2005-11-22

Generating Data-Extraction Ontologies By Example

Yuanqiu Zhou
Brigham Young University - Provo

GENERATING DATA-EXTRACTION ONTOLOGIES BY EXAMPLE

by

Yuanqiu Zhou

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

December 2005

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by
Yuanqiu Zhou

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	Michael Jones

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Yuanqiu Zhou 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	Parris K. Egbert Graduate Coordinator
Accepted for the College	G. Rex Bryce, Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

GENERATING DATA-EXTRACTION ONTOLOGIES BY EXAMPLE

Yuanqiu Zhou

Department of Computer Science

Master of Science

Ontology-based data-extraction is a resilient web data-extraction approach. A major limitation of this approach is that ontology experts must manually develop and maintain data-extraction ontologies. The limitation prevents ordinary users who have little knowledge of conceptual models from making use of this resilient approach. In this thesis we have designed and implemented a general framework, OntoByE, to generate data-extraction ontologies semi-automatically through a small set of examples collected by users. With the assistance of a limited amount of prior knowledge, experimental evidence shows that OntoByE is capable of interacting with users to generate data-extraction ontologies for domains of interest to them.

ACKNOWLEDGMENTS

I am indebted to many people for their assistance and support to my thesis work. I would like to give my great appreciations to the following people:

All my committee members for their time and efforts on my behalf. Particularly, my graduate advisor and committee chair, *Dr. David Embley*, for his invaluable suggestions to my research and for his attentions in his very busy days.

All my fellow members at DEG group, whose work inspired me and provided me with various tools. Special thanks to my friends: *Yihong Ding, ZongHui Lian* and *Cui Tao* for their assistances during my trips back to Provo from Los Angeles.

My friends: *Jie Liu, Janice Fu* and *Jiayun Zhuang* in Los Angeles where I am currently living and working. Without their kindly care through my physical hardship since early this year, I could never possibly finish my thesis on time.

My friend and co-worker, *Robert Humphreys*, a senior technical writer at J2 Global Communications Inc., for his time and efforts to polish my thesis.

My parents and siblings, who always support me all the time in my life.

This research has been supported in part by the National Science Foundation under grant #IIS-0083127.

In memory of those days at Provo, UT and all those people whom I get to know there.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Related Work

The amount of useful information on the World Wide Web continues to grow at a stunning pace. Typically, humans browse web pages but cannot easily query desired content of the pages. Many researchers have expended a tremendous amount of energy working on the problem of how to extract semi-structured web data and convert it into a structured form that can be easily queried. They have proposed a number of different information-extraction (IE) approaches in the past decade; several surveys ([Eikvil99, Muslea99, and LRST02]) summarize these approaches.

The most common way to extract web data is by generating wrappers. Researchers have constructed wrappers manually (e.g. TSIMMIS [HGNY+97]), semi-automatically (e.g. RAPIER [CM99], SRV [Freitag98], WHISK [Soderland99], WIEN [KWD97], SoftMealy [Hsu98], STALKER [MMK99], XWRAP Elite [BLP01] and DEByE [RLS01]) and even fully automatically (e.g. RoadRunner [CMM01]). Since the extraction patterns generated by all these systems are more or less based on delimiters or HTML tags bound to the text to be extracted, they are sensitive to changes of web page format. In this sense, they are source-dependent, because they need to be either reworked or rerun to discover new patterns following source-page changes. Furthermore, the wrappers they produce do not work for new pages in the same domain.

To solve these problems, the Data Extraction Group ([DEG]) at Brigham Young University has developed a resilient approach to wrapper generation based on conceptual models or ontologies ([ECJL+99]). An ontology, which is defined using a conceptual

model, describes the data of interest, including relationships, lexical appearance, and context keywords. Since the ontology-based approach does not depend on delimiters or HTML tags to identify the data to be extracted, the ontology developed for a particular domain works for all web pages in that domain, and is not sensitive to changes in web page format. By parsing the ontology, the BYU system automatically produces a database schema and recognizers for constants and keywords. A major limitation of this ontology approach, however, is that ontology experts must manually develop and maintain the ontology. Thus, a principal effort of our current research aims to generate ontologies, if not automatically, at least semi-automatically.

One possible solution to semi-automatically generate ontologies is a "by-example" approach motivated by Query by Example (QBE) ([Zloof77]) and Programming by Example (PBE) ([PBE01]). The DEByE (Data Extraction by Example) system ([FSLE02a, FSLE02b and LRS02]) was the first to make use of an example-based approach for web data-extraction. This approach offers a demonstration-oriented interface in which the user shows the system which information to extract. Using a graphical interface, the user performs extraction by example, showing the application which data to extract. This by-example approach is relatively user-friendly, in that it does not require that the user possess expert knowledge in wrapper coding. However, since DEByE uses delimiter-based extraction patterns and cannot induce the structure of the concepts in the domain of interest, it is brittle: a DEByE-generated wrapper will break when a site changes or when it encounters a new site with a different structure.

1.2 Thesis Statement

In this thesis work we have employed the by-example approach to build a system called OntoByE (Ontology By Example) that semi-automatically generates ontologies for our conceptual-model-based data-extraction system. The OntoByE system is designed to provide users with an intuitive interface through which a small number of data examples are collected and subsequently used to construct an extraction ontology for general use in the domain of user interest. Utilizing a set of forms, the OntoByE system interface first helps users to define the data in which they are interested. Users are then guided through a process during which they show OntoByE sample data of interest from a small number of web pages, highlighting the desired data and filling in the forms. Finally, OntoByE generates data-extraction ontologies based on user-defined forms and the information gathered from sample pages. OntoByE does not use HTML tags or page-dependent delimiters when generating its data-extraction ontology, thereby retaining the resilience to induce the concepts in the domain of interest. We built our OntoByE system with assistance of previous work done in the Data Extraction Group of Brigham Young University. OntoByE provides a more efficient way to generate ontologies for our conceptual-model-based data-extraction system and serves as a useful tool to facilitate future ontology-based data-extraction research by the DEG group.

This thesis presents details specific to the development and design of the OntoByE system. To keep this thesis self-contained, Chapter 2 briefly explains the conceptual model used to specify the domain of interest for data extraction. Chapter 3 describes a system-wide overview of OntoByE. Chapter 4 demonstrates the usage of the OntoByE system interface. Chapter 5 describes the back-end ontology generation

process. Chapter 6 describes our experience with brief field tests and subsequent observations and analyses of the strengths and weaknesses of OntoByE. Finally, Chapter 7 concludes the thesis, discusses OntoByE limitations, and explores possibilities for future work.

2 Extraction Ontology

An extraction ontology is an instance of a conceptual model, or *Object-oriented System Model (OSM),* for a narrow domain of interest. It consists of two components:

- An *Object-Relationship Model (ORM)* instance that describes sets of objects sets of relationships among objects, and constraints over object and relationship sets. We represent the conceptual model of our ontologies as ORM diagrams as described in [Emb98].

- A *data frame* that defines the contents of each object set. A data frame for an object set defines the lexical appearance of constant objects for the object set and establishes appropriate keywords that are likely to appear in a document when objects in the object set are recognized.

In ORM diagrams, object sets are depicted as boxes with interconnecting lines. Figure 1 shows the ORM diagram of our ontology for a digital camera application, a running example that will be used throughout the thesis. Each object set depicted in the ORM diagram may be nonlexical, represented by a solid border, or lexical, represented by a dashed border. Lexical objects are objects that are indistinguishable from their representation. The object set *Price* is lexical because a price is indistinguishable from its representation as a number. Nonlexical objects are those that must be represented by identifiers. The object set *Digital Camera* is nonlexical because its instances are identifiers, such as DC01, DC02 and so on. In an extraction ontology, there is one and only one object set designated as the primary object set, the highest-level concept to be

extracted. The primary object set is denoted by an arrow followed by a dot (→●). In our Digital Camera ontology diagram in Figure 1, *Digital Camera* is the primary object set.

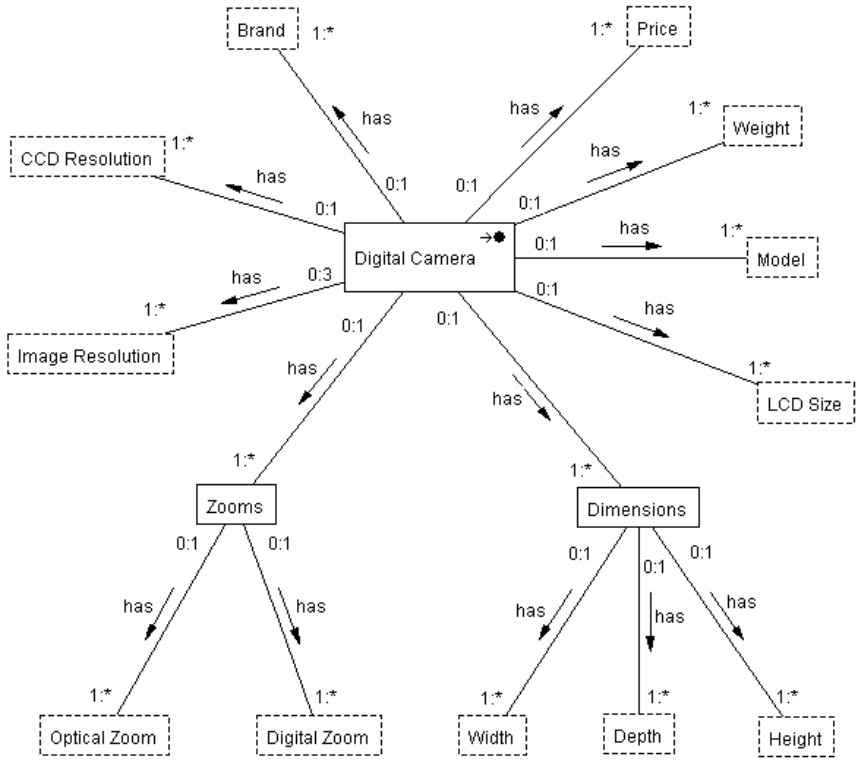


Figure 1: An Ontology Diagram for the Digital Camera Application

Relationship sets, depicted by lines between the boxes in ORM diagrams, connect the object sets in an extraction ontology. Each relationship set has a name and a reading-direction. For example, the relationship between *Digital Camera* and *Price* reads, *Digital Camera has Price*. Each relationship set is also labeled with a participation constraint that indicates the number of times an object in the object set may participate in this relationship set. The participation constraint consists of a minimum, and a maximum number, each separated by colons. In our notation, a star represents an arbitrarily large number. For example, the participation constraint *0:1* next to *Digital Camera* on the

relationship set between *Digital Camera* and *Price* indicates that a *Digital Camera* object may be related to at most one *Price*. The participation constraint *1:** next to *Price* indicates that a *Price* object may be related to one or more *Digital Camera* objects.

In addition to components found in an ORM diagram, an extraction ontology also has a data frame for each object set. A data frame contains a list of value phrases and, in some cases, a list of keyword phrases. A value phrase consists of a value expression with encapsulating left and right context expressions. A value expression is an extraction pattern, written as a regular expression, that describes the data content to be extracted. A context expression describes the characters immediately adjacent to the beginning and end of the data content of a value phrase. A keyword phrase consists of keyword expressions which are typically meaningful indicators of the presence of a particular set of values. Keywords occur near, but not necessarily immediately adjacent to, data to be extracted. Figure 2 shows a sample data frame for an object set named *Digital Zoom* from our *Digital Camera* ontology. The value phrase in the data frame *Digital Zoom* matches a digit, optionally followed by a period and another digit, but only when followed by an character "x". The trailing "x" may be uppercase or lowercase and it can be separated from the preceding number by a white-space character (e.g. space or tab). Sometimes, the value phrase of a data frame matches with multiple value candidates for an object, such as *4.1* and *3* in "*4.1 x* digital zoom" and "*3 x* optical zoom" for the digital zoom of a digital camera. In such a case, the presence of the keywords from the data frame, such as "Digital Zoom" or "digital zoom", will help the data-extraction engine to select the most appropriate candidate, such as *4.1* in this example.

Value Phrase
Value Expression: `\d(\.\d)?`
Left Context Expression:
Right Context Expression: `(\s)?(x|X)`

Keyword Phrase
Keyword Expression: `Digital\sZoom|digital\szoom`

Figure 2: A Data Frame for the Object Set Digital Zoom

Given a data-extraction ontology, such as a *Digital Camera* ontology, we can apply it to text such as the digital camera advertisement in Figure 3. In Figure 3, the highlighted text (*4.1*) is recognized by the data frame *Digital Zoom* in the extraction ontology. Given the recognized text, we can use the ontology, its constraints and implied relationships, to extract the information and populate the ontology with corresponding instance data.

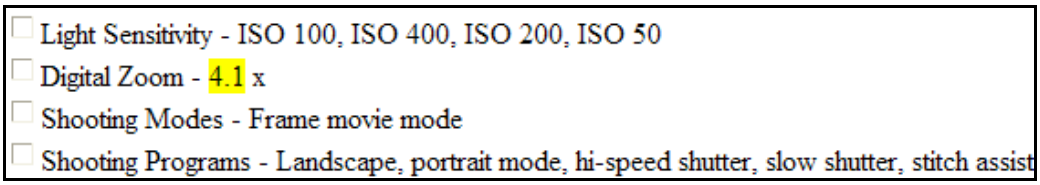


Figure 3: A Partial Sample Digital Camera Advertisement

Our general approach to information extraction consists of the following steps.

1. We develop the data-extraction ontology over the area of interest.

2. We parse this ontology to generate a database schema and to generate rules for matching constants and keywords.

3. Given an applicable web page with multiple records (like classified ads), we invoke a record extractor that separates an unstructured web document into

individual record-size chunks, removes markup-language tags, and presents them as individual unstructured record documents for further processing.

