ONTOLOGY GENERATION, INFORMATION HARVESTING AND SEMANTIC ANNOTATION FOR MACHINE-GENERATED WEB PAGES

by

Cui Tao

A dissertation submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Brigham Young University

December 2008

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the dissertation of Cui Tao 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

Date	Kent E. Seamons Graduate Coordinator

Accepted for the College

Date	Thomas W. Sederberg Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

ONTOLOGY GENERATION, INFORMATION HARVESTING AND SEMANTIC ANNOTATION FOR MACHINE-GENERATED WEB PAGES

Cui Tao
Department of Computer Science
Doctor of Philosophy

The current World Wide Web is a web of pages. Users have to guess possible keywords that might lead through search engines to the pages that contain information of interest and browse hundreds or even thousands of the returned pages in order to obtain what they want. This frustrating problem motivates an approach to turn the web of pages into a web of knowledge, so that web users can query the information of interest directly. This dissertation provides a step in this direction and a way to partially overcome the challenges. Specifically, this dissertation shows how to turn machine-generated web pages like those on the hidden web into semantic web pages for the web of knowledge. We design and develop three systems to address the challenge of turning the web pages into web-of-knowledge pages: TISP (Table Interpretation for Sibling Pages), TISP++, and FOCIH (Form-based Ontology Creation and Information Harvesting). TISP can automatically interpret hidden-web tables.

Given interpreted tables, TISP++ can generate ontologies and semantically annotate the information present in the interpreted tables automatically. This way, we can offer a way to make the hidden information publicly accessible. We also provide users with a way where they can generate personalized ontologies. FOCIH provides users with an interface with which they can provide their own view by creating a form that specifies the information they want. Based on the form, FOCIH can generate user-specific ontologies, and based on patterns in hidden-web pages, FOCIH can harvest information and annotate these pages with respect to the generated ontology. Users can directly query on the annotated information. With these contributions, this dissertation serves as a foundational pillar for turning the current web of pages into a web of knowledge.

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor, Dr. David W. Embley. Under his guidance, I successfully overcame many difficulties and learned a lot about how to be a good researcher.

Secondly, I would like to thank my other committee members. I thank Dr. Stephen W. Liddle for his unique insight into software implementation and his help in coding. I thank Dr. Deryle W. Lonsdale for his advices on polishing and presenting research ideas. I thank Dr. Dan R. Olsen and Dr. Kevin D. Seppi for their time and efforts on my dissertation.

I would like to thank my husband, Zonghui, for his continued love, patience, understanding and support during my Ph.D study and my entire endeavor to become more educated. I also want to thank my parents for teaching me how to learn and how to learn well and for encouraging my interest in science since I was a little girl. I thank my son, Andrew, for bringing me so many happiness.

I thank the National Science Foundation for supporting this research under grant No.0414644.

Last, but not least, I thank all the BYU data-extraction research group members for their support and suggestions on my research.

Contents

List of Figures

Chapter 1

Introduction

The World Wide Web serves as a powerful resource for every community. More and more users tend to go online to search for valuable facts or knowledge. The current web, however, is a web of linked pages, instead of a web of facts. Users have to guess possible keywords that might lead through search engines to the pages that contain information of interest. A search engine usually returns thousands of result pages and users have to manually traverse them to find the information they are looking for. This process sometimes can be frustrating.

This frustrating problem motivates an approach to turn the web of pages into a web of knowledge, so that web users can query the information of interest directly. More specifically, the approach superimposes a web of knowledge over the web of pages, allowing users to query the information and then, if desired, taking them directly to the information within a page. This web of knowledge can be realized by transforming web pages into populated conceptual models. We can use a conceptual model to describe the domain a user is interested in and then locate the information of interest on each web page and annotate it with respect to the conceptual model. After we have populated the conceptual model, we can freely query the annotated data.

There are many challenges we have to overcome to achieve this goal. First, creating a conceptual model or an ontology is non-trivial. It not only requires domain expertise, but also requires an understanding of conceptual modeling and a specific ontology language. In order to cover the vast amount of information available online, we might need hundreds of thousands of domain ontologies. In addition, different users have different views even for the same domain. Therefore, they may want their data represent and queried in different ways. To satisfy this desire, we need user-specific ontologies to describe each user's view. All of this makes the ontology creation process even more challenging. Second, to annotate millions of available online pages is cumbersome, especially if we want to annotate with respect to each user's individual view. A way to automate both ontology creation and semantic annotation appears to be necessary if the vision of a web of knowledge is to become reality.

This dissertation provides a step in this direction and a way to partially overcome the challenges. Specifically, this dissertation aims at turning machine-generated web pages like those on the hidden web into semantic web pages for the web of knowledge. Hidden-web pages are usually generated dynamically from a pre-defined templates in response to submitted queries. Hence, pages from the same hidden-web site usually share similar structures. We call this kind of pages *sibling pages.* In addition, a lot of information on the hidden web is presented in tables. We use these hidden web tables in sibling pages as resources to do our research. These tables provide valuable information about what people think is reasonable to represent a domain. We can therefore leverage the representations to generate domain ontologies automatically. Further, if we annotate information stored in these tables, we can offer a way to make the hidden information publicly accessible. In addition, we provide users with an interface with which they can provide their own view by creating a form that specifies the information they want. Based on the form, we can generate user-specific ontologies, and based on patterns in sibling pages, we can harvest information and annotate these pages with respect to the generated ontology.

To address the challenges of automatic ontology generation and semantic annotation based on tables, our first step is to automatically interpret the tables. To interpret a table is to properly associate table category labels with table data values. We have created a table interpretation system called TISP (Table Interpretation for Sibling Pages), which offers a solution for automatically interpret sibling tables. TISP compares them to identify and connect nonvarying components (category labels) and varying components (data values). Experimental results show that TISP can successfully identify sibling tables, generate structure patterns, interpret tables using the generated patterns, and automatically adjust the structure patterns as it processes a sequence of hidden-web pages.

With the ability to automatically interpret hidden-web tables, our next steps are to automatically generate ontologies from these tables and annotate the data in these tables with respect to the generated ontologies. We extended TISP and implemented TISP++. TISP++ can automatically generate OWL ontologies based on hidden-web tables and then automatically annotate information in the tables with respect to the generated ontologies. Being able to interpret tables leads immediately to a conceptualization of the data in these interpreted tables and thus also to a way to semantically annotate these interpreted tables with respect to the ontological conceptualization. Labels in nested table structures yield ontological concepts and interrelationships among these concepts, and associated data values become annotated information. The semantically annotated data leads immediately to queryable data in our envisioned web of knowledge.

TISP and TISP++ provide a fully automatic process to transform facts embedded within hidden-web tables into facts accessible by standard query engines. However, the ontologies are generated based on tables as specified in the hidden-web pages, and the information needs to be queried in the way the tables represent it. Users do not have control over the representation of the concepts, relationships, and constraints of the generated ontologies. To facilitate user-oriented, personalized information gathering and querying, we created FOCIH.

