Avenues of future research

Our development of the framework and the OSMX specification has opened doors to areas of research that were impractical under the legacy system. We do not attempt to address all such avenues here, but we examine two of these as representative: declarative extraction plans and automated computation of precision and recall.

6.3.1 Declarative extraction plans

An example of possible future research is support for declarative extraction plans. This feature would permit the extraction algorithm to be specified by a declarative script, rather than a hard-coded imperative Java method. Since declarative scripts can be generated either manually or by some automated process, the extraction algorithm becomes much more manipulable, which opens doors to extraction engines that adapt their behavior dynamically according to the current environment and inputs.

The `ExtractionPlan` abstract class does not specify whether extraction plans are imperative or declarative, so we can presumably replace the imperative implementation with a declarative module. Under such a scenario, the `ExtractionPlan` implementation would need at least to provide a method to locate or store the declarative script before the `execute()` method is invoked, since `execute()` is defined to take no arguments. The details of parsing and processing the declarative instructions are entirely up to the implementor to handle. Access to other modules in the extraction framework is available via the `getInitializationParameter()` method of the `DataExtractionEngine`

reference required by `ExtractionPlan`'s constructor. No code needs to be changed in the `DataExtractionEngine` class or any other module in order to implement and integrate this functionality.

6.3.2 Automated computation of precision and recall

Tallying precision and recall figures from a data-extraction technique is a laborious process even on relatively small datasets. Because the legacy system output extracted data to a relational database, a considerable amount of metadata used to answer accuracy questions (such as extraction location, which informs us whether we have extracted a partial value or an incorrect value) was unavailable after the extraction process completed. This made it difficult to assess accuracy except by manual inspection of the results.

Since extracted data is now stored inline with the ontology in OSMX, together with essential metadata linking the extracted values back to their origins in the source documents as well as to the elements of the ontology to which they map, it is much more feasible to perform an automated comparison of extraction results. A human would tag a document or corpus to provide a reference data instance, and then the automated extraction would take place. With both data instances stored alongside the ontology, an automated recall/precision computer can simply take the ontology as input, traverse the object-relationship trees within the data instances, compare values, and report the results. Without such automation, a human has to attempt to correlate each piece of data from the test data instance with those in the reference data instance—an extraordinarily time-consuming and error-prone process even when only one test data instance needs to be assessed. By reducing the cost of comparing data instances, automating precision and

recall calculations can make possible extraction experiments at a much larger scale than what has heretofore been practical.

7 Conclusion

We have proposed, designed, and provided a reference implementation for a framework for ontology-based data extraction. This framework offers improved modularity and extensibility to support further data-extraction research. We have demonstrated that the framework is sufficiently developed to support a re-implementation of Ontos that preserves the quality of legacy Ontos while also using modular heuristics code.

Additionally, we have designed an XML Schema, OSMX, that provides an XML storage definition for OSM ontologies. Newly added features give OSMX greater capabilities for representing data-extraction ontologies and the instance data extracted for them. A library of JAXB-generated Java classes supports programmatic access to OSMX-compliant documents. This library allows Ontos, OntologyEditor, and future tools to exchange ontologies and data, expanding the research possibilities while minimizing the need for specialized information interchange protocols or file formats.

The products of these efforts provide a solid basis for continued research on ontology-based data extraction. Future researchers will be better able to focus on specific problems in the field while maintaining confidence that plugging their code into the existing system and comparing the results can rapidly validate their work. The value of this thesis is in future research opportunities made possible or practical because of the framework.

8 Bibliography

[Abi97] Abiteboul, Serge. "Querying semi-structured data." In *Proceedings of the 6th International Conference on Database Theory*, Delphi, Greece, 8-10 January 1997, pp. 1-18.

[AK97] Ashish, N. and C. Knoblock. "Wrapper generation for semi-structured Internet sources," *SIGMOD Record*, Volume 26, Number 4, December 1997, pp. 8-15.

[Apa04] *The Struts Project*. The Apache Software Foundation, 2004. At http://struts.apache.org/.

[Cam04] "Framework" (dictionary entry). *Cambridge Advanced Learner's Dictionary*. Cambridge Dictionaries Online, 2004. At http://dictionary.cambridge.org.

[CEL04] Chen, X., D.W. Embley, and S.W. Liddle. "Query rewriting for extracting data behind HTML forms." In *Proceedings of the International Workshop on Conceptual Model-directed, Web Information Integration and Mining*, Shanghai, China, 8-12 November 2004, pp. 335-345.

[Cha03] Chartrand, Tim. "Ontology-based extraction of RDF data from the World Wide Web." Master's thesis, Brigham Young University, March 2003.

[Che04] Chen, Xueqi. "Query rewriting for extracting data behind HTML forms." Master's thesis, Brigham Young University, March 2004.

[CMM01] Crescenzi, Valter, Giansalvatore Mecca, Paolo Merialdo. "RoadRunner: Towards automatic data extraction from large Web sites," In *Proceedings*

of the 27th International Conference on Very Large Data Bases, Roma, Italy, 11-14 September 2001, pp. 109-118.

[Din03] Ding, Yihong. "Semiautomatic generation of resilient data-extraction ontologies." Master's thesis, Brigham Young University, June 2003.

[DMR02] Davulcu, H., S. Mukherjee, I.V. Ramakrishnan. "Extraction techniques for mining services from Web sources," In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, Maebashi, Japan, 9-12 December 2002, pp. 601-604.