4. We invoke recognizers that employ matching rules obtained from the data frames to identify potential constant data values and their keywords and keyword expressions in the cleaned records.

5. Finally, we populate the generated database by using heuristics to determine which constants populate which records in the database. These heuristics correlate extracted keywords with extracted constants and use cardinality constraints in the ontology to determine how to construct records and insert them into the database. Once the data is extracted, we can issue queries using a standard database query language. To make our approach general, we fix the ontology parser, web record extractor, keyword and constant recognizer, and database record generator; we change only the ontology as we move from one application domain to another.

3 OntoByE System Architecture

The OntoByE system consists of three major components:

- A graphical user interface (GUI)
- A data frame library
- A back-end data-extraction ontology generator

Figure 4 illustrates the OntoByE system architecture. Subsequent sections describe the function of each component in detail.

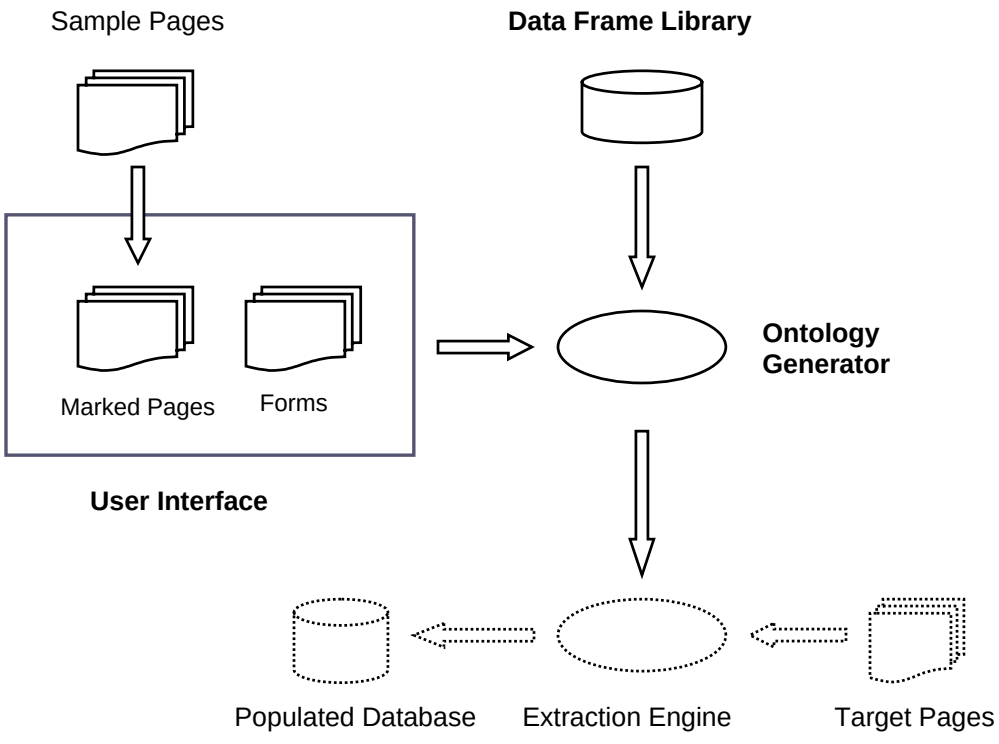


Figure 4: OntoByE System Architecture

3.1 User Interface

OntoByE presents a graphical user interface that helps users describe the data in which they are interested and provide sample data values from sample HTML pages. A form editor assists users in creating application-dependent forms that allow them to describe the data in which they are interested. Users then specify, or mark, the desired data values by filling in user-defined forms with values from sample HTML pages. Finally, the interface allows users to save both forms and marked HTML training pages for further processing.

As shown in Figure 5, the GUI is embedded in a web browser and consists of two panes. The left pane of the GUI allows a user to display a sample HTML page through the web browser. The right pane contains a form editor Java applet running within the browser. The form editor helps users define application-dependent forms. In typical operation, a user would define a form in the right pane and upload a sample page in the left pane. The user then selects the desired data on the sample page and fills in the forms on the right. Simultaneously, values selected on the sample page are marked with special tags indicating the labels as specified in the form. Finally, users can save their forms and marked HTML pages in designated directories on their local machine for future modifications and further ontology generation. Chapter 4 describes the usage of the interface in detail.

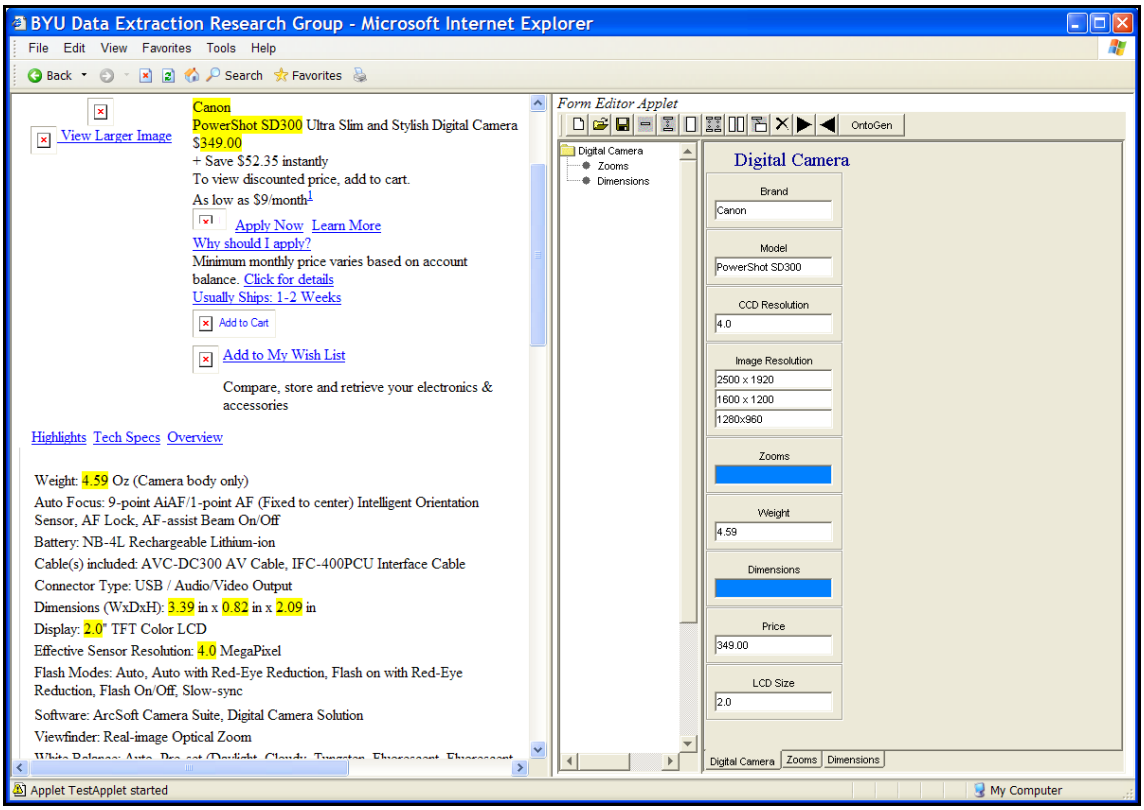


Figure 5: OntoByE System User Interface

3.2 Data Frame Library

A data frame library is a collection of prior knowledge in the form of data frames for data values to be extracted. To do ontology-based web data-extraction, experts need to construct an initial set of data frames to accommodate some common types of data (e.g. number, date, phone number, and price), which apply across different data-extraction applications. In the process of ontology generation by OntoByE, the system takes advantage of the prior knowledge in the library by searching for appropriate data frames for user-marked data on the sample pages. Users then interact with OntoByE to select existing data frames or to construct new data frames for their applications. The new data frames could be used to expand the library for other data-extraction applications

in the future. As times goes by, the library will grow more comprehensive and the need for expert involvement and user interaction with the library will be diminished.

3.3 Ontology Generator

As described in Chapter 2, a data-extraction ontology consists of two major components: (1) object and relationship sets together with related constraints, and (2) data frames for the object sets. The OntoByE's ontology-generator component performs the back-end generation of a compliant data-extraction ontology using the user defined forms and marked HTML pages from the GUI as inputs with the assistance of the data frame library. The ontology generator consists of the following sub-components:

- The *Form Analyzer* constructs object and relationship sets, along with their constraints, based on the user-defined forms.
- The *Context Phrase Locator* extracts a list of context phrases within marked data for each object set from the marked HTML pages and passes context phrases to the *Data Frame Matcher*.
- The *Data Frame Matcher* matches the list of context phrases against all data frames from the data frame library to find all matching data frames for each object set, and ranks the matching data frames for each object set based on some heuristics.
- The *Keyword and Context Expression Recognizer* scans the list of context phrases to recognize possible keywords and context expressions for user-marked data.

- The *Data Frame Editor* presents a ranked list of matching data frames, along with recognized keywords and context expressions, for the user's examination. The user selects an appropriate existing data frame, if any, from the ranked list of matching data frames returned by the *Data Frame Matcher*, or creates a new data frame for the domain of their interest with the assistance of those matching data frames in the library and keywords and context expressions constructed by the *Keyword and Context Expression Recognizer*.

The workflow of these sub-components of the ontology generator is shown in Figure 6. After the user selects or constructs a data frame for each object set, OntoByE combines the object and relationship sets and constraints constructed from the forms to generate a data-extraction ontology for the domain of interest.

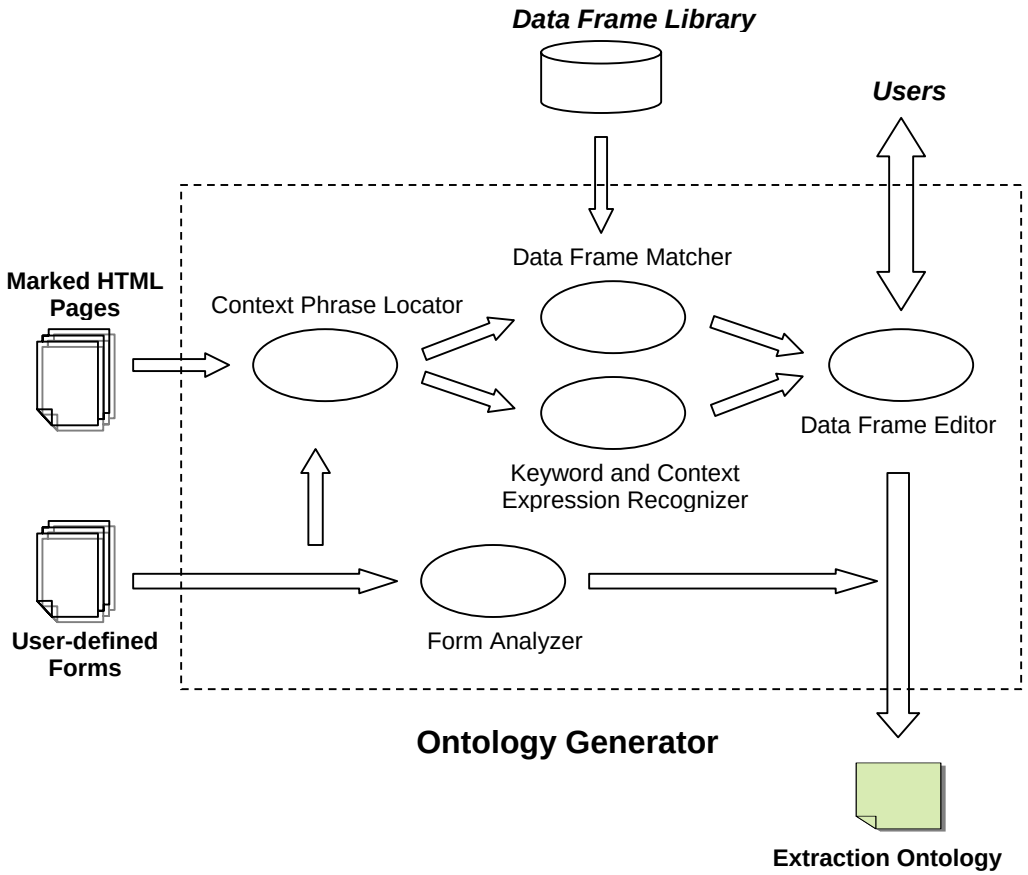


Figure 6: The Workflow of Ontology Generator

4 User Interface

OntoByE provides a GUI, shown in Figure 5 of Chapter 3, through which web users can define forms, provide sample HTML pages, fill in the forms with desired data and mark the data with special tags on the training pages for future processing. The following sections describe the form creation and the marked HTML page preparation in detail.

4.1 Creating Forms

The form editor is a Java applet tool that provides users with an intuitive method for defining forms. First of all, it allows users to give forms meaningful titles. Then, it provides five basic patterns, or building blocks, through a toolbar with which users can construct form elements. After users title a form, they can add to the current form any number of elements by clicking on patterns or icons in the toolbar. Although users can define one and only one base form for each application, they can recursively construct nested forms inside elements of the base form. The nested forms allow users to describe their interests in more structured and meaningful ways and are defined in separate panels in the same way as users define the base form. The following sections show how to use the form editor to create form elements and nested forms, and then demonstrate a set of forms for a Digital Camera application.

4.1.1 Form Titles

To create a base form, a user needs to give the form a meaningful title for a real application of interest, such as *Digital Camera, Car, Book* and so on. The default title that the editor provides is *BaseForm*. Users can specify the form title when creating a

form, or can change the title of an existing form at any time by clicking on the title in the form and typing another name.