FOCIH (Form-based Ontology Creation and Information Harvesting, pronounced *foh*·sī) is a system that provides for personalized information harvesting—personalized in the sense that the user can specify the ontology into which the information is to be harvested. The form-based part of the name emphasizes the means by which a user creates the ontology—namely by creating a form to be filled in by the system as it harvests information. FOCIH allows a user to generate a form that describes the information the user wishes to harvest. Given a form, FOCIH can generate an extraction ontology, match information with the user's view, and collect information from hidden-web tables automatically. In addition, as an aid to initiating forms, FOCIH can reverse engineer existing tables or ontologies to forms, use them directly, or allow users to make modifications until they have their desired ontological view. By doing so, FOCIH opens a door for annotating information in hidden-web tables with any view—not just the way web-site tables represent it, but according to any view users want—either user created, or based on other tables or on any existing ontology.

The contributions of the dissertation are:

1. it provides a solution to automatic interpretation for sibling tables on the hidden web;
2. it offers a way to automate ontology generation given an interpreted table;
3. it enables automatic semantic annotation for interpreted tables;
4. it gives users a way, without knowing ontology languages, to create ontologies based on their own view;
5. it facilities automatic or semi-automatic information harvesting; and

6. it allows us to create a web of knowledge and superimpose it over hidden-web pages with sibling tables.

We present the details of these contributions in this dissertation as follows. Chapter 2 explains how TISP automatically interprets sibling pages from the hidden web. This chapter is mainly based a paper published in *Proceedings of the 26th International Conference on Conceptual Modeling (*ER'07*)* [52]. Chapter 3 explains how TISP++ generates ontologies based on the interpreted tables and how it annotates information with respect of the generated ontologies and allows the information to be queried. This chapter is mainly based on an invited paper to be published in *Data & Knowledge Engineering* [51]. Chapter 4 explains how FOCIH generates user-specific ontologies and how it harvests and annotates information with respect of user-specific views. This chapter is partially based on a paper published in Proceedings of the *First International Workshop on Conceptual Modelling for Life Sciences Applications (CMLSA'07)* [53]. Chapter 5 concludes the dissertation and discusses future work.

Chapter 2

Automatic Hidden-Web Table Interpretation

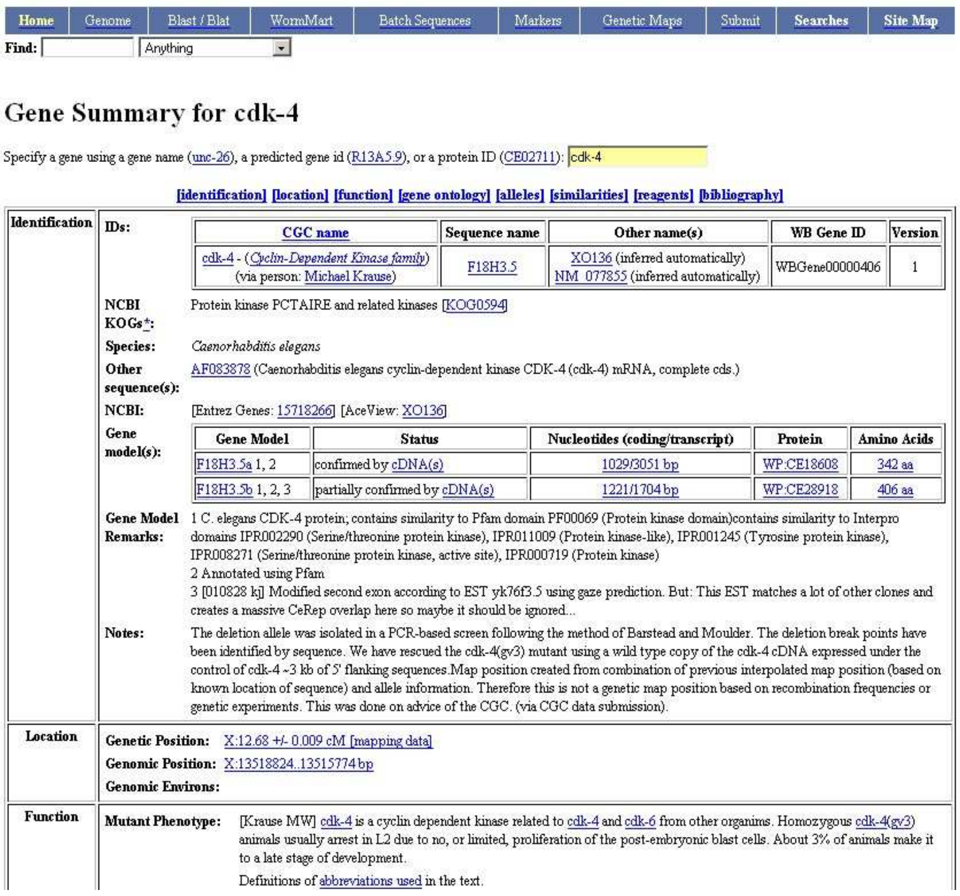
2.1 Introduction

The World Wide Web serves as a powerful resource for every community. Much of this online information, indeed, the vast majority, is stored in databases on the so-called hidden web.[1] Hidden-web information is usually only accessible to users through search forms and is typically presented to them in tables. Automatically understanding hidden-web pages is a challenging task. In this paper, we introduce a domain independent, web-site independent, unsupervised way to automatically interpret tables from hidden-web pages.

Tables present information in a simplified and compact way in rows and columns. Data in one row/column usually belongs to the same category or provides values for the same concept. The labels of a row/column describe this category or concept.

Although a table with a simple row and column structure is common, tables can be much more complex. Figure 2.1 shows an example. Tables may be nested or conjoined as are the tables in Figure 2.1. Labels may span across several cells to give a general description as does *Identification* and *Location* in Figure 2.1. Sometimes tables are rearranged to fit the space available. Label-value pairs may appear in

[1]There are more than 500 billion hidden-web pages. The surface web, which is indexed by common search engines only constitutes less than 1% of the World Wide Web. The hidden web is several orders of magnitude larger than the surface web [32].



Gene Summary for cdk-4

Specify a gene using a gene name (unc-26), a predicted gene id (R13A5.9), or a protein ID (CE02711): cdk-4

[identification] [location] [function] [gene ontology] [alleles] [similarities] [reagents] [bibliography]

Identification

IDs:

CGC name	Sequence name	Other name(s)	WB Gene ID	Version
cdk-4 - (*Cyclin-Dependent Kinase family*) (via person: Michael Krause)	F18H3.5	XO136 (inferred automatically) NM_077855 (inferred automatically)	WBGene00000406	1

NCBI KOGs*: Protein kinase PCTAIRE and related kinases [KOG0594]

Species: *Caenorhabditis elegans*

Other sequence(s): AF083878 (Caenorhabditis elegans cyclin-dependent kinase CDK-4 (cdk-4) mRNA, complete cds.)