[ECJ+99] Embley, D.W., D.M. Campbell, Y.S. Jiang, S.W. Liddle, D.W. Lonsdale, Y.-K. Ng, R.D. Smith. "Conceptual-model-based data extraction from multiple-record Web pages," *Data & Knowledge Engineering* 31 (1999), pp. 227-251.

[Eik99] Eikvil, Line. "Information extraction from the World Wide Web: A survey," *Technical Report 945*, Norwegian Computing Center, 1999.

[EJN99] Embley, D.W., Y.S. Jiang, Y.-K. Ng. "Record-boundary discovery in Web documents." In *Proceedings of 1999 ACM SIGMOD International Conference on Management of Data*, Philadelphia, Pennsylvania, 31 May–3 June 1999, pp. 467-478.

[EKW92] Embley, D.W., Barry D. Kurtz, Scott N. Woodfield. *Object-Oriented Systems Analysis: A Model-Driven Approach*. Yourdon Press, 1992.

[Emb80] Embley, David W. "Programming with data frames for everyday data items," *AFIPS '80 Proceedings*, Anaheim, California, 19-22 May 1980, pp. 301-305.

[Eng02] Engels, Robert. "Del 7: CORPORUM OntoWrapper: Extraction of structured information from web based resources." On-to-Knowledge Consortium, 2002. At http://www.ontoknowledge.org.

[ETL02] Embley, D.W., C. Tao, S.W. Liddle. "Automatically extracting ontologically specified data from HTML tables with unknown structure." In *Proceedings of the 21st International Conference on Conceptual Modeling*, Tampere, Finland, 7-11 October 2002, pp. 322-327.

[GHJV95] Gamma, Erich, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, 1995.

[Hew00] Hewett, Kimball. "An integrated ontology development environment for data extraction." Master's thesis, Brigham Young University, April 2000.

[HGN+97] Hammer, Joachim, Hector Garcia-Molina, Svetlozar Nestorov, Ramana Yerneni, Marcus Breunig, Vasilis Vassalos. "Template-based wrappers in the Tsimmis system," In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, 13-15 May1997, pp. 532-535.

[JAXB03] *JSR 31: XML Data Binding Specification*, March 2003. At http://jcp.org/en/jsr/detail?id=31.

[KWD97] Kushmerick, N., D. Weld, R. Doorenbos. "Wrapper induction for information extraction," In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Nagoya, Japan, 23-29 August 1997, pp. 729-737.

[LEW00] Liddle, S.W., D.W. Embley, S.N. Woodfield. "An active, object-oriented, model-equivalent programming language." *Advances in Object-Oriented Data Modeling*, MIT Press, 2000, pp. 333-361.

[LR+02] Laender, A.H.F., B.A. Ribeiro-Neto, A.S. da Silva, J.S. Teixeira. "A brief survey of Web data extraction tools," *SIGMOD Record*, Volume 31, Number 2, June 2002, pp. 84-93.

[Mai83] Maier, David. *Theory of Relational Databases*. W.H. Freeman & Co., March 1983.

[McN04] McNichol, Tom. "Engineering Google results to make a point," *The New York Times*, Jan. 22, 2004. http://www.nytimes.com/2004/01/22/technology/circuits/22goog.html

[Ora04] "Oracle Application Development Framework Overview," Oracle Corporation, April 2004. At http://www.oracle.com/technology/products/jdev/collateral/papers/10g/ADF_overview.pdf.

[SFM03] Shah, Urvi, Tim Finin, James Mayfield. "Information retrieval on the Semantic Web." In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, McLean, Virginia, 4-9 November 2002, pp. 461-468.

[Sun04] "Collections," *The Java™ Tutorial*, Sun Microsystems, Inc., 2004. At http://java.sun.com/docs/books/tutorial/collections/index.html.

[Tao03] Tao, Cui. "Schema matching and data extraction over HTML tables." Master's thesis, Brigham Young University, September 2003.

[W3C01] *XML Schema*. The World Wide Web Consortium, May 2001. http://www.w3.org/XML/Schema

[W3C04] *Web Ontology Language (OWL)*. The World Wide Web Consortium, February 2004. At http://www.w3.org/2004/OWL/.

[W3C99] *HTML 4.01 Specification*. The World Wide Web Consortium, December 1999. At http://www.w3.org/TR/REC-html40/.

[WE04] Walker, Troy and David W. Embley. "Automatic location and separation of records: a case study in the genealogical domain." In *Proceedings of the International Workshop on Conceptual Model-directed Web Information Integration and Mining*, Shanghai, China, 8-12 November 2004, pp. 302-313.

[Wes02] Wessman, Alan. "Ontology-based data extraction with a hidden Markov model learner." Master's thesis proposal (unpublished), 2002.

[Wes04] Wessman, Alan. *OSMX Specification*, 2004. At http://www.deg.byu.edu/xml/osmx.xsd.

9 Appendix

This appendix provides references to documentation on the systems and libraries developed for this thesis, since the documentation is too large to publish directly in this appendix.

9.1 Framework documentation

The framework API documentation may be found at http://www.deg.byu.edu/api/framework/index.html.

9.2 OSMX documentation

9.2.1 XML Schema definition

The current OSMX definition, which is an XML Schema instance, may be found at http://www.deg.byu.edu/xml/osmx.xsd.

9.2.2 Documentation for implementation subclasses

Documentation for the classes used to represent OSMX structures in Java is located at http://www.deg.byu.edu/api/osmx/index.html.

9.3 OntosEngine documentation

API documents for the reference implementation of the framework is located at http://www.deg.byu.edu/api/ontos/index.html.