4.1.2 Form Editor Toolbar

The form editor presents a tool bar, shown in Figure 7, to help users create, edit and save forms. The usage of toolbar icons are summarized as follows: The first three icons  are a set of icons for New, Open and Save operations on form files. The next five icons  create form elements in five pre-defined basic patterns. The next icon  creates nested forms inside form elements. The next icon  deletes elements from forms. The following icons  are for filling in or deleting from forms the selected data values from sample HTML pages (See Section 4.2). Finally, the last icon  invokes the ontology generation process (See Chapter 5).

Figure 7: Form Editor Toolbar

4.1.3 Creating Basic Form Elements

After users specify form titles, they use the  icons on the toolbar to add form elements in five basic patterns, pattern A through E from left to right respectively. Each form element may contain one or more columns, each column having a label and one or more value fields. Users select the pattern that best describes the data of their interest and label the columns with meaningful names.

Toolbar icons for patterns A, B and C will allow users to construct a form element which represents a single column with one value field, a limited number of value fields or an unlimited number of value fields respectively. Patterns D or E will help users generate a form element which represents a group of columns with a limited number of value fields or an unlimited number of value fields respectively.

4.1.3.1 Creating Form Elements in Pattern A

For the form elements in pattern A, the number of columns and the number of value fields are both pre-set to 1. To add an element from pattern A to a form, users specify the label of the column in an input dialog as shown in Figure 8(a).

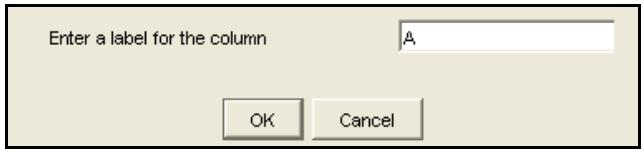


Figure 8(a): Input Dialog for Creating Form Elements in Pattern A

The form editor adds an element consisting of a single column and a single value field with the column label provided in the input dialog. Figure 8(b) shows an element generated from pattern A based on the information from the input dialog in Figure 8(a).

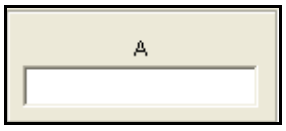


Figure 8(b): A Sample Form Element Created in Pattern A

4.1.3.2 Creating Form Elements in Pattern B

For the form elements in pattern B, the number of columns is pre-set to 1. Users specify the label of the column, such as B, and the number of value fields in the column, such as 3, in input dialogs as shown in Figure 9(a).

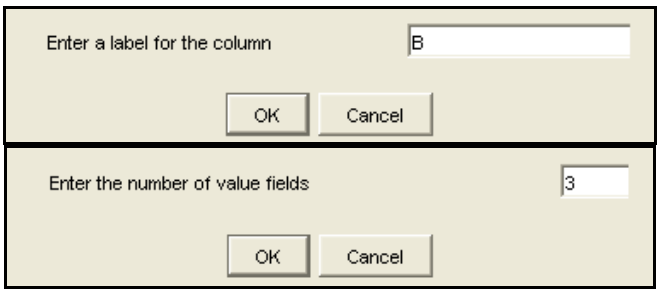


Figure 9(a): Input Dialogs for Creating Form Elements in Pattern B

The form editor adds an element consisting of a single column and a limited number of value fields with the column label provided in the input dialogs. Figure 9(b) shows an element generated from pattern B based on the information from the input dialogs in Figure 9(a).

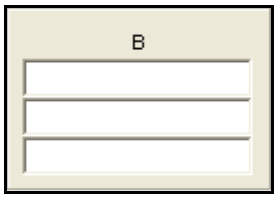


Figure 9(b): A Sample Form Element Created in Pattern B

4.1.3.3 Creating Form Elements in Pattern C

For the form elements in pattern C, the number of columns is pre-set to 1 and the number of value fields is pre-set to unlimited. Users specify the label of the column, such as C, through an input dialog shown in Figure 10(a).

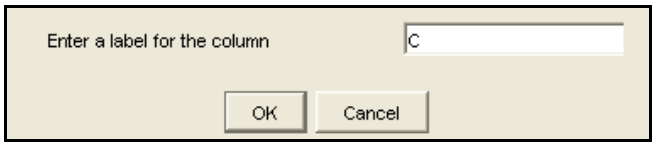


Figure 10(a): Input Dialog for Creating Form Elements in Pattern C

The form editor adds an element consisting of a single column and a text area with the column label provided in the input dialogs. The form editor uses the text area to represent an unlimited number of value fields. Figure 10(b) shows an element generated from pattern C based on the information from the input dialog in Figure 10(a).

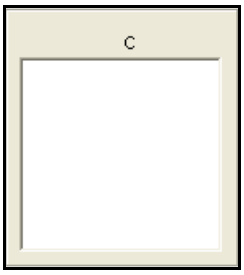


Figure 10(b): A Sample Form Element Created in Pattern C

4.1.3.4 Creating Form Elements in Pattern D

For form elements in pattern D, through the input dialogs depicted in Figure 11(a), users specify the number of columns, the number of values fields, and a label for each column.

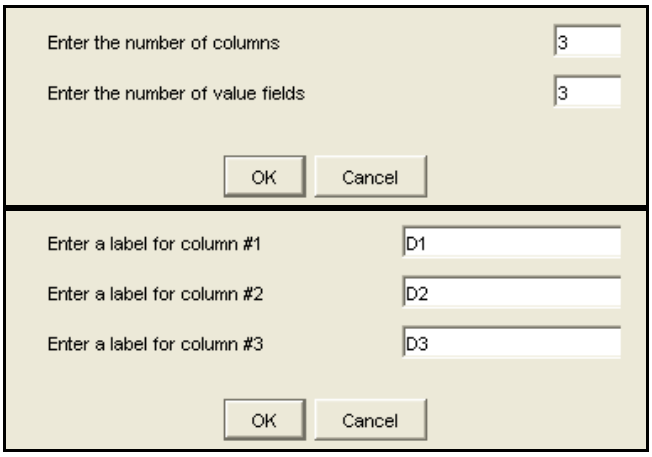


Figure 11(a): Input Dialogs for Creating Form Elements in Pattern D

The form editor adds an element consisting of multiple columns and multiple value fields with the labels provided in the input dialogs. Figure 11(b) shows an element generated using pattern D based on the information from the input dialogs in Figure 11(a).

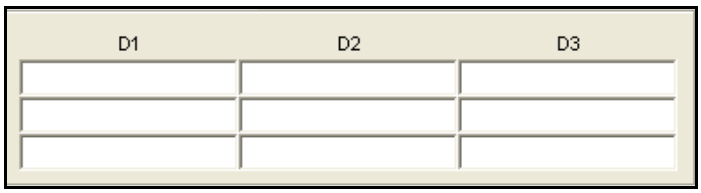


Figure 11(b): A Sample Form Element created in Pattern D

4.1.3.5 Creating Form Elements in Pattern E

For the form elements in pattern E, the number of value fields is pre-set to unlimited. Through the input dialogs depicted in Figure 12(a), users specify the number of columns, and the label for each column.

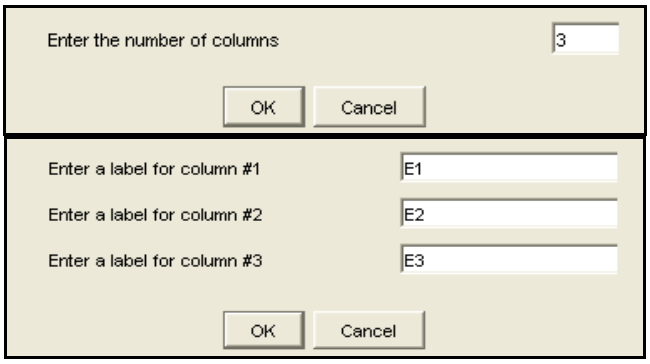


Figure 12(a): Dialogs for Creating Form Elements in Pattern E

The form editor adds an element consisting of multiple columns and multiple text areas with the labels provided in the input dialogs. The form editor uses a text area to represent an unlimited number of value fields. Figure 12(b) shows an element generated using pattern E based on the information from the input dialogs in Figure 12(a).

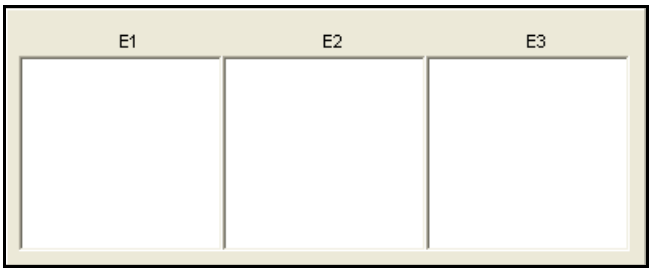


Figure 12(b): A Sample Form Element Created in Pattern E

Figure 13 shows a final sample form titled “Base Form” with the form elements created in Figures 8-12.

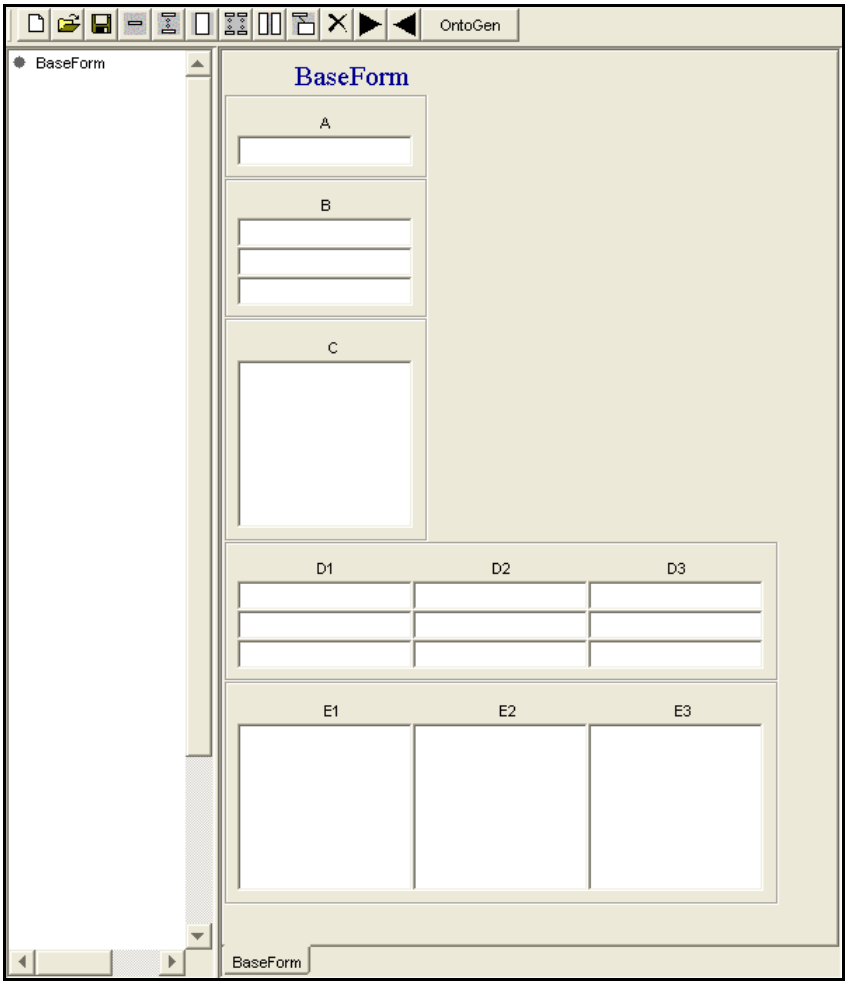


Figure 13: A Sample Base Form with Elements of Five Basic Patterns

A previously added element can be removed from a form by selecting the undesired element and clicking  on the toolbar. The element will be deleted only after user confirmation of a deletion warning dialog.

4.1.4 Nesting Forms

After adding elements to the base form, users can further nest forms inside the elements if they intend to specify information for the elements. To nest a form inside a form element, users select a column in the element and click the nesting form icon  in the toolbar. The form editor will create a nested form for the selected column and title the nested form with the label of the selected column. To specify the information contained in the nested forms, users add form elements in the same way as they do to the base form. Furthermore, users can recursively nest forms inside the elements of other forms. Each column in the form elements may contain either text fields for data values or a nested form, but not both. Figure 14 illustrates the nested forms defined inside elements from patterns A, B and C.

In the form editor window, the nested forms are defined on separated panels in the same manner as the base form is defined. The editor provides two methods, shown in Figure 15, to help users navigate through forms or panels: 1) a tree pane on the left of the editor window which describes the hierarchy of forms and 2) a tabbed pane at the bottom of the editor window which shows form titles.