NCBI: [Entrez Genes: 15718266] [AceView: XO136]

Gene model(s):

Gene Model	Status	Nucleotides (coding/transcript)	Protein	Amino Acids
F18H3.5a 1, 2	confirmed by cDNA(s)	1029/3051 bp	WP:CE18608	342 aa
F18H3.5b 1, 2, 3	partially confirmed by cDNA(s)	1221/1704 bp	WP:CE28918	406 aa

Gene Model Remarks: 1 C. elegans CDK-4 protein; contains similarity to Pfam domain PF00069 (Protein kinase domain)contains similarity to Interpro domains IPR002290 (Serine/threonine protein kinase), IPR011009 (Protein kinase-like), IPR001245 (Tyrosine protein kinase), IPR008271 (Serine/threonine protein kinase, active site), IPR000719 (Protein kinase)
2 Annotated using Pfam
3 [010828 kj] Modified second exon according to EST yk76f3.5 using gaze prediction. But: This EST matches a lot of other clones and creates a massive CeRep overlap here so maybe it should be ignored...

Notes: The deletion allele was isolated in a PCR-based screen following the method of Barstead and Moulder. The deletion break points have been identified by sequence. We have rescued the cdk-4(gv3) mutant using a wild type copy of the cdk-4 cDNA expressed under the control of cdk-4 ~3 kb of 5' flanking sequences.Map position created from combination of previous interpolated map position (based on known location of sequence) and allele information. Therefore this is not a genetic map position based on recombination frequencies or genetic experiments. This was done on advice of the CGC. (via CGC data submission).

Location

Genetic Position: X:12.68 +/- 0.009 cM [mapping data]
Genomic Position: X:13518824..13515774 bp
Genomic Environs:

Function

Mutant Phenotype: [Krause MW] cdk-4 is a cyclin dependent kinase related to cdk-4 and cdk-6 from other organims. Homozygous cdk-4(gv3) animals usually arrest in L2 due to no, or limited, proliferation of the post-embryonic blast cells. About 3% of animals make it to a late stage of development.
Definitions of abbreviations used in the text.

Figure 2.1: A Sample Table from WormBase [62].

multiple columns across a page or in multiple rows placed below one another down a page. These complexities make automatic table interpretation challenging.

To interpret a table is to properly associate table category labels with table data values. Using Figure 2.1 as an example, we see that *Identification*, *Location*, and *Function* are labels for the large rectangular table. Inside the right cell of the first row is another table with headers *IDs*, *NCBI KOGs*, *Species*, etc. Nested inside of this cell are two tables with labels *CGC name, Sequence name, Other name(s), WB Gene ID*, *Version*, and *Gene Model, Status, Nucleotides (coding/transcript)*, *Protein*, and *Amino Acids*. Most of the rest of the text in the large rectangular table comprises the data values. If we look more closely, however, we may conclude that some category

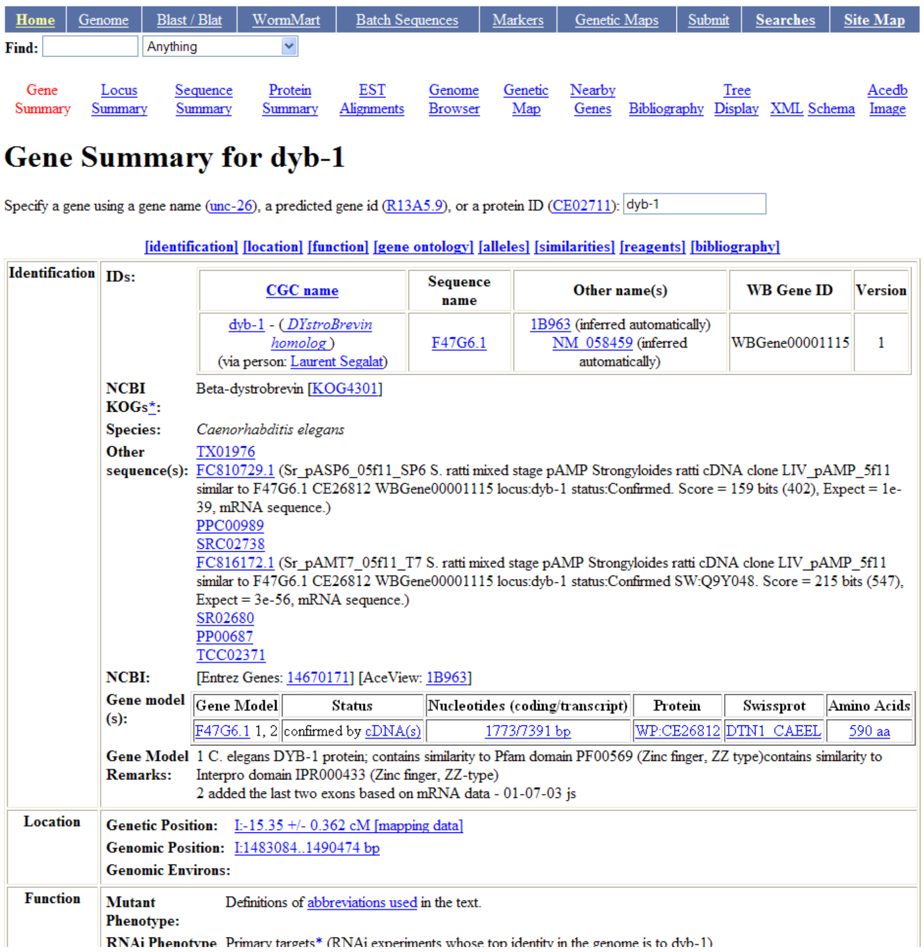
(*Identification.IDs.CGC name*) $\mapsto$
    cdk-4-(Cyclin-Dependent Kinase family)
    (via person: Michael Krause);
(*Identification.IDs.Sequence name*) $\mapsto$ *F18H3.5*;
...
(*Identification.Gene model(s).Amino Acids, 2*) $\mapsto$ *406 aa*;
...

Figure 2.2: Interpretation for the Tables in Figure 2.1 (Partial).

labels are interleaved in the text. For example, *via person* appears to be a label under *CGC name*, as does *Entrez Genes* and *Ace View* beside *NCBI.*

Once category labels and data values are found, we want to properly associate them. For example, the associated label for the value *F18H3.5* should be the sequences of labels *Identification*, *IDs*, and *Sequence name.* Given the source table in Figure 2.1, we match category labels with values as Figure 2.2 shows. We associate one or more sequences of labels with each data value in a table. Borrowing notation from Wang [59], the left hand side of the arrow is a sequence of one or more table labels, and the right hand side of the arrow is a data value. For the first two label-value pairs in Figure 2.2, there is only one label sequence. The third, however, has two: *Identification.Gene model(s).Amino Acids* and *2.* Each label sequence represents a dimension. In general, a table may have one, two, three, or more dimensions. If a table has multiple records (usually multiple rows) and if the records do not have labels, we add record numbers. The table under *Identification.Gene model(s)*, for example, has two records (two rows), but no row labels. We therefore label records with sequence numbers—the first record *1* and the second record *2.* Thus, the label-value association becomes (*Identification.Gene model(s).Amino Acids, 2*) $\mapsto$ *406 aa* where *Identification.Gene model(s).Amino Acids* is the label for the first dimension, and *2* is the row label for the second dimension.

Although automatic table interpretation can be complex, if we have another page, such as the one in Figure 2.3, that has essentially the same structure, the



Home	Genome	Blast / Blat	WormMart	Batch Sequences	Markers	Genetic Maps	Submit	Searches	Site Map

Find: Anything

Gene Summary · Locus Summary · Sequence Summary · Protein Summary · EST Alignments · Genome Browser · Genetic Map · Nearby Genes · Bibliography · Tree Display · XML Schema · Acedb Image

Gene Summary for dyb-1

Specify a gene using a gene name (unc-26), a predicted gene id (R13A5.9), or a protein ID (CE02711): dyb-1

[identification] [location] [function] [gene ontology] [alleles] [similarities] [reagents] [bibliography]

Identification

IDs:

CGC name	Sequence name	Other name(s)	WB Gene ID	Version
dyb-1 - (*DYstroBrevin homolog*) (via person: Laurent Segalat)	F47G6.1	1B963 (inferred automatically) NM_058459 (inferred automatically)	WBGene00001115	1

NCBI KOGs*: Beta-dystrobrevin [KOG4301]

Species: *Caenorhabditis elegans*

Other sequence(s): TX01976
FC810729.1 (Sr_pASP6_05f11_SP6 S. ratti mixed stage pAMP Strongyloides ratti cDNA clone LIV_pAMP_5f11 similar to F47G6.1 CE26812 WBGene00001115 locus:dyb-1 status:Confirmed. Score = 159 bits (402), Expect = 1e-39, mRNA sequence.)
PPC00989
SRC02738
FC816172.1 (Sr_pAMT7_05f11_T7 S. ratti mixed stage pAMP Strongyloides ratti cDNA clone LIV_pAMP_5f11 similar to F47G6.1 CE26812 WBGene00001115 locus:dyb-1 status:Confirmed SW:Q9Y048. Score = 215 bits (547), Expect = 3e-56, mRNA sequence.)
SR02680
PP00687
TCC02371

NCBI: [Entrez Genes: 14670171] [AceView: 1B963]

Gene model (s):

Gene Model	Status	Nucleotides (coding/transcript)	Protein	Swissprot	Amino Acids
F47G6.1 1, 2	confirmed by cDNA(s)	1773/7391 bp	WP:CE26812	DTN1_CAEEL	590 aa

Gene Model Remarks: 1 C. elegans DYB-1 protein; contains similarity to Pfam domain PF00569 (Zinc finger, ZZ type)contains similarity to Interpro domain IPR000433 (Zinc finger, ZZ-type)
2 added the last two exons based on mRNA data - 01-07-03 js

Location

Genetic Position: I:-15.35 +/- 0.362 cM [mapping data]
Genomic Position: I:1483084..1490474 bp
Genomic Environs:

Function

Mutant Phenotype: Definitions of abbreviations used in the text.
RNAi Phenotype Primary targets* (RNAi experiments whose top identity in the genome is to dyb-1)

Figure 2.3: A Second Sample Table from WormBase.

system might be able to obtain enough information about the structure to make automatic interpretation possible. We call pages that are from the same web site and have similar structures *sibling pages*.[2] The two pages in Figures 2.1 and 2.3 are sibling pages. They have the same basic structure, with the same top banners that appear in all the pages from this web site, with the same table title (*Gene Summary for* some particular gene), and a table that contains information about the gene. Corresponding tables in sibling pages are called *sibling tables*. If we compare the two large tables in the main part of the sibling pages, we can see that the first columns of each table are exactly the same. If we look at the cells under the *Identification*

[2]Hidden-web pages are usually generated dynamically from a pre-defined templates in response to submitted queries, therefore they are usually sibling pages

label in the two tables, both contain another table with two columns. In both cases, the first column contains identical labels *IDs*, *NCBI KOGs*, ..., *Putative ortholog(s)*. Further, the tables under *Identification.IDs* also have identical header rows. The data rows, however, vary considerably. Generally speaking, we can look for commonalities to find labels and look for variations to find data values.

Given that we can find most of the label and data cells in this way, our next task is to infer the general structure pattern of the web site and of the individual tables embedded within pages of the web site.[3] With respect to identified labels, we look below or to the right for value associations; we may also need to look above or to the left. In Figure 2.1, the values for *Identification.Gene Model(s).Gene Model* are below, and the values for *Identification.Species* are to the right.

Although we look for commonalities to find labels and look for variations to find data values, we must be careful about being too strict. Sometimes there are additional or missing label-value pairs. The two nested tables whose first column header is *Gene Model* in Figures 2.1 and 2.3 do not share exactly the same structure. The table in Figure 2.1 has five columns and three rows, while the table in Figure 2.3 has six columns and two rows. Although they have these differences, we can still identify the structure pattern by comparing them. The top rows in the two tables are very similar. Observe that the table in Figure 2.3 only has an additional *Swissprot* column inserted between the *Protein* and *Amino Acids* columns. It is still not difficult, however, to tell that the top rows are rows for labels.

In addition to discovering the structure pattern for a web site, we can also dynamically adjust the pattern if the system encounters a table that varies from the pattern. If there is an additional or missing label, the system can change the pattern by either adding the new label and marking it optional or marking the missing label

[3] "Structure patterns" are the pattern expressions (path expressions and regular expressions) we use to identify the location of tables within an HTML page and to associate table labels with table values.

optional. For example, if we had not seen the extra *Swissprot* column in our initial pair of sibling pages, the system can add *Swissprot* as a new label and mark it as optional. The basic label-value association pattern is still the same.

We call our system *TISP* (*Table Interpretation with Sibling Pages*). We present the details of TISP and our contribution to table interpretation by sibling page comparison in the remainder of the paper as follows. Section 2.2 provides the details about how TISP analyzes a source page to recognize all HTML tables and how it decomposes nested tables, if any. Section 2.3 introduces the matching algorithms we use. Section 2.4 describes how we interpreted various matching results and find data tables. Section 2.5 explains how TISP infers the general structure patterns of a web site and therefore how it interprets the tables from the site. Section 2.5 also explains how to automatically adjust the generated patterns when variations are encountered. In Section 2.6, we report the results of experiments we conducted involving sites for car advertisements, and molecular biology. Section 2.7 discusses related work. In Section 2.8, we make concluding remarks.