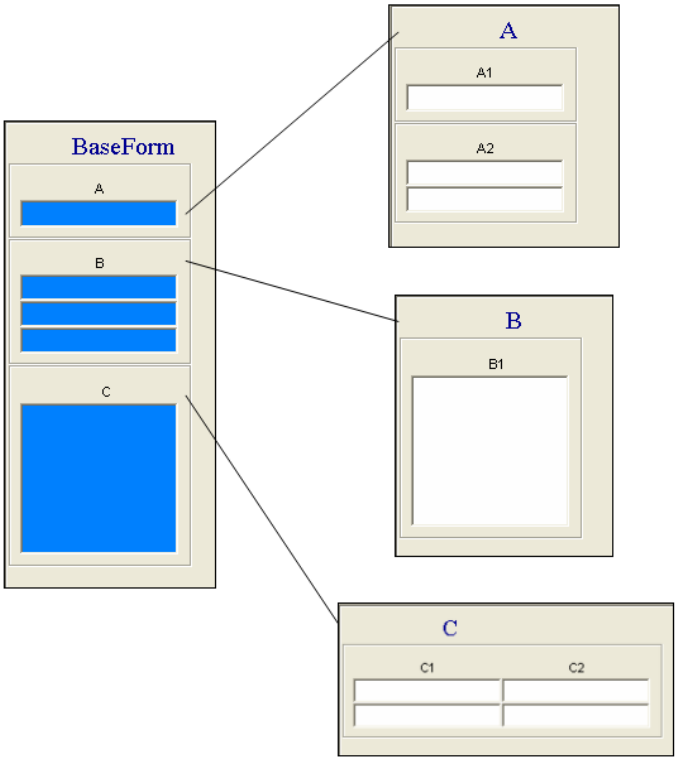


Figure 14: Nested Forms in the Base Form

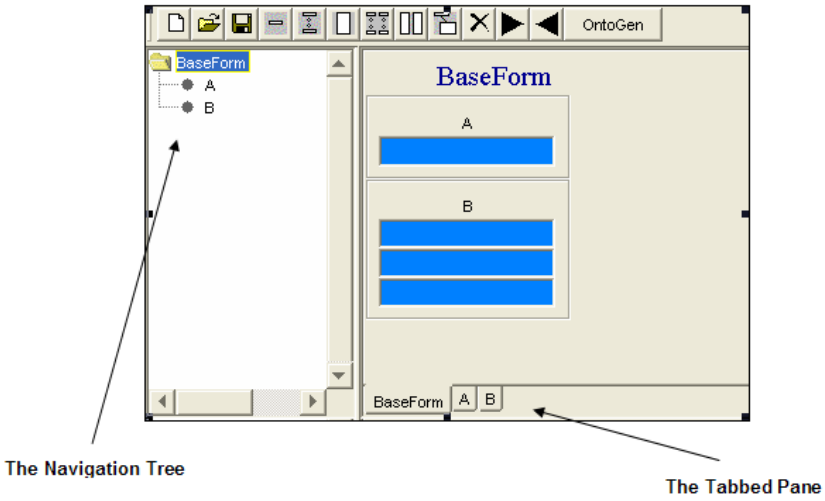


Figure 15: Navigating Nested Forms

4.1.5 Sample Application Dependent Forms

This section presents a set of forms created for a real application "Digital Camera". Shown in Figure 16, the base form, titled *Digital Camera*, has nine elements: *Brand, Model, CCD Resolution, Image Resolution, Zooms, Weight, Dimensions, Price*

and *LCD Size*. Among these elements, *Zooms* and *Dimensions* contain nested forms shown in Figure 16. Within the nested forms, *Zooms* contains the elements *Optical Zoom* and *Digital Zoom*, shown in Figure 17(a), while *Dimensions* contains *Width*, *Depth* and *Height*, shown in Figure 17(b).

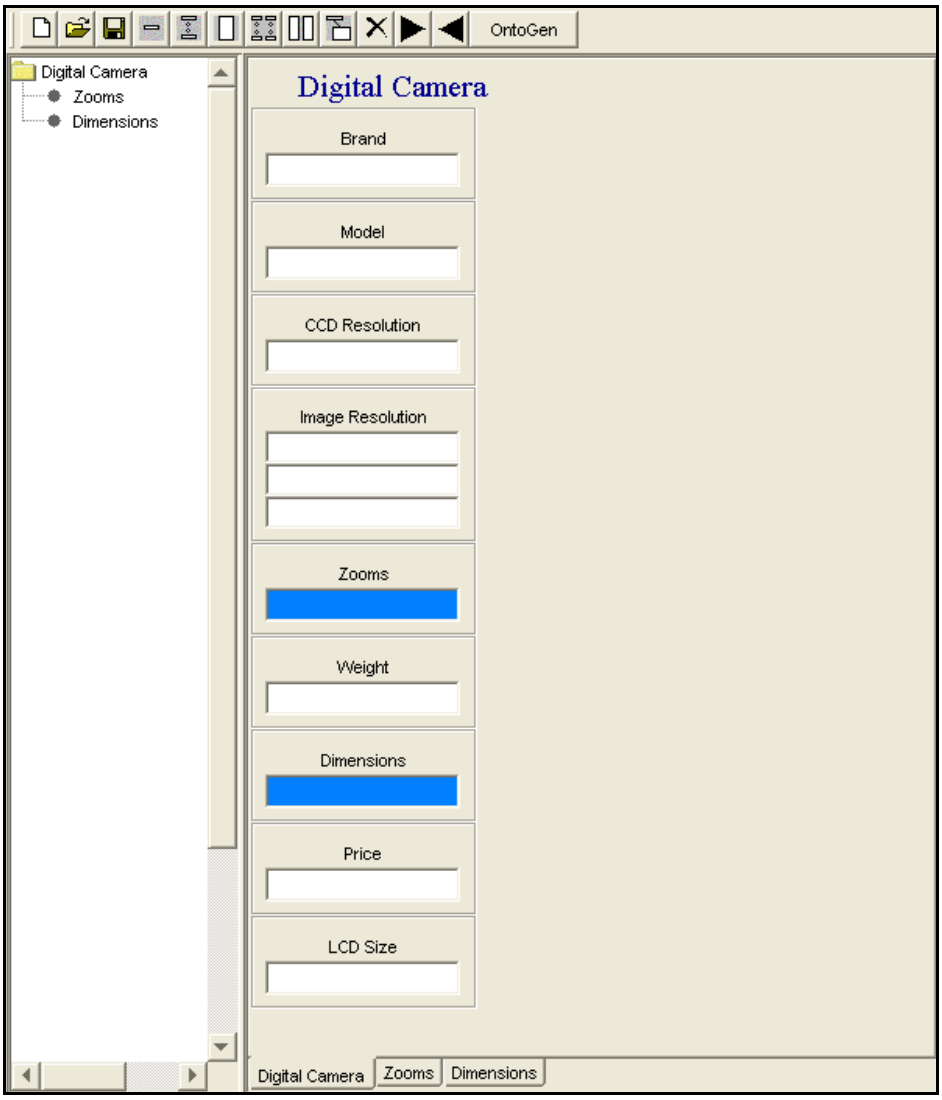


Figure 16: Application-Dependent Forms for the Digital Camera Application

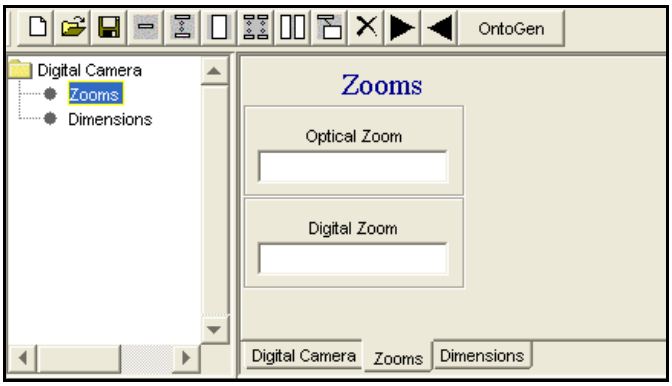


Figure 17(a): Nested Form *Zooms* in the Base Form *Digital Camera*

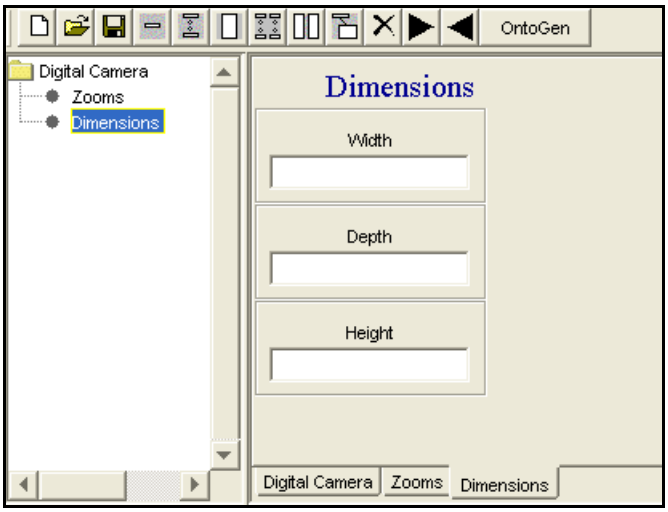


Figure 17(b): Nested Form *Dimensions* in the Base Form *Digital Camera*

4.2 Preparing Sample HTML Documents

After users define application-dependent forms to describe the data of their interest, such as the *Digital Camera* forms in Figures 16-17, they can load a pre-collected sample HTML page in the left pane of the interface to fill in forms with sample data values on the page. To fill in data from the sample page on the left to a value field in a form on the right, users highlight the desired text value on the HTML document using the mouse, click the destination value field in the form and click  to fill in the field with the selected text value from the HTML page. During the fill-in, the sample data on the

HTML page is marked with a highlighted background for later recognition by users, and the text data in the HTML content is tagged with the column label from the form. To delete a selected value from a value field, users click on the value field in the form and then click  . When the value is deleted from the form, the corresponding text value is de-marked and de-tagged on the HTML page. Figure 18 shows a sample marked HTML page with the filled-in forms for Digital Camera application.

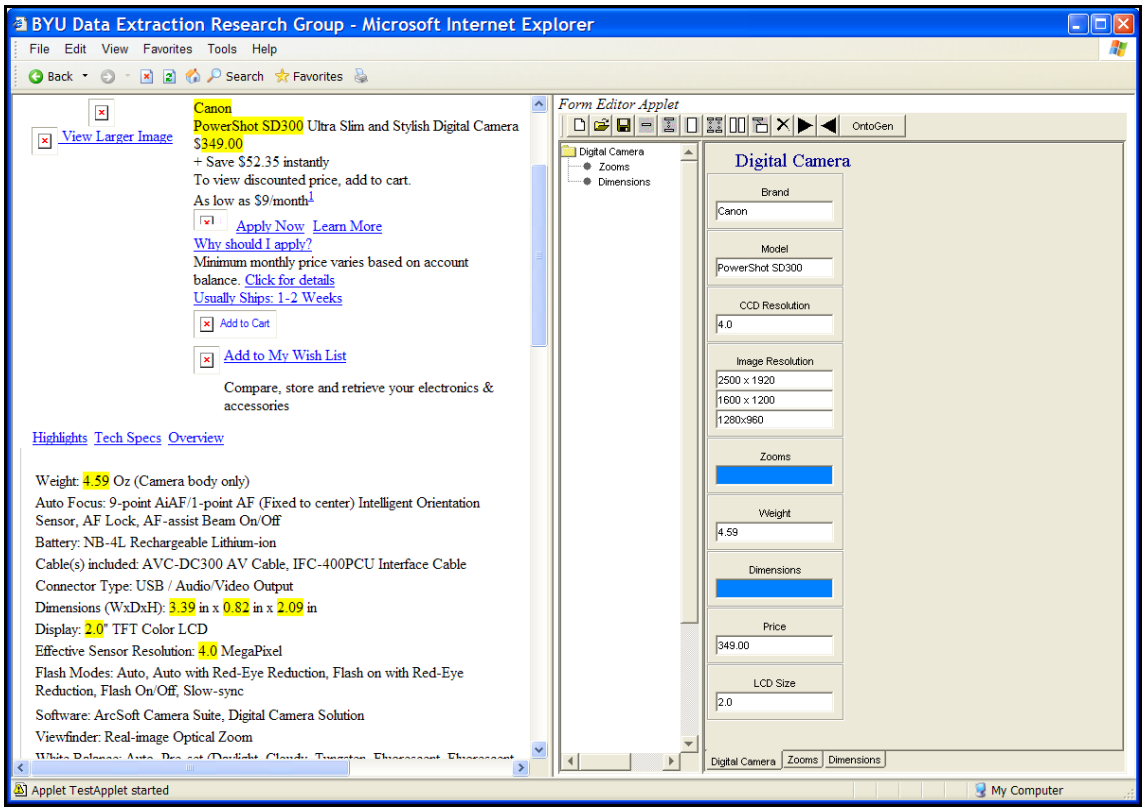


Figure 18: Training Web Document Preparation

After marking the sample data on one HTML page, users save the marked page to a directory and repeat the operation for each pre-collected sample page. Users may edit existing forms and previously marked HTML pages through the user interface. When users open an existing form and load a previously marked sample page, the previously marked values on the sample page will automatically fill the forms for easy recognition

and further editing. After users finish creating forms and marking desired data on sample pages, the forms, along with all marked HTML pages, will serve as training information to the OntoByE system for further ontology-generation purposes.

5 Data-Extraction Ontology Generation

The back-end ontology generator in OntoByE takes the user-defined forms and marked HTML pages as inputs and generates data-extraction ontologies with the assistance from the pre-existing data frame library.

The OntoByE back-end ontology generator is invoked via the ontology generation icon on the form editor toolbar as in Figure 7 of Chapter 4. Upon initial invocation for a new application, OntoByE launches an ontology generator configuration wizard, as depicted in Figure 19, which prompts the user to specify the directory in which the marked HTML pages reside, along with the Data Frame Library URL.

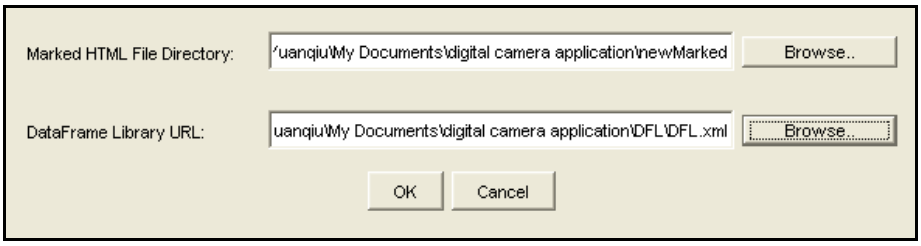


Figure 19: The Configuration Window for the Ontology Generator

The file path and URL are saved and associated with the application for subsequent operations. Both attributes may be changed by manually completing the same configuration wizard. The following sections will describe how OntoByE generates data-extraction ontologies based on the user-defined forms and the marked HTML sample pages.