2.2 Initial Table Processing

The tags <table> and </table> delimit HTML tables in a web document. In each HTML table, there may be tags that specify the structure of the table. The tag <th> is designed to declare a header, <tr> is designed to declare a row, and <td> is designed to declare a data entry. Unfortunately, we cannot count on users to consistently apply these tags as they were originally intended. Most table designers simply use the <td> tag for every table entry without regard to whether it is a header or a data value. In addition, a web page designer might (1) use table tags for layout (i.e. to line up columns and rows of symbols, or values, or statements with no thought of table headers, values. and their associations), or (2) not use HTML tags to represent a table (i.e. use verbatim layout of symbols, values, and statements to form a table). For the first case, TISP needs to determine that the object delimited by HTML table tags is not a table. For the second case, the solution requires techniques beyond those discussed in this paper. We consider this to be interesting future research, and proceed with our discussion of HTML tables.

After obtaining a source document, TISP first parses the source code and locates all HTML components enclosed by <table> and </table> tags (tagged tables). When tagged tables are nested inside of one another, TISP finds them and unnests them. In Figure 2.1, there are several levels of nesting in the large rectangular table. The first level is a table with two columns. The first column contains *Identification*, *Location*, and *Function*, and the second column contains some complex structures. Figure 2.1 shows only the first three rows of this table — one row for *Identification*, one for *Location*, and one for *Function*. (For the purpose of being explicit in this paper, we assume that these three rows are the only rows in this table.) The second column of the large rectangular table in Figure 2.1 contains three second-level nested tables, the first starting with *IDs*, the second with *Genetic Position*, and the third with *Mutant Phenotype*. In the right most cell of the first row is another table. There are also two third-level nested tables.

We treat each tagged table as an individual table and assign an identifying number to it. If the table is nested, we replace the table in the upper level with its identifying number. By so doing, we are able to remove nested tables from upper level tables. As a result, TISP decomposes the page in Figure 2.1 into the set of tables in Figure 2.4.

2.3 Table Matching

To compare and match tables, we first transform each HTML table into a DOM tree [18]. $Tree_1$ in Figure 2.5 shows the DOM tree for Table 7 in Figure 2.4, and $Tree_2$ in Figure 2.5 shows the DOM tree for its corresponding table in Figure 2.3.

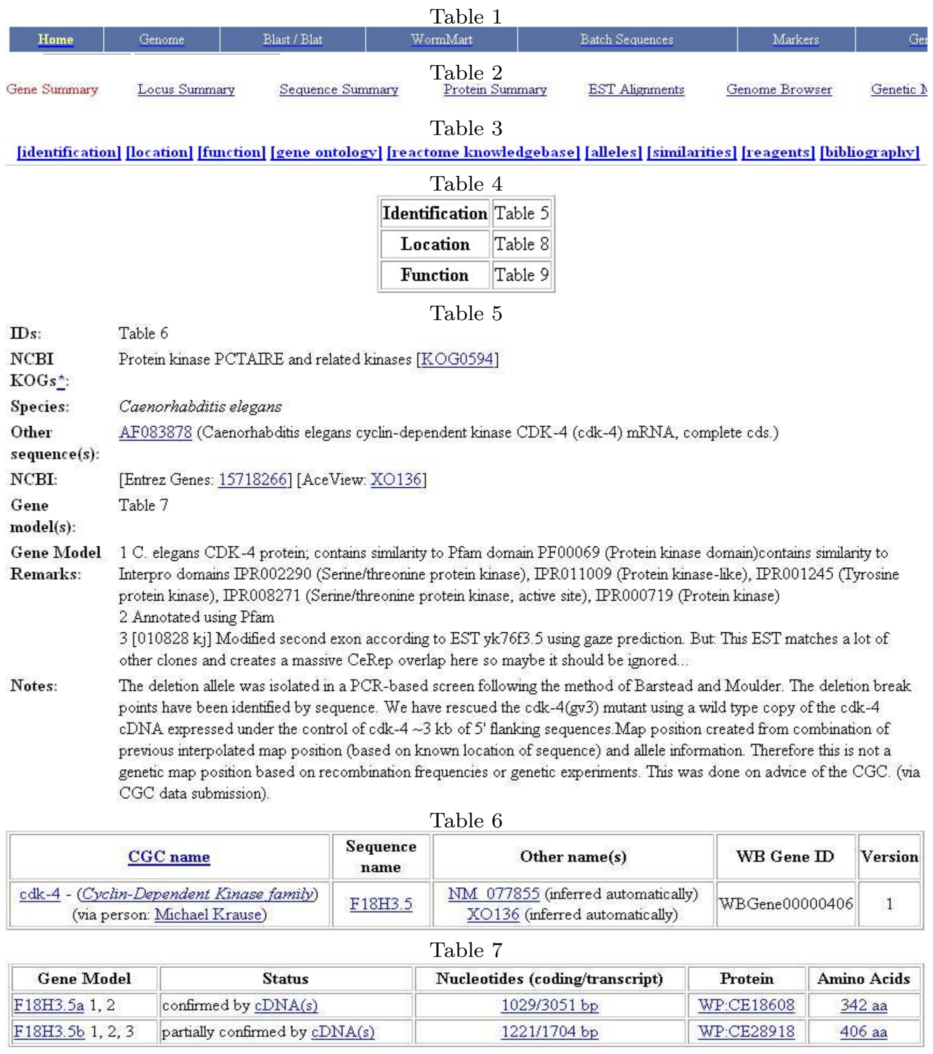


Table 1

Home	Genome	Blast / Blat	WormMart	Batch Sequences	Markers	Ge

Table 2

Gene Summary	Locus Summary	Sequence Summary	Protein Summary	EST Alignments	Genome Browser	Genetic M

Table 3

[identification] [location] [function] [gene ontology] [reactome knowledgebase] [alleles] [similarities] [reagents] [bibliography]

Table 4

Identification	Table 5
Location	Table 8
Function	Table 9

Table 5

IDs:	Table 6
NCBI KOGs*:	Protein kinase PCTAIRE and related kinases [KOG0594]
Species:	*Caenorhabditis elegans*
Other sequence(s):	AF083878 (Caenorhabditis elegans cyclin-dependent kinase CDK-4 (cdk-4) mRNA, complete cds.)
NCBI:	[Entrez Genes: 15718266] [AceView: XO136]
Gene model(s):	Table 7
Gene Model Remarks:	1 C. elegans CDK-4 protein; contains similarity to Pfam domain PF00069 (Protein kinase domain)contains similarity to Interpro domains IPR002290 (Serine/threonine protein kinase), IPR011009 (Protein kinase-like), IPR001245 (Tyrosine protein kinase), IPR008271 (Serine/threonine protein kinase, active site), IPR000719 (Protein kinase) 2 Annotated using Pfam 3 [010828 kj] Modified second exon according to EST yk76f3.5 using gaze prediction. But: This EST matches a lot of other clones and creates a massive CeRep overlap here so maybe it should be ignored...
Notes:	The deletion allele was isolated in a PCR-based screen following the method of Barstead and Moulder. The deletion break points have been identified by sequence. We have rescued the cdk-4(gv3) mutant using a wild type copy of the cdk-4 cDNA expressed under the control of cdk-4 ~3 kb of 5' flanking sequences.Map position created from combination of previous interpolated map position (based on known location of sequence) and allele information. Therefore this is not a genetic map position based on recombination frequencies or genetic experiments. This was done on advice of the CGC. (via CGC data submission).