5.1 Constructing Object and Relationship Sets and Constraints

The form analyzer of the ontology generator constructs object and relationship sets and constraints for a data-extraction ontology after users define the application-

dependent forms. Since each form and each column defined in the form represents a concept or an object set in the data-extraction ontology, the analyzer constructs the names of object sets from user-specified form titles and column labels.

After supplying a title to the base form of an application, the analyzer constructs a primary object set, named after the base form title. As elements are added to the forms, the analyzer constructs object set names for form elements by taking user-specified column labels. The analyzer constructs relationships between the form element object sets and the form title object set, and adds participation constraints over these object sets and their relationships. The following example shows how our system constructs object and relationship sets and constraints for a base form *BaseForm*. Using the sample forms from Figure 13 in Chapter 4, Figure 20 demonstrates the method by which OntoByE constructs the object and relationship sets and constraints.

BaseForm [0:1] A [1:*]
BaseForm [0:3] B [1:*]
BaseForm [0:*] C [1:*]

BaseForm [0:3] D1 [1:*] D2 [1:*] D3 [1:*]
BaseForm [0:*] E1 [1:*] E2 [1:*] E3 [1:*]

Figure 20: Object and Relationship Sets and Constraints for the Base Form

In Figure 20, *A, B, C, D1, D2, D3, E1, E2* and *E3* are column labels for patterns A, B, C, D and E respectively. The analyzer generates binary relationship sets for the elements of single-column patterns A, B and C and n-ary relationship sets for elements of multiple-column patterns D and E. For the object set *BaseForm* representing the current form, the minimum participation constraint is 0 by default and the maximum constraint is

1, 3 (a user-supplied value) or * (an unlimited number). The participation constraints on the object sets in form elements are always set to [1:*].

As described in section 4.1.4, each column in a form element contains either a nested form or data value fields. The columns associated with nested forms represent non-lexical object sets, while the columns containing data value fields represent lexical objects sets. Accordingly, the analyzer constructs non-lexical object sets for columns with nested forms and lexical object sets for columns containing data value fields. For example, if a user defines nested forms as shown in Figure 14, the analyzer constructs object and relationship sets and constraints, shown in Figure 21, where *BaseForm* is the primary object set, *A, B* and *C* are non-lexical object sets and *A1, A2, B1, C1* and *C2* are lexical object sets. The analyzer constructs participation constraints for non-lexical object sets in the same way as lexical object sets.

BaseForm [0:1] A [1:*]
BaseForm [0:3] B [1:*]
BaseForm [0:*] C [1:*]

A [0:1] A1 [1:*]
A [0:2] A2 [1:*]
B [0:*] B1 [1:*]
C [0:2] C1 [1:*] C2 [1:*]

Figure 21: Object and Relationship Sets and Constraints for Nested Forms in the Base Form

Detailed in Figure 22 are the object and relationship sets and contraints that the form analyzer constructed for the Digital Camera application from Figures 16 and 17 in Chapter 4.

Digital Camera [0:1] Brand [1:*]
Digital Camera [0:1] Model [1:*]
Digital Camera [0:1] CCD Resolution [1:*]
Digital Camera [0:3] Image Resolution [1:*]
Digital Camera [0:1] Zooms [1:*]
Digital Camera [0:1] Weight [1:*]
Digital Camera [0:1] Dimensions [1:*]
Digital Camera [0:1] Price [1:*]
Digital Camera [0:1] LCD Size [1:*]

Zooms [0:1] Optical Zoom [1:*]
Zooms [0:1] Digital Zoom [1:*]

Dimensions [0:1] Width [1:*]
Dimensions [0:1] Depth [1:*]
Dimensions [0:1] Height [1:*]

Figure 22: Object and Relationship Sets and Constraints from the Digital Camera Application Forms

5.2 Constructing Data Frames for Object Sets

The back-end ontology generator in OntoByE consists of four components.

- The *Context Phrase Locator* extracts context phrases for user marked data from sample pages
- The *Data Frame Matcher* searches for matching data frames from the data frame library for user marked data,
- The *Keyword and Context Expression Recognizer* identifies possible keyword and context expressions for user-marked data from the context phrases.

- The *Data Frame Editor* presents the matching data frames, along with recognized keywords and context expressions from context phrases, if any, and allows user to select, edit or create data frames for object sets.

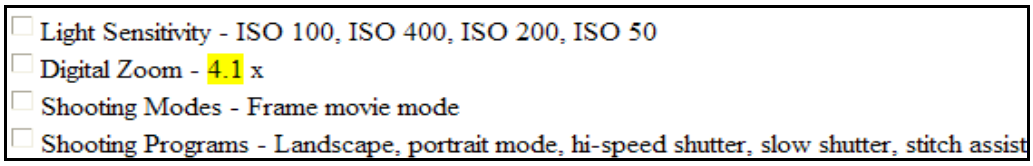
The following sections describe how each component of the ontology generator contributes to data-extraction ontology generation.

5.2.1 Context Phrase Locator

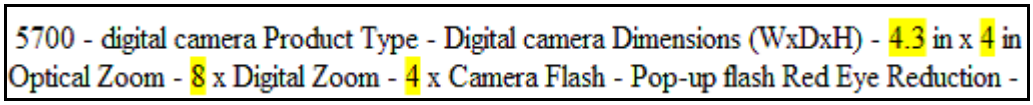
A context phrase of the user-desired data consists of user-marked data values and their surrounding characters from sample HTML pages. The context phrase may contain keyword and context expressions from the marked data that help to better describe the desired data and distinguish the desired data from other data with the same value expressions. The context phrase locator is designed for extracting context phrases for user-marked data from sample pages. The context phrases will be used to search data frames from the data frame library and to facilitate the recognition of context and keyword expressions in forthcoming ontology generation processes.

The context phrase locator takes a list of object set names from the user-defined forms and scans each sample HTML page in a user-specified directory. For each object set on each page, the locator identifies the marked object values and a certain number, or a padding length, of adjacent text characters attached to the beginning and the end of the marked data. The locator uses the marked data and adjacent characters to construct a context phrase for the object set. In some instances, two marked data may be so close together in the text content of sample pages that one marked value may appear in the context phrase of the other. In such a case, the locator will truncate the related context phrase from the beginning or the end to ensure that the context phrase of one marked data

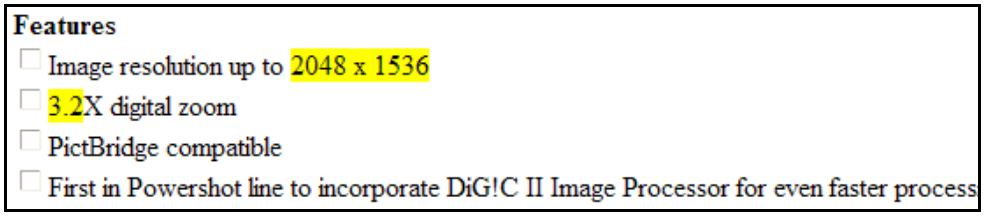
will not contain the other marked data. The truncation improves the accuracy of locating meaningful context phrases for the marked data. Figure 23 demonstrates the marked data for the object set *Digital Zoom* on three HTML sample pages along with their context phrases as extracted by the context phrase locator in our Digital Camera application. In this example, the context phrase locator padding length is set to 40. As observed in Figure 23, the context phrases for the marked values *4* and *3.2* in sample 2 and sample 3 have been truncated to exclude other marked values from the same pages.



Context Phrase 1:
400, ISO 200, ISO 50 Digital Zoom - **4.1** x Shooting Modes - Frame movie mode



Context Phrase 2:
x Digital Zoom - **4** x Camera Flash - Pop-up flash Red Eye R



Context Phrase 3:
3.2X digital zoom PictBridge compatibl

Figure 23: User-marked Data and Their Context Phrases for the Object Set *Digital Zoom* in the Digital Camera Application

When a context phrase is extracted for marked data, the locator saves the start and end positions of the marked value in the context phrase for future data frame matching.

After scanning each sample page, the locator constructs a list of context phrases for each object set.

5.2.2 Data Frame Matcher

The data frame matcher leverages prior knowledge in the data frame library by searching for appropriate existing data frames that can recognize the user-marked data for new domains of interest.

To search for appropriate data frames for each object set, the matcher applies each data frame in the library to the user-marked data in the context phrases of the object set constructed from different sample pages by the context phrase locator. If a data frame recognizes one or more user-marked data for the object set, the matcher records the data frame as a data frame candidate for the object set. The matcher constructs and returns for users' examination a list of data frame candidates for each object set based on the search results. The list of candidates is ranked for each object set to indicate the degree to which the data frame candidates matched the user-marked data.

For each object set, the matcher ranks the data frame candidates using the following heuristic method:

- Candidates are initially ranked by the number of user-marked data that are recognized by the data frame. The more user-marked data recognized, the higher the possibility that the data frame is a better candidate.

- If two data frames recognize the same number of user-marked data, the matcher executes a secondary search for their keywords and context expressions, if any, for the user-marked data from their context phrases. If

one data frame recognizes its keywords and/or context expressions for the user-marked data from their context phrases while the other data frame does not, the matcher ranks the former data frame higher than the latter, since keywords and context usually indicate the presence of user desired data. For example, in the Digital Camera application, both data frames *Price* and *RealNumber* match all user-marked price values from different sample pages. But *Price* also recognizes its keyword, **(price|Price)**, and its context expression, **$(\s)?**, in the context phrases of user-marked prices, while *RealNumber* does not. Therefore, the matcher ranks *Price* higher than *RealNumber* in the data frame candidate list for the object set Price.

- If two data frames recognize the same number of user-marked data but neither keywords nor context expressions, the matcher ranks the more specific data frame higher than the more general one. We define one data frame as being more specific than the other if the data instances recognized by one data frame are a subset of those recognized by the other. For example, in the Digital Camera application, both data frames *Integer* (0, positive and negative of any number) and *SingleDigit* (from 0 to 9) recognize all user-marked *Optical Zoom* values and neither of the data frames have keyword or context expressions. Since the data instances for *SingleDigit* are a subset of those for *Integer*, the matcher ranks *SingleDigit* higher than *Integer* in the candidate list.

The matcher displays the ranked list of data frame candidates for each object set through the selection window shown in Figure 24. Each object set has the names of its

data frame candidates contained in a drop-down list. For each object set, users can browse data frame expressions for all candidates through a data frame editor (refer to section 5.2.4) by clicking on the "View" button next to the drop-down list.

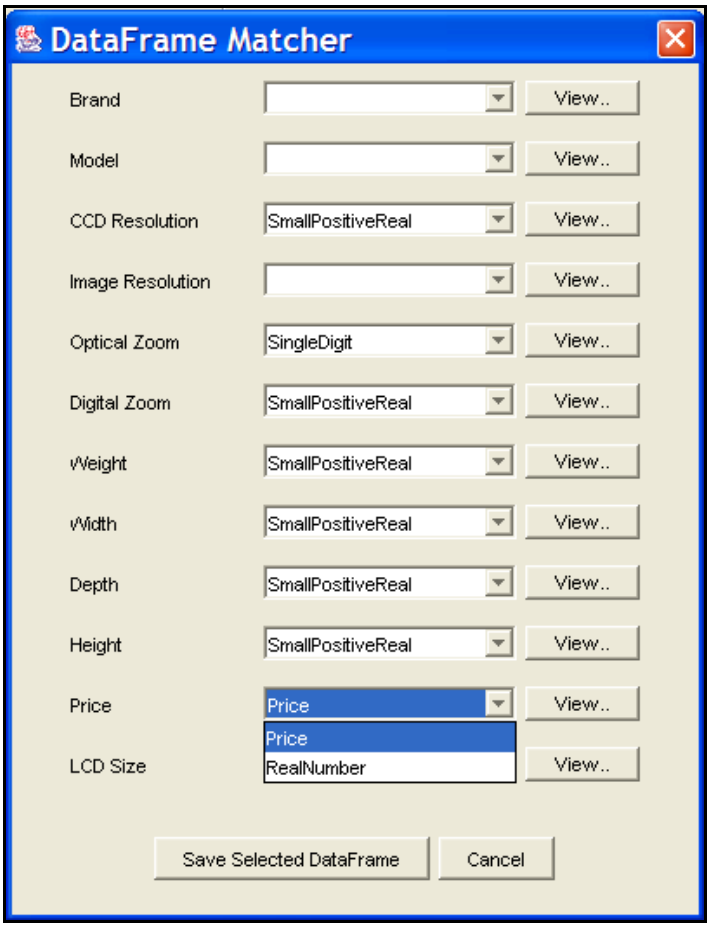


Figure 24: Matching Data Frames for the Digital Camera Application

5.2.3 Keyword and Context Expression Recognizer

The keyword and context expression recognizer is designed to identify the possible keyword and context expressions for user-marked data from sample pages. The identified keywords and context expressions can help users to construct new data frames or expand the existing data frames for object sets in data-extraction ontologies for the domains of interest. As described in Chapter 2, the context expressions are the characters

immediately adjacent to the beginning or the end of the user desired data, while the keywords are common strings that appear near, but not always immediately adjacent to, the data of interest.

5.2.3.1 Constructing Context Expressions

A context expression may consist of a left context expression and a right context expression. To construct the left and right context expressions, we define, for each context phrase from sample pages, the *left context phrase* as the characters from the beginning of the context phrase to the character immediately adjacent to the left of the user-marked data and the *right context phrase* as characters from the character immediately adjacent to the right of the user-marked data to the end of the context phrase.

To identify the left context expression for user-marked data, the recognizer scans the list of left context phrases character by character from right to left in a case-insensitive manner. The recognizer maintains an index pointer for each context phrase. At the beginning, the indexes are pointed to the right end of left context phrases. Then, the recognizer compares the characters at the index positions of all context phrases. If the characters at the index positions of all context phrases are case-insensitively identical, the recognizer will record the character into the left context expression of the marked data. If the character appears in both upper and lower case across the context phrases, the recognizer will record both upper and lower case in a logical OR regular expression relation, such as ***(x|X)***. If the common character is whitespace, the recognizer will record ***\s*** in the context expression. If, at the comparison, the whitespace appears at the index positions for some, but not all, context phrases, the recognizer will move the index pointers from the whitespaces to the next characters in those context phrases and record

(\s)? in the context expression, thereby indicating the whitespace was skipped during comparison. The comparison and recording will continue all the way through the left context phrases from the right to the left until the non-whitespace characters from all context phrases at the index positions are not case-insensitively identical or the characters in any left context phrase have been exhausted. The recognizer will return what it identifies, if not only white-spaces, as the left context expression for user-marked data. The recognizer constructs the right context expressions in a similar fashion, the only difference being that scanning goes from left to right in the right context phrases.

Figure 25 shows an example of context phrases (shown previsouly in Figure 23) for the object set *Digital Zoom* from sample pages and the left and right context expressions constructed from these context phrases for user-marked values (4.1, 4 and 3.2). Note that the recognizer identifies nothing in common before the marked data and an optional white space, ***(\s)?***, followed by a case-insensitive character x, ***(x|X)***, after the marked data. Thus, the recognizer leaves the left context expression empty while it constructs the right context expression as ***(\s)?(x|X)***.

Context Phrase 1:
400, ISO 200, ISO 50 Digital Zoom - **4.1** x Shooting Modes - Frame movie mode

Context Phrase 2:
x Digital Zoom - **4** x Camera Flash - Pop-up flash Red Eye R

Context Phrase 3:
3.2X digital zoom PictBridge compatibl

Left Context Expression:
Right Context Expression: `(\s)?(x|X)`
Keywords: `Digital\sZoom|digital\szoom`

Figure 25: Recognizing Keywords and Context Expressions for the Object Set *Digital Zoom* from Its Context Phrases

5.2.3.2 Constructing Keywords

Keywords or keyword expressions for the user-marked data are defined as common strings in context phrases that are neither left nor right context expressions. They indicate the presence of the desired data during the ontology-based data-extraction.

To identify the keywords, the recognizer tokenizes the left and the right context phrases for the marked data with common delimiters, such as colon, semi-colon, white space, comma and so on. The recognizer compares the tokenized strings in a case-insensitive manner and records distinguished strings, along with the number of times they appear in the context phrases. If a common string appears more than once in the same context phrase, such as the string *flash* in context phrase 2 shown in Figure 25, the recognizer will count its appearance only once for that context phrase. For common strings that appear in more than one context phrase, the recognizer eliminates the common stop-words (such as *a, an, the* and so on), single letters (such as a, b …, x, y, z) and common symbols (such as hyphen '-') and saves the remaining common strings as keyword candidates. The recognizer scans the list of keyword candidates to identify the longest possible common strings that the keywords form in the context phrases. For example, shown in Figure 25, though the recognizer identifies both ***Digital*** and ***Zoom*** as individual keyword candidates from the context phrases, ***Digital Zoom*** is a longer common string from the context phrases that subsumes them. In such a case, the recognizer adds the longest possible common string formed by the individual candidates as a new candidate, such as ***Digital Zoom***, to the list while all subsumed individual candidates, such as ***Digital*** and ***Zoom***, will be discarded. Finally, if a keyword candidate appears case-sensitively, such as ***Digital Zoom*** and ***digital zoom***, the recognizer will

record the candidate through a logical OR relation in the keyword expression, such as ***(Digital\sZoom|digital\szoom)***, where the white space is recorded as **\s** in the expression.

After constructing context and keyword expressions, the recognizer presents the expressions, if any, in regular expressions in a data frame editor.

5.2.4 Data Frame Editor

The data frame editor was originally designed by the BYU DEG group as a tool for presenting and editing existing data frames. For this thesis, the original data frame editor has been expanded to show not only the existing data frames that match user-marked data, but also the keywords and context expressions recognized from the context phrases on the sample pages. In addition, the expanded editor allows users to construct new data frames based on both existing data frames and the recognized keywords and context expressions. Figure 26 (a) illustrates the newly expanded data frame editor, containing the original "Data Frames" panel, an expanded "Suggested Keyword and Context Expressions" panel and a set of data frame editing operation buttons such as "New DataFrame" and "Save Changes".

The "Data Frames" panel displays matching data frames identified from the data frame library by the data frame matcher. For example, in Figure 26 (a), the "Data Frames" panel displays the matching data frames *RealNumber* and *SmallPositiveReal* for the object set *Digital Zoom*. The "Suggested Keyword and Context Expressions" panel displays any keywords and left and right context expressions recognized by the keyword and context expression recognizer. As illustrated in Figure 26 (a), the "Suggested Keyword and Context Expressions" panel displays the right context expression **(\s)?(x|X)** and keywords **Digital\sZoom|digital\szoom** for the object set *Digital Zoom*.

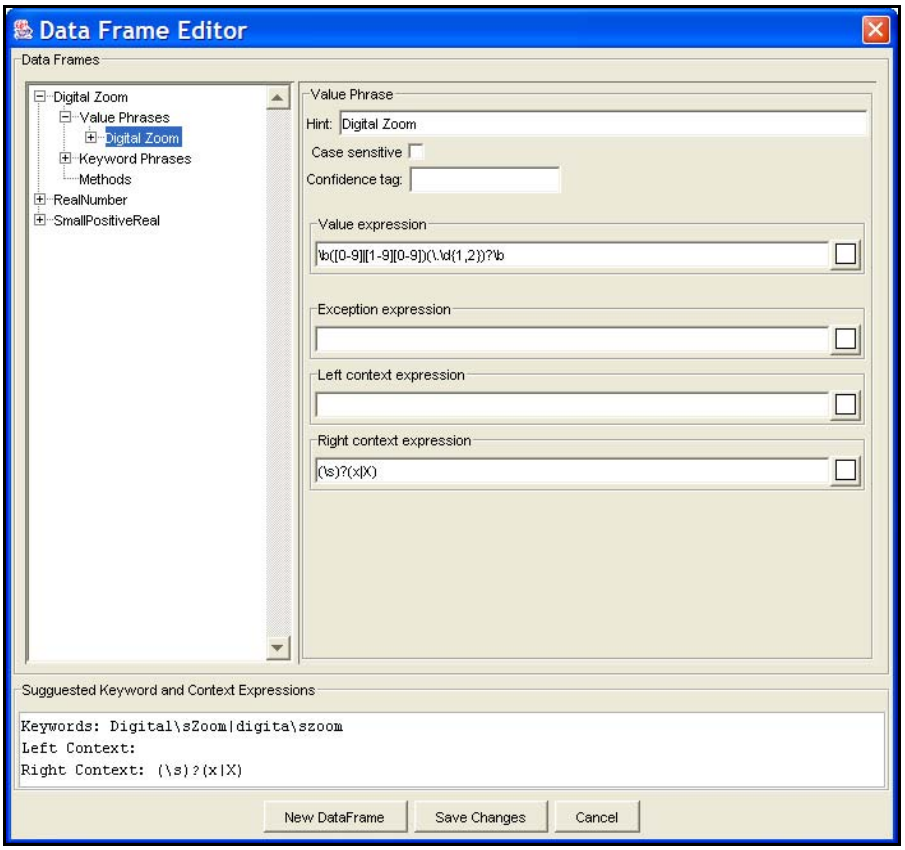


Figure 26(a): Constructing Value Expressions and Context Expressions for a New Data Frame *Digital Zoom*

As described in Section 5.2.2, the data frame matcher ranks the matching data frames for an object set in some heuristics. Through the data frame editor, if any matching data frames exist and the highest-ranked data frame does not contain its own keyword and context expressions, OntoByE automatically constructs a new data frame named after the object set by copying the value phrases from the highest-ranked data frame and populating the new data frame with the recognized keywords and context expressions.

Shown in Figures 26(a) and 26(b) together, OntoByE has constructed the new data frame *Digital Zoom* for the object set *Digital Zoom* in the Digital Camera application. This new data frame contains the value expression **\b([0-9]|[1-9][0-**

9])(\.\d{1,2})?\b as copied from the existing data frame *SmallPositiveReal*, and the keyword **Digital\sZoom|digital\szoom** and the right context expression **(\s)?(x|X)** populated from those recognized in the context phrases.

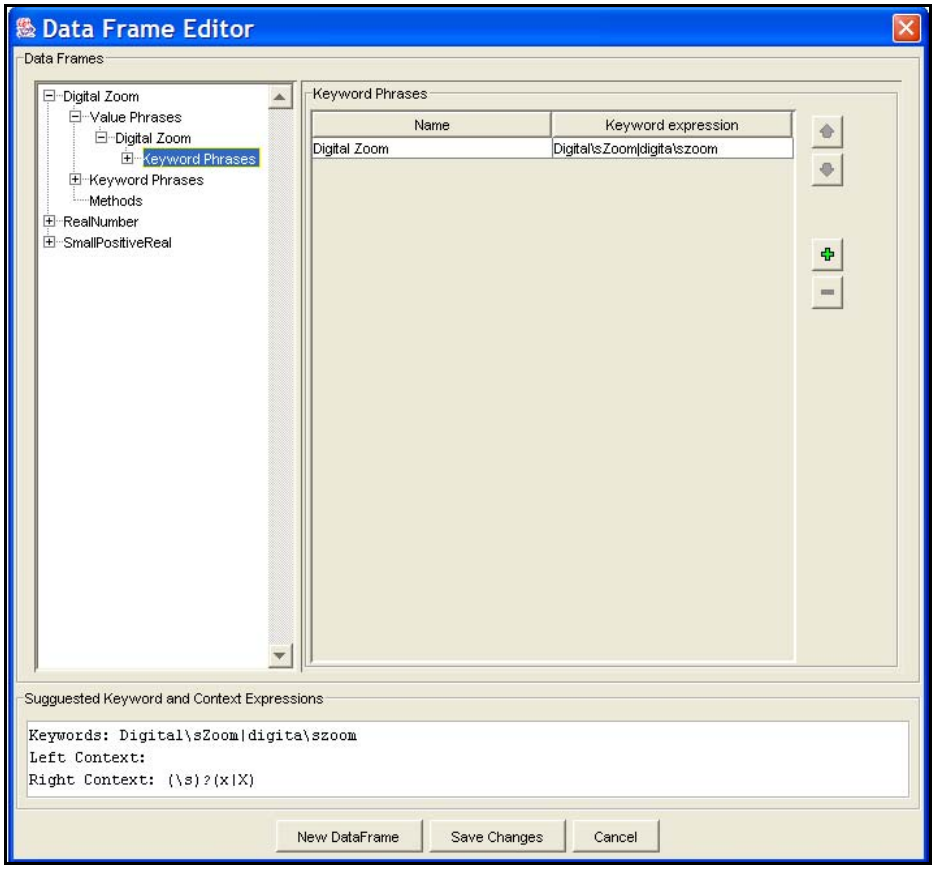


Figure 26(b): Constructing Keywords for a New Data Frame *Digital Zoom*

OntoByE allows users to further modify the new data frame manually based on their knowledge of regular expressions. In the event that the highest-ranked matching data frame has its own keyword and context expressions, OntoByE does not automatically construct a new data frame, but leaves it to the users' discretion to either expand the existing data frame with the recognized keywords and context expressions, or construct a new data frame manually by clicking on the data frame editor "New DataFrame" button. For example, in the sample Digital Camera application, the data

frame matcher matches the existing data frame *Price* with user-marked data for the object set *Price* and the data frame *Price* has its own keywords and context expressions. Therefore, users may decide to expand the data frame *Price* with the recognized keywords and context expressions, if the expressions are different from those in the data frame. If there are no matching data frames in the library for an object set, OntoByE constructs a new data frame, named after the object set, with an empty value phrase and any recognized keywords and context expressions. Users are then required to write regular expressions on their own to describe the marked-data.

After users expand an existing data frame or construct a new data frame for an object set and save the changes through the data frame editor, OntoByE will save the data frame as the user-selected data frame for the corresponding object set in the data frame matcher window as shown in Figure 24. Users then invoke a new data frame editor and repeat the same data frame editing process for each object set. After users finish editing data frames for all object sets, they click the "Save Selected DataFrame" button in the wizard shown in Figure 24 to resume the ontology generation process.

5.3 Generating Data-Extraction Ontologies

As described in the system overview in Chapter 2, OntoByE generates data-extraction ontologies for the domains of interest after users create application-dependent forms, collect HTML sample pages, mark the data of interest from the sample pages, and edit data frames for object sets by interacting with the back-end ontology generator in OntoByE.

In the running Digital Camera sample application, users created Digital Camera forms (shown in Figure 16, 17(a) and 17(b) of Chapter 4), and collected and marked

HTML sample pages (illustrated in Figure 18 of Chapter 4). Then, OntoByE constructed *Object and Relationship Sets and Constraints* (shown in Figure 22 of Section 5.1) from the user-defined forms and constructed *Data Frames* with user-interaction for object sets (illustrated in Figure 26(a) and 26(b) of Section 5.2) based on the sample pages and the exisiting data frame library. Finally, OntoByE combines *Object and Relationship Sets and Constraints* and *Data Frames* for object sets to output a data-extraction ontology for the Digital Camera application.