Table 6

CGC name	Sequence name	Other name(s)	WB Gene ID	Version
cdk-4 - (*Cyclin-Dependent Kinase family*) (via person: Michael Krause)	F18H3.5	NM_077855 (inferred automatically) XO136 (inferred automatically)	WBGene00000406	1

Table 7

Gene Model	Status	Nucleotides (coding/transcript)	Protein	Amino Acids
F18H3.5a 1, 2	confirmed by cDNA(s)	1029/3051 bp	WP:CE18608	342 aa
F18H3.5b 1, 2, 3	partially confirmed by cDNA(s)	1221/1704 bp	WP:CE28918	406 aa

Figure 2.4: Decomposition for the Tables in Figure 2.1.

Tai [50] gives a well acknowledged formal definition of the concept of a tree mapping for labeled ordered rooted trees:

> Let T be a labeled ordered rooted tree and let $T[i]$ be the i^{th} node in level order of tree T. A *mapping* from tree T to tree T' is defined as a triple $(M,\ T,\ T')$, where M is a set of ordered pairs $(i,\ j)$, where i is from T and j is from T', satisfying the following conditions for all $(i_1,\ j_1)$, $(i_2,\ j_2) \in$ M, where i_1 and i_2 are two nodes from T and j_1 and j_2 are two nodes from T':
>
> (1) $i_1 = i_2$ iff $j_1 = j_2$;
>
> (2) $T[i_1]$ comes before $T[i_2]$ iff $T'[j_1]$ comes before $T'[j_2]$ in level order;
>
> (3) $T[i_1]$ is an ancestor of $T[i_2]$ iff $T'[j_1]$ is an ancestor of $T'[j_2]$.

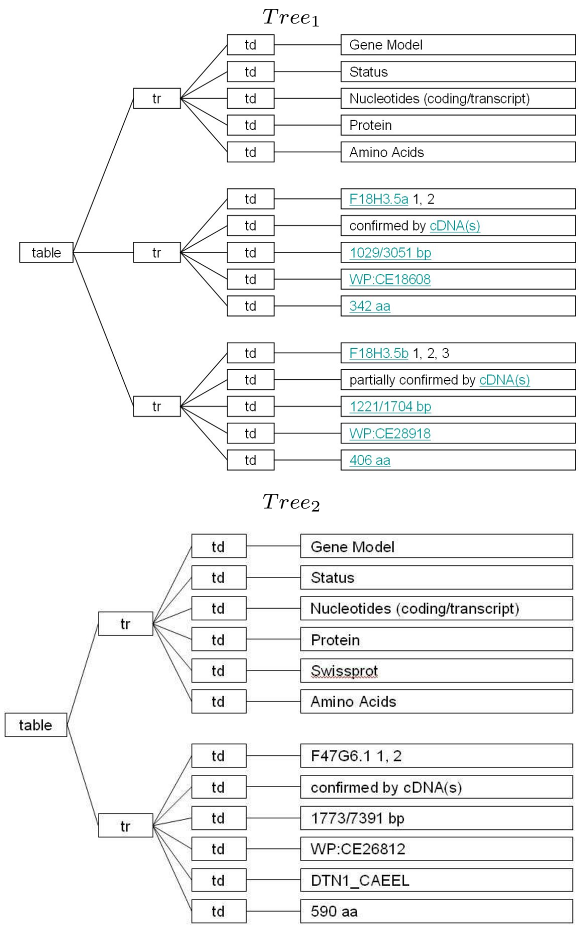


Figure 2.5: Trees for Table 7 in Figure 2.4 and its Sibling Table in Figure 2.3.

According to this definition, each node appears at most once in a mapping — the order between sibling nodes and the hierarchical relation between nodes being preserved. The best match between two trees is a mapping with the maximum number of ordered pairs.

We use a simple tree matching algorithm introduced in [64] which was first proposed to compare two computer programs in software engineering. It calculates the similarity of two trees by finding the best match through dynamic programming with complexity O(n_1n_2), where n_1 is the size (number of nodes) of T and n_2 is the size of T'. This algorithm counts the matches of all possible combination pairs of nodes from the same level, one from each tree, and finds the pairs with maximum matches. The simple tree match algorithm returns the number of these maximum matched pairs. The highlighted part in $tree_1$ in Figure 2.5 shows the matched nodes for $tree_1$ with respect to $tree_2$ in Figure 2.5. The highlighted nodes indicate a match.

2.4 Sibling Table Identification

In our research, we use the results of the simple tree matching algorithm for three tasks: (1) we filter out those HTML tables that are only for layout; (2) we identify the corresponding tables (sibling tables) from sibling pages; and (3) we match nodes in a sibling table pair.

For each pair of trees, we use the simple tree matching algorithm to find the maximum number of matched nodes among the two trees. We call this number the *match score.* For each table in one source page, we obtain match scores. Sibling tables should have a one-to-one correspondence. Based on the match scores, we use the Gale-Shapley stable marriage algorithm [27] to pair sibling tables one-to-one from two sibling pages.

For each pair of tables, we calculate the *sibling table match percentage*, 100 times the match score divided by the number of nodes of the smaller tree. The match percentage between the two trees in Figure 2.5, for example, is 19 (match score) divided by 27 (tree size of $Tree_2$), which, expressed as a percentage, is 70.4%.

We classify table matches into three categories: (1) exact match or near exact match; (2) false match; and (3) sibling-table match. We use two threshold boundaries to classify table matches: a higher threshold between exact or near exact match and sibling-table match, and a lower threshold between sibling-table match and false match. Usually a large gap exists between the range of exact or near exact match percentages and the range of sibling-table match percentages, as well as between the range of sibling-table match percentages and the range of false match percentages. After some observation, we set the upper threshold at 90% and the lower threshold at 20%.

In our example, Tables 1, 2, and 3 have match percentages of 100% with their sibling tables. The match percentages for Tables 4, 5, 6 and 7, and their corresponding sibling tables, are 66.7%, 58.8%, 69.2%, and 70.4% respectively. Our example has no false matches. A false match usually happens when a table does not have a corresponding table in the sibling page. In this case, we save the table. When more sibling pages are compared, we might find a matching table.

2.5 Structure Patterns

The first component of a structure pattern for a table specifies the table's location in a web page. To specify the location, we use XPath [63], which describes the path of the table from the root HTML tag of the document. For example, The location for Table 7 in Figure 2.4 is:
/html/table[4]/tbody/tr[1]/td[2]/table[2]/tbody/tr[1]/td[2]. An XPath simply lists the nodes (HTML tag names) of a path in a DOM tree for the HTML document where [n] designates the nth sibling node in the ordered subtree.