Figure 27 shows a partial data-extraction ontology generated by OntoByE for the sample Digital Camera application based on the examples depicted in previous sections The partial ontology contains object sets (such as a primary object set *Digital Camera*, a non-lexical object set *Zooms* and a lexical object set *Digital Zoom*), their relationships and constraints, and the data frame constructed for the lexical object set *Digital Zoom.*

```
<ObjectSet name="Digital Camera" primary="true" id="osmx1" />
<ObjectSet name="Zooms" lexical="false" id="osmx5" />
<ObjectSet name="Digital Zoom" lexical="true" id="osxm11">
- <DataFrame>
  - <InternalRepresentation>
      <DataType typeName="java:java.lang.String" />
    </InternalRepresentation>
  - <ValuePhrase hint="Digital Zoom" caseSensitive="false">
    - <ValueExpression>
        <ExpressionText>\b([0-9]|[1-9][0-9])(\.\d{1,2})?\b</ExpressionText>
      </ValueExpression>
    - <LeftContextExpression>
        <ExpressionText />
      </LeftContextExpression>
    - <RightContextExpression>
        <ExpressionText>(\s)?(x|X)</ExpressionText>
      </RightContextExpression>
    - <KeywordPhrase hint="Digital Zoom">
      - <KeywordExpression>
          <ExpressionText>(Digital\sZoom|digital\szoom)</ExpressionText>
        </KeywordExpression>
      </KeywordPhrase>
    </ValuePhrase>
  </DataFrame>
</ObjectSet>
<RelationshipSet id="osmx29">
  <Name content="has" />
- <RelSetConnection objectSet="osmx1" id="osmx27">
    <ParticipationConstraint content="0:1" />
  </RelSetConnection>
- <RelSetConnection objectSet="osmx5" id="osmx28">
    <ParticipationConstraint content="1:*" />
  </RelSetConnection>
</RelationshipSet>
<RelationshipSet id="osmx41">
  <Name content="has" />
- <RelSetConnection objectSet="osmx5" id="osmx39">
    <ParticipationConstraint content="0:1" />
  </RelSetConnection>
- <RelSetConnection objectSet="osmx11" id="osmx40">
    <ParticipationConstraint content="1:*" />
  </RelSetConnection>
</RelationshipSet>
```

Figure 27: The Partial Data-Extraction Ontology for the Digital Camera Application

6 Experimental Observations and Analyses

For the purposes of this thesis, we tested the OntoByE system on two field applications: digital camera advertisements and apartment rental advertisements. The system, however, is not limited to the two applications on which we experimented. It will work with other applications with the assistance of an initial data frame library containing some common data frames and a small set of user-collected sample HTML pages from the domains of interest. In this chapter we describe our experimental structure and report our observations. We also discuss the strengths and weaknesses of the OntoByE system.

6.1 Preparation

The prerequisites for generating a data-extraction ontology through OntoByE include the following (see Chapter 3):

- Experts construct an initial data frame library, which contains some common data frames.
- Users create forms to describe the data of interest in the domain.
- Users collect and mark a small set of sample HTML pages from the domain.

For our field tests, we began by constructing a small initial data frame library with common data frames that could be applied across different applications. The library in our experiments contained the following common data frames: *Integer* (any integer

value), *SmallPositiveInteger* (from 1 to 99), *SingleDigit* (from 0 to 9), *RealNumber* (any real value), *SmallPositiveReal* (from 0.01 to 99.99), *Date*, *Email*, *PhoneNumber*, and *Price*.

Then for each application we created forms to assist in describing the data of interest within the application domain. After creating the forms, we collected a small set of sample pages from different web sites and marked data of interest on the sample pages by filling them into the forms. The *Digital Camera* application forms and sample marked HTML pages have been illustrated throughout previous chapters. For the Apartment Rental application, by comparison, we used a set of information described in a previous ontology that was hand-written by a member of our Data Extraction Research Group for this domain. Using this information, we created a base form titled *Apartment Rental*, as shown in Figure 28. The form contains 12 elements: *Apt Type*, *Bedroom Number*, *Bathroom Number*, *Gender Requirement*, *Date Available*, *Monthly Rate*, *Deposit*, *Features*, *Contact Phone*, *Contact Person*, *Furnished Condition* and *Utility*, where all elements contain a single-value field except *Features*, which contains an unlimited-value field. We collected and marked the desired data values on a few HTML pages containing apartment rental advertisements. Figure 29 shows a marked sample page for the Apartment Rental application.

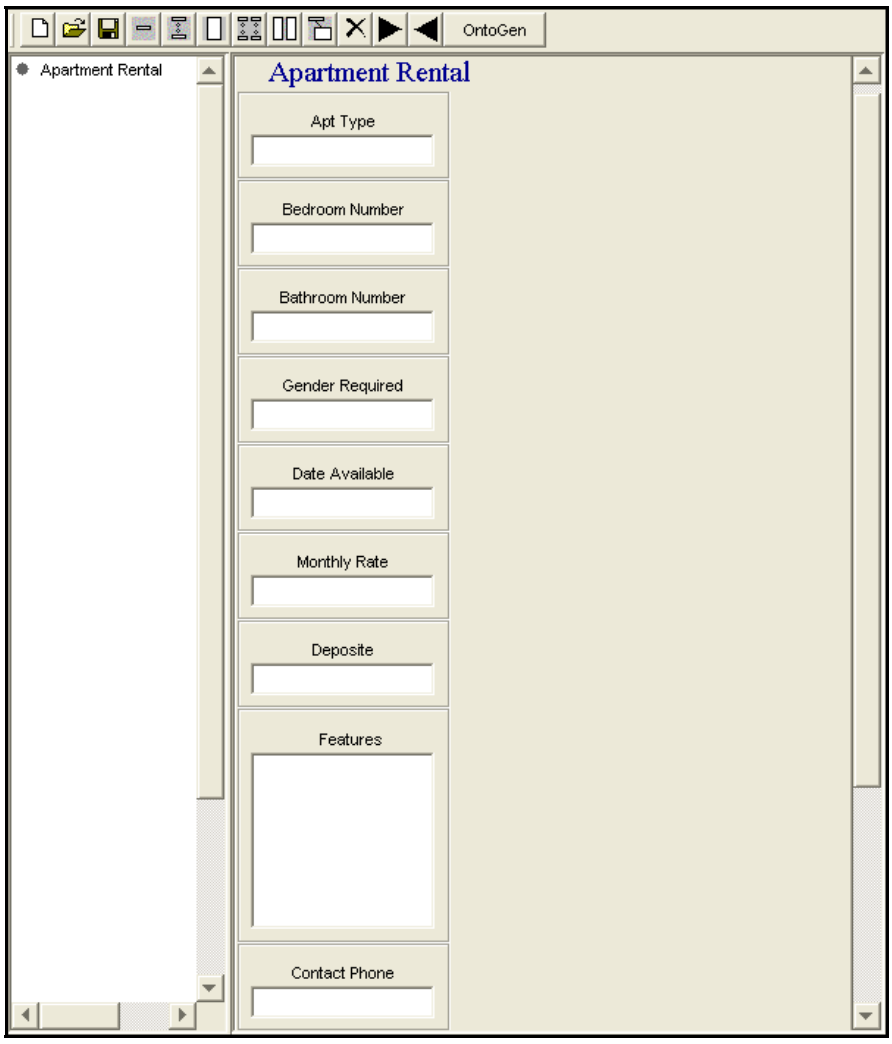


Figure 28: Forms for the Apartment Rental Application

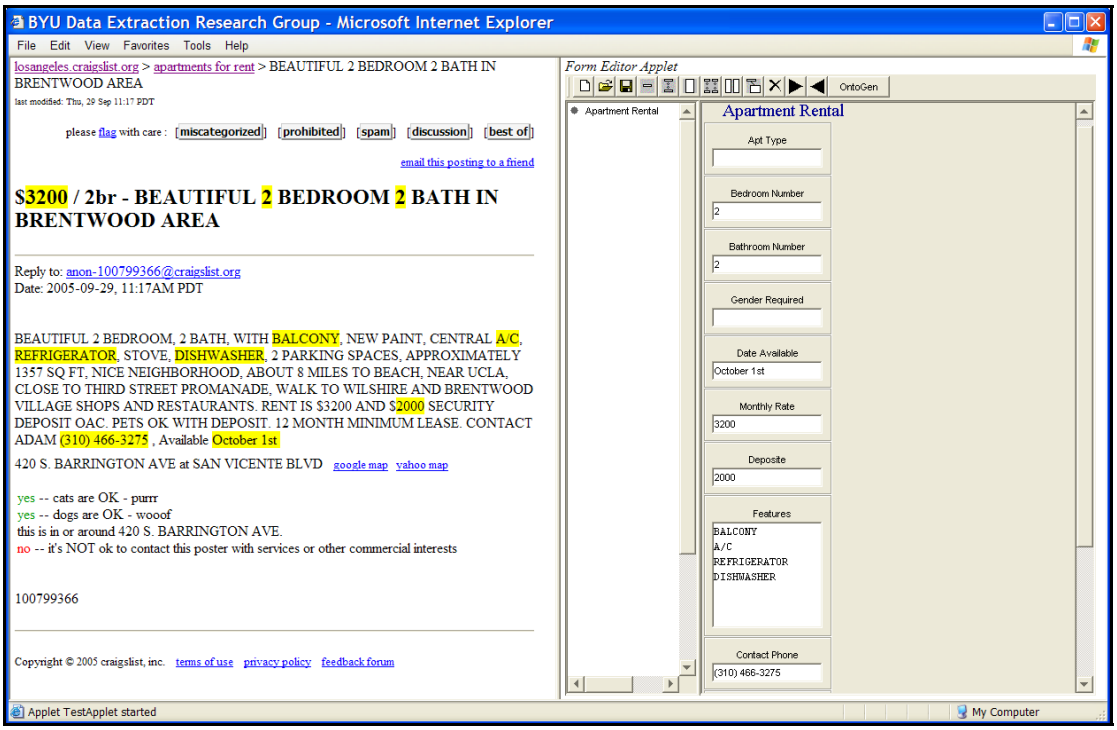


Figure 29: A Sample Marked HTML Page for the Apartment Rental Application

6.2 Results and Observations

In the following sections, we observe the ontology-generation process in OntoByE and analyze the experimental results for both the Digital Camera and Apartment Rental applications.

A data-extraction ontology consists of two major components (see Chapter 2): (1) object and relationship sets together with related constraints, and (2) data frames for object sets. OntoByE constructs object sets, relationship sets, and associated constraints by translating user-defined forms to corresponding pre-defined sets and constraints, as described in Section 5.1. The translation is consistent and straightforward. Therefore, in the analyses of onotology generation by OntoByE, the most challenging part is to evaluate the performance of the construction of data frames for object sets. Since OntoByE interacts with users to construct data frames with the assistance of both prior

knowledge and user-marked sample pages, the evaluation of OntoByE performance is inevitably affected by the following variables:

1. The number and the representativeness of user-collected sample HTML pages for a specific domain,

2. The quality and the scope of prior knowledge, such as lexicons and regular expressions, from the intial data frame library, and

3. The user's knowledge of writing regular rexpressions and their common knowledge about the domain.

The evaluation of these variables is subjective. We therefore make observations of the experimental results of OntoByE only with respect to the initial data frame library we have chosen and the sample pages we found for the two applications. Based on these observations, we discuss the major strengths and limitations of OntoByE in generating data-extraction ontologies.

6.2.1 Digital Camera Advertisement

For the Digital Camera application, OntoByE constructs object and relationship sets and constraints as shown in Figure 22 of Section 5.1.

Experimental results for the Digital Camera application show that OntoByE works well when searching for the most appropriate existing data frames for object sets with numeric values based on limited prior knowledge (i.e. a small initial data frame library). In Table 1, OntoByE matches the most appropriate existing data frames for 9 object sets (*CCD Resolution, Optical Zoom, Digital Zoom, Width, Depth, Height, Price*

and *LCD Size*) from a total possible 12 object sets. Data frame matches include such examples as *SmallPositiveReal* for *CCD Resolution* and *SingleDigit* for *Optical Zoom*.

OntoByE, however, did not find appropriate data frames for the three object sets *Brand*, *Model*, and *Image Resolution*. This failure was expected, since both *Brand* and *Model* rely on application-dependent lexicons, while *Image Resolution* requires regular expressions that were not in the intial library. To construct the data frames for these three object sets in the digital camera ontology, users would need to collect lexicons for *Brand* (such as "Nikon", "Canon", or "Kodak") and *Model* (such as "PowerShot A50" or "Coolpix 5700"), and also write their own regular expressions for *Image Resolution* (such as **\d{3,4}(\s)?(x|X)(\s)?\d{3,4}**). Although sometimes it is possible to generate regular expressions for user-marked sample data, the topic is beyond the scope of this thesis.

Object Set	*Matching Data Frame*	*Left Context Expression*	*Right Context Expression*	*Keywords*
Brand	*	-	-	-
Model	*	-	-	-
CCD Resolution	SmallPositiveReal	-	\s(Megapixel |MegaPixel)	-
Image Resolution	*	-	-	-
Optical Zoom	SingleDigit	-	(\s)?(x|X)	(Optical\sZoom |optical\szoom)
Digital Zoom	SmallPositiveReal	-	(\s)?(x|X)	(Digital\sZoom |digital\szoom)
Weight	SmallPositiveReal	-	\s(oz|Oz)	(Weight|weight)
Width	SmallPositiveReal	-	\s(in)	(Width|width)
Depth	SmallPositiveReal	-	\s(in)	(Depth|depth)
Height	SmallPositiveReal	-	\s(in)	(Height|height)
Price	Price	($)(\s)?	-	(Price|price)
LCD Size	SmallPositiveReal	-	(")	LCD

Note: * Application-dependent Lexicons not in Initial Data Frame Library
- Not Available from Sample Pages

Table 1: Experimental Results of Constructing Data Frames for the Digital Camera Application

The experimental results in Table 1 also demonstrate that OntoByE was successfully able to identify the keywords and context expressions, if any were present on sample pages, from the context phrases of user-marked data. In Table 1, OntoByE properly identifies possible keywords and/or context expressions for 9 object sets (CCD Resolution, Optical Zoom, *Digital Zoom*, *Width*, *Depth*, *Height*, *Price* and *LCD Size*), which rely on neither application-dependent lexicons nor regular expressions other than those existing in our initial data frame library.

Another observation we made during our experiments is that sample pages from different web sites help improve the accuracy of keyword and context expression recognition for user-desired data, because pages with different layouts help OntoByE eliminate common strings that are not actually keywords or context for user-marked data. We observed that OntoByE sometimes recognized more keywords and context expressions than it should for user-marked data where sample pages from the same web site had matching layouts. For example, on two sample pages collected from the same site, OntoByE identified not only the real keyword "Price" but also other common words such as "List" and "Save" from the context phrases of price values. For these two pages from the same site, a similar problem occurred during the construction of context expressions. Because the user-marked prices are always presented with the same layout, such as "Price: $**100**" and "Price: $**200**", in the context phrases from this site, OntoByE constructs **(Price:)\s$** as the left context of user-marked data 100 and 200. This context obviously would likely not apply to all other price values on HTML pages from other web sites. Our experimental results show that collecting sample pages with different layouts from different sites helps eliminate the non-keyword common words, such as

"List" and "Save", and the non-context common strings, such as **Price:\s**, for user-marked prices. Therefore, to avoid such an over-recognition problem during keyword and context expression construction, we suggest that users always collect sample pages with different layouts from different web sites. This suggestion is reasonable because users usually desire a data-extration ontology which can be applied to all web pages in the domain, not just the pages with a specific layout or pages from a specific web site.

6.2.2 Apartment Rental Advertisement

We employed an Apartment Rental application to compare the OntoByE-generated *Apartment Rental* ontology with a previous *Apartment Rental* ontology hand-written by a member of the BYU Data Extraction Research Group. This comparison helped us observe some of the strengths and limitations of our current OntoByE system.

Figure 30 shows the object sets, relationship sets, and related constraints for the *Apartment Rental* ontology that OntoByE constructed based on the user-defined form, as shown in Figure 28, for the Apartment Rental application. Based on our forms, OntoByE generated exactly the same object sets, relationship sets, and constraints as those in the human-written ontology. Since OntoByE introduces a user-friendly form editor, ordinary users are not required to understand abstract conceptual-model concepts, such as object set, relationship set, and participation constraint. Instead, they only need to know how to design application-dependent forms through the user interface.

The experimental results in Table 2 show that, in the Apartment Rental application, OntoByE found the appropriate data frames for only 6 out of a total 12 object sets (*Bedroom Number, Bathroom Number, Monthly Rate, Deposit* and *Contact Phone*),

Apartment Rental [0:1] Apt Type [1:*]
Apartment Rental [0:1] Bedroom Number [1:*]
Apartment Rental [0:1] Bathroom Number [1:*]
Apartment Rental [0:1] Gender Requirement [1:*]
Apartment Rental [0:1] Date Available [1:*]
Apartment Rental [0:1] Monthly Rate [1:*]
Apartment Rental [0:1] Deposit [1:*]
Apartment Rental [0:*] Features [1:*]
Apartment Rental [0:1] Contact Phone [1:*]
Apartment Rental [0:1] Contact Person [1:*]
Apartment Rental [0:1] Furnished Condition [1:*]
Apartment Rental [0:1] Utility [1:*]

Figure 30: Object and Relationship Sets and Constraints for the Apartment Rental Application

because all other object sets (*Apt Type, Furnished Condition, Gender Requirement, Utility, Features* and *Contact Person*) rely on application-dependent lexicons not yet in our initial data frame library. Therefore, the small scope of the initial data frame library limited OntoByE's abilities to search for appropriate data frames in this specific application. However, for an object set with a small set of constant values, such as *Apt Type* ("private", "shared"), *Utility* ("paid", "not inclued"), *Furnished Condition* ("Furn", "Unfurn") and *Gender Requirement* ("male", "female"), users can easily make use of the data frame editor in OntoByE to create a new data frame with value expressions containing constant values such as **private|shared** and **male|female** which are equivalent to those in the human-written ontology. Furthermore, with considerable future work, we may have OntoByE learn to analyze user-marked data and gather small sets of values.

Table 2 also shows that OntoByE successfully recognized possible keywords and context expressions from sample pages. The comparison analysis of the OntoByE-generated versus human-written ontologies demonstrate that OntoByE's difficulty

constructing complicated keyword and context expressions is mainly due to 1) the limited amount of sample data and 2) the complexity of keyword and context representations in the application. For example, in an apartment ad, people may represent the word "Bedroom" in different ways, such as "bdrm", "bedroom" or "bd." To recognize all these different representations by OntoByE as keywords, which are constructed in the human-written ontology, users need to provide the system with at least two samples for each representation.

Object Set	*Matching Data Frames*	*Left Context Expression*	*Right Context Expression*	*Keywords*
Apt Type	*****	**-**	**-**	**-**
Bedroom Number	SingleDigit	-	-	(Bedroom|bdrm)
Bathroom Number	SmallPositiveReal	**-**	-	(Bathroom|bath)
Furnish Condition	*	-	-	-
Gender Requirement	*	**-**	-	-
Utility	*	**-**	-	-
Features	*	**-**	-	-
Contact Phone	PhoneNumber	**-**	-	(Contact|contact)
Contact Person	*	**-**	-	-
Monthly Rate	Price	$(\s)?	-	-
Deposit	Price	$(\s)?	-	(Deposit|deposit)
Date Available	Date	**-**	-	Available

Note: * Application-dependent Lexicons not in Initial Data Frame Library
- Not Available from Sample Pages

Table 2: Experimental Results of Constructing Data Frames for the Apartment Rental Application

6.3 Summary of OntoByE's Strengths and Weaknesses

Our experimental evidence demonstrates that OntoByE is capable of helping users to construct data frames by searching for the appropriate existing data frames for user-marked data and by recognizing keywords and context expressions from user-provided sample pages.

In general, the comparison between the OntoByE-generated ontology and the human-written ontology shows the strengths and weaknesses of generating ontologies through OntoByE as follows.

Strengths:

- OntoByE allows ordinary users, who have little knowledge of conceptual modeling concepts, to describe information in their domain of interest by creating a set of forms through a user-friendly interface.
- With a limited amount of prior knowledge, OntoByE works well to search for and suggest appropriate existing data frames for some object sets with application-independent values.
- OntoByE recognizes and suggests possible keywords and context expressions for the user-desired data from sample pages. Users can make use of the keywords and context expressions to modify existing data frames or construct new data frames for object sets in their applications. New data frames could subsequently be applied to new applications in the future.

Weaknesses:

- The performance of searching for or constructing data frames by OntoByE is limited by the scope and the quality of prior knowledge.
- The accuracy and completeness of keyword and context expression construction are limited by the number and representativeness of user samples.
- Constructing value expressions for application-dependent data frames requires that users know how to write regular expressions.

7 Conclusion, Limitations and Future Work

In this research, we designed and implemented a semi-automatic ontology generation system, OntoByE, which makes two contributions to conceptual-model-based web data-extraction. First, we implemented a user-friendly interface that helps ordinary users with little knowledge of conceptual models and data-extraction ontologies to take advantage of our resilient web data-extraction approach. Second, we developed a framework for interacting with ordinary users to semi-automatically generate data-extraction ontologies by example. Through examples, OntoByE can leverage prior knowledge from an initial data frame library to help users make use of existing data frames or construct new data frames for a domain of their choosing. The generated data frames can be used to augment the data frame library for future applications. The framework gains the advantage of the by-example approach (user-friendly wrapper creation) without losing the advantage of the BYU DEG approach (resilient wrappers that do not break when a page changes or the wrapper encounters a new domain-applicable page). Our experiments show that OntoByE works well to generate extraction ontologies semi-automatically for two specific domains of interest, the Digital Camera Advertisement and the Apartment Rental Advertisement, with an initial data frame library containing only a small set of application-independent data frames.

In general use, however, our system has room for improvement in several places:

- In this thesis, OntoByE is searching for existing data frames for user-marked data. It does not build application-dependent lexicons for users'

applications. In future work, we may have OntoByE learn to build the lexicons by analyzing user-marked data and by observing patterns of where the data are located on sample pages.

- Currently, the context phrase locator of OntoByE takes a certain number of characters around the user-desired data from the HTML source file to construct context phrases. It does not work well when attempting to locate context and keywords for structured data such as resides in HTML tables. Since the construction of more meaningful context phrases results in a higher probability of locating keywords and context expressions, a more sophisticated context locator may help the data frame matching process achieve higher accuracy.

Despite the current limitations, OntoByE provides a successful by-example framework through which ordinary users can take advantage of the ontology-based approach for their web data-extraction applications. As time goes by, along with the expansion of prior knowledge and the improvements of OntoByE's sub-components, the system will achieve better performance in helping ordinary users to extract data of interest from World Wide Web.

Bibliography

[BLP01] David Buttler, Ling Liu, Calton Pu. A Fully Automated Object Extract System for the Web. In *Proceedings of the 21st International Conference on Distributed Computing (ICDCS-21)*, pp. 361-370, Phoenix, Arizona, 16-19 April 2001.

[CM99] M. E. Califf and R. J. Mooney. Relational Learning of Pattern-Match Rules for Information Extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pp. 328-334, Orlando, Florida, 18-22 July 1999.

[CMM01] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. ROADRUNNER: Towards Automatic Data Extraction from Large Web Sites. In *Proceedings of the 27th VLDB Conference*, pp. 109-118, Roma, Italy, 11-14 September 2001.

[DEG] DEG group and ontology demo home page: *http://www.deg.byu.edu*

[ECJL+99] D.W. Embley, E.M. Campbell, Y.S. Jiang, S.W. Liddle, D.W. Lonsdale, Y.-K. Ng, and R.D. Smith. Conceptual-Model-Based Data Extraction from Multiple-Record Web Documents, *Data and Knowledge Engineering*, Vol. 31, No. 3, pp. 227-251, November 1999.

[Eikvil99] Line Eikvil. Information Extraction from World Wide Web: A Survey. *Technical Report 945*, *Norwegian Computing Center*, 1999.

[Emb98] D. W. Embley. *Object Database Development: Concepts and Principles.* Addison-Wesley, Reading, Massachusetts, 1998.

[Freitag98] Dayne Freitag. Information Extraction from HTML: Application of a General Machine Learning Approach. In *Proceedings of the Fifteenth Conference on Artificial Intelligence AAAI-98*, pp. 517-523, Madison, Wisconsin, 26-30 July 1998.

[FSLE02a] I.M.E. Filha, A.S. da Silva, A.H.F. Laender, and D.W. Embley. Using Nested Tables for Representing and Querying Semistructured Web Data, *The Fourteenth International Conference on Advanced Information Systems Engineering (CAiSE 2002)*, and also A.B. Pidduck, J. Mylopoulos, C.C. Woo and M.T. Ozsu (editors). *Lecture Notes in Computer Science 2348*, pp. 719-723, Toronto, Ontario, Canada, 27-31 May 2002.

[FSLE02b] I.M.E. Filha, A.S. da Silva and A.H.F. Laender and D.W. Embley. Representing and Querying Semistructured Web Data Using Nested Tables with Structural Variants. In *Proceedings of the 21st International Conference on Conceptual Modeling (ER2002)*, pp. 135-151, Tampere, Finland, 7-11 October 2002.

[Hewett00] Hewett, Kimball A. An Integrated Ontology Development Environment for Data Extraction. Masters Thesis, Brigham Young University, 2000.

[HGNY+97] Joachim Hammer, Hector Garcia-Molina, Svetlozar Nestorov, Ramana Yerneni, Marcus Breunig, and Vasilis Vassalos. Template-based wrappers in the TSIMMIS system. In Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 532-535, Tucson, Arizona, 13-15 May 1997.

[Hsu98] Chun-Nan Hsu. Initial Results on Wrapping Semi-structured Web Pages with Finite-State Transducers and Contextual Rules. In *the 1998 Workshop on AI and Information Integration*, pp. 66-73, Madison, Wisconsin, 26-27 July 1998.

[KWD97] N. Kushmerick, D. Weld, and R. Doorenbos. Wrapper Induction for Information Extraction. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97),* pp. 729-735, Nagoya, Japan, 23-29 August 1997.

[LRS02] A.H.F. Laender, B. Ribeiro-Neto and A. Soares da Silva. DEByE - Data Extraction by Example. *Data and Knowledge Engineering*, Vol. 40, No. 2, pp. 121-154, 2002.

[LRST02] A.H.F. Laender, B.A. Ribeiro-Neto, A.S. da Silva and J.S. Teixeira. A Brief Survey of Web Data Extraction Tools. *SIGMOD Record*, Vol. 31, No. 2, pp. 84-93, June 2002.

[MMK99] I. Muslea, S. Minton and G. Knoblock. A Hierarchical Approach to Wrapper Induction. In *Proceedings of the 3rd Conference on Autonomous Agents (Agents '99)*, pp. 190-197, Seattle, Washington, 1-5 May 1999.

[Muslea99] Ion Muslea. Extraction Patterns for Information Extraction Tasks: A Survey. *The AAAI-99 Workshop on Machine Learning for Information Extraction*, Technical Report WS-99-11, Orlando, Florida, 19 July 1999.

[PBE01] Henry Lieberman (editor). *Your Wish Is My Command: Programming by Example*, Morgan Kaufmann Publishers, San Francisco, California, 2001.

[Soderland99] S. Soderland. Learning information extraction rules for semi-structured and free text, *Journal of Machine Learning*, Vol. 34, No. 1-3, pp. 233-272, 1999.

[Zloof77] Zloof, M. M. Query-by-Example: A Data Base Language. *IBM Systems Journal,* Vol. 16, No. 4, pp. 324-343, 1977.