The second component of a structure pattern specifies the label-value pairs for a table and thus provides the interpretation. We now give the details about how we identify the proper label-value pattern template (Section 2.5.1) and use it to generate the specific label-value-pair pattern for the table (Section 2.5.2). We then explain how TISP uses the generated pattern to extract label-value pairs from the table and how TISP produces an interpretation for the table (Section 2.5.3). Combinations of basic patterns are also possible; we thus also explain how to generate and use combination patterns (Section 2.5.4). Finally, we explain how TISP dynamically adjusts a pattern to accommodate table variations it may encounter as it extracts label-value pairs from sibling tables in the web site (Section 2.5.5).

Pattern 1:

$$<\mathit{table}>(<\mathit{tbody}>)?\boxed{<\mathit{tr}>(<(\mathit{td}|\mathit{th})>\{L\})^n}\boxed{(<\mathit{tr}>(<(\mathit{td}|\mathit{th})>\{V\})^n}\,)^+$$

Pattern 2:

$$<\mathit{table}>(<\mathit{tbody}>)?(\boxed{<\mathit{tr}><(\mathit{td}|\mathit{th})>\{L\}(<(\mathit{td}|\mathit{th})>\{V\})^n}\,)^+$$

Pattern 3:

$$<\mathit{table}>(<\mathit{tbody}>)?\boxed{<\mathit{tr}>(<(\mathit{td}|\mathit{th})>\{L\})^n}$$
$$(\boxed{<\mathit{tr}><(\mathit{td}|\mathit{th})>\{L\}(<(\mathit{td}|\mathit{th})>\{V\})^{(n-1)}}\,)^+$$

Figure 2.6: Some Basic Pre-defined Pattern Templates.

2.5.1 Pattern Templates

We use regular expressions to describe table structure pattern templates. If we traverse a DOM tree, which is ordered and labeled, in a preorder traversal, we can layout the tree labels textually and linearly. We can then use regular-expression-like notation to represent the table structure patterns (see Figure 2.6). In both templates and generated patterns we use standard notation: ? (optional), + (one or more repetitions), and | (alternative). In templates, we augment the notation as follows. A variable (e.g. n) or an expression (e.g. n-1) can replace a repetition symbol to designate a specific number of repetitions. A pair of braces { } indicates a leaf node. A capital letter L is a position holder for a label and a capital letter V is a position holder for value. The part in a box is an *atomic pattern* which we use for combinational structural patterns in Section 2.5.4.

Figure 2.6 shows three basic pre-defined pattern templates. Pattern 1 is for tables with n labels in the first row and with n values in each of the rest of the rows. The association between labels and values is column-wise; the label at the top of the column is the label for all the values in each column. Pattern 2 is for tables with labels in the left-most column and values in the rest of the columns. Each row has a label followed by n values. The label-value association is row-wise; each label labels all values in the row. Pattern 3 is for two-dimensional tables with labels on both the top and the left. Each value in this kind of table associates with both the row header label and the column header label. As future work, we could define additional patterns and experiment with them or allow users to define additional patterns, but the patterns and combinations of patterns we have constitute a large majority of HTML tables.

2.5.2 Pattern Generation

To check whether a table matches any pre-defined pattern template, TISP tests each template until it finds a match. When we search for a matching template, we only consider leaf nodes and seek matches for labels and mismatches for values. Variations, however, exist and we must allow for them. In tables, labels or values are usually grouped. We are seeking for a structure pattern instead of classifying individual cells. Sometimes we find a matched node, but all other nodes in the group are mismatched nodes and agree with a certain pattern in; in such case TISP should ignore the disagreement and assume the matched node is a mismatched node of values too. Specifically, we calculate a *template match percentage* between a pre-defined pattern template and a matched result, 100 times the number of leaf nodes that agree with a pattern template divided by total number of leaf nodes in the tree. We calculate the template match percentage between a table and each pre-defined structure template. A match must satisfy two conditions: (1) it must be the highest match percentage, and (2) the match percentage must be greater than a threshold, which we set at 80%.

Consider the mapped result in Figure 2.5 as an example. The highlighted nodes are matched nodes in $tree_1$. Comparing the template match percentage for this mapped result for the three pattern templates in Figure 2.6, we obtain 93.3%, 53.3%, and 80% respectively. Pattern 1 has the highest match percentage, and it is greater than the threshold. Therefore we choose Pattern 1.

We now impose the chosen pattern, ignoring matches and mismatches. Note that for $tree_1$ in Figure 2.5, the first branch matches the part in Pattern 1 in the first box, and the second and the third branch each match the part in the second box,

/html/table[4]/tbody/tr[1]/td[2]/table[2]/tbody/tr[1]/td[2]
$< table >< tr >$
$< td >$ *Gene Model*
$< td >$ *Status*
$< td >$ *Nucleotides(coding/transcript)*
$< td >$ *Protein*
$< td >$ *Amino Acids*
$(< tr >$
$< td > V_{Gene\ Model}$
$< td > V_{Status}$
$< td > V_{Nucleotides(coding/transcript)}$
$< td > V_{Protein}$
$< td > V_{Amino\ Acids})^{+}$

Figure 2.7: Structure Pattern for Table 7 in Figure 2.4.

where n is five. For Pattern 1, when n=1, we have a one-dimensional table; and when n>1, we have a two-dimensional table for which we must generate record numbers.

After TISP matches a table with a pre-defined pattern template, it generates a specific structure pattern for the table by substituting the actual labels for each L and by substituting a placeholder V_L for each value. The subscript L for a value V designates the label for the label-value pair for each record in a table. Figure 2.7 shows the specific structure pattern for Table 7 in Figure 2.4.

2.5.3 Pattern Usage

With a structure pattern for a specific table, we can interpret the table and all its sibling tables. The XPath gives the location of the table, and the generated pattern gives the label-value pairs. The pattern must match exactly in the sense that each label string encountered must be identical to the pattern's corresponding label string. Any failure is reported to TISP. (In Section 2.5.5, we explain how TISP reacts to a failure notification.)

When the pattern matches exactly, TISP can generate an interpretation for the table. For our example, the chosen pattern is Pattern 1 (a table with column headers and one or more data rows). Thus, TISP needs to add another dimension and add row numbers. Since the table is inside of other tables, TISP recursively searches for the tables in the upper levels of nesting and collects all needed labels.

Location	chr8	Strand	+
Sequence Length	5095	Total Exon Length	2161
Number of Exons	4	Number of SNPs	0
Max Exon Length	1044	Min Exon Length	93

Figure 2.8: An Example for Pattern Combination from MutDB.

2.5.4 Pattern Combinations

It is possible that TISP cannot match any pre-defined template. In this case, it looks for pattern combinations. Using Figure 2.8 as an example, assume that TISP matches all cells in the first and third column, but none in the second and forth column. Comparing the template match percentage for this mapped result for the three pattern templates in Figure 2.6, we obtain 50%, 75%, and 68.8% respectively. None of them is greater than the threshold, 80%. The first two columns, however, match Pattern 2 perfectly, as do the last two columns.

Patterns can be combined row-wise or column-wise. In a row-wise combination, one pattern template can appear after another, but only the first pattern template has the header: $< table >$ $(< tbody >)?$. Therefore, a row-wise combined structure pattern has a few rows matching one template and other rows matching another template. In a column-wise combination, we combine different atomic patterns. If a pattern template has two atomic patterns, both patterns must appear in the combined pattern, in the same order, but they can be interleaved with other atomic patterns. If one atomic pattern appears after another atomic pattern from a different pattern template, the $< tr >$ tag at the beginning is removed. Figure 2.9 shows two examples of pattern combinations. Example 1 combines Pattern 2 and Pattern 1 row-wise. Example 2 combines Pattern 2 with itself column-wise. This second pattern matches the table in Figure 2.8, where $n = m = 1$, and the plus $(+)$ is 4.

The initial search for combinations is similar to the search for single patterns. TISP checks patterns until it finds mismatches, it then checks to see whether the

Example 1:
$< table > (< tbody >)?$
$(< tr >< (td|th) >\{\mathrm{L}\}(< (td|th) > \{V\})^{n})^{+}$
$< tr > (< (td|th) > \{L\})^{m}(< tr > (< (td|th) > \{V\})^{m})^{+}$
Example 2:
$< table > (< tbody >)?$
$(< tr >< (td|th) >\{\mathrm{L}\}(< (td|th) > \{V\})^{n}< (td|th) >\{\mathrm{L}\}(< (td|th) > \{V\})^{m})^{+}$

Figure 2.9: Two Examples of Pattern Combinations.

mismatched part matches with some other pattern. TISP first searches row-wise for rows of labels and then uses these rows as delimiters to divide the table into several groups. If it cannot find any row of labels, it repeats the same process column-wise. TISP then tries to match each sub group with a pre-defined template. This process repeats recursively until all sub-groups match with a template or the process fails to finding any matching template.

For the example in Figure 2.8, TISP is unable to find any rows of labels, but finds two columns of labels, the first and third column. It then divides the table into two groups using these two columns and tries to match each group with a pre-defined template. It matches each group with Pattern 2. Therefore, this table matches column-wise with Pattern 2 used twice.

2.5.5 Dynamic Pattern Adjustment

Given a structure pattern for a table, we know where the table is in the source document (its XPath), the location of the labels and values, and the association between labels and values. When TISP encounters a new sibling page, it tries to locate each sibling table following the XPath, and then tries to interpret it by matching it with the sibling table structure pattern. If the encountered table matches the structure pattern regular expression perfectly, we successfully interpret this table. Otherwise, we might need to do some pattern adjustment. There are two ways to adjust a structure pattern: (1) adjust the XPath to locate a table, and (2) adjust the generated structure pattern regular expression.

$< table >< tr >< td >$*Gene Model*$< td >$*Status* $< td >$*Nucleotides(coding/transcript)*
$< td >$*Protein* $(< td >$*Swissprot*$)? < td >$*Amino Acids*
$(< tr > < td > V_{Gene\ Model} < td > V_{Status} < td > V_{Nucleotides(coding/transcript)}$
$< td > V_{Protein}(< td > V_{Swissprot})? < td > V_{Amino\ Acids})^{+}$

Figure 2.10: Structure Pattern for the Table in Figure 2.3.

Although sibling pages usually have the same base structure, some variations might exist. Some sibling pages might have additional or missing tables. Thus, sometimes, following the XPath, we cannot locate the sibling table for which we are looking. In this case, TISP searches for tables at the same level of nesting, looking for one that matches the pattern. If TISP finds one, it obtains the XPath and adds it as an alternative. Thus, for future sibling pages, TISP can (in fact, always does) check all alternative XPaths before searching for another alternative XPath. If TISP finds no matching table, it simply continues its processing with the next table.

We adjust a table pattern when we encounter a variation of an existing table. There might be additional or missing labels in the encountered variation. In this case, we need to adjust the structure pattern regular expression, to add the new optional label or to mark the missing label as optional. Consider the table that starts with *Gene Model* in Figure 2.3 (the sibling table of Table 7 in Figure 2.4) as an example. The table matches the pattern in Figure 2.7 until we encounter the label *Swissprot*. If we skip *Swissprot*, the next label *Amino Acids* matches the structure pattern. In this case, we treat *Swissprot* as an additional label, and we add it as an optional label as Figure 2.10 shows.

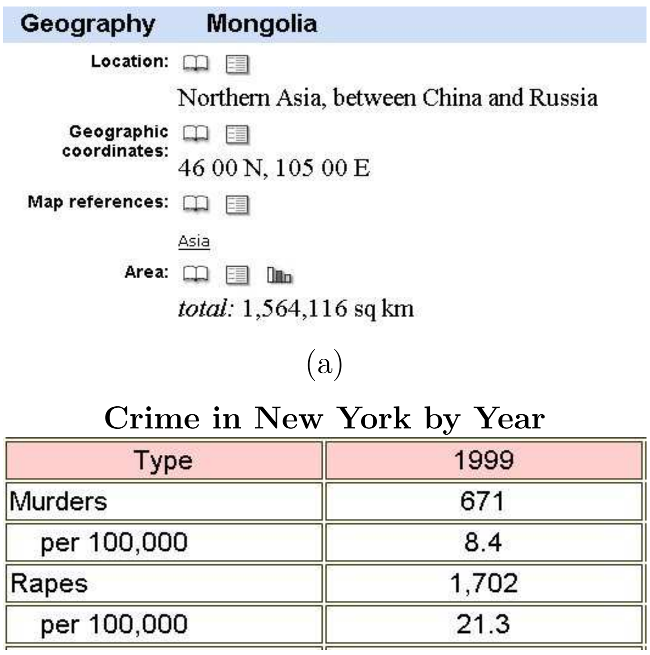
2.6 Experimental Results

We tested TISP using source pages from commercial data, scientific data, and geopolitical data. We picked pages from each field: car advertisements for commercial data, molecular biology for scientific data, and interesting information about US states and about countries for geopolitical data. Most of the source pages were collected from popular and well-known web sites such as cars.com, NCBI database, Wormbase, MTB (Mouse Tumor Biology Database), CIA's World Factbook, and U.S. Geological Survey (usgs.gov). We tested more than 2,000 tables found in 275 sibling pages in 35 web sites. Most pages from the molecular biology domain and the geopolitical domain have relatively complicated structures. Seven out of ten sites in the geopolitical domain and eight out of ten sites in the molecular biology domain contained multiple data tables per page. Two of the geopolitical sites and eight of the molecular biology sites contained nested HTML tables.

For each web site, we randomly chose two sibling pages for initial pattern generation. For the initial two sibling pages, we tested (1) whether TISP was able to recognize HTML data tables and discard HTML tables used only for layout, (2) whether it was able to pair all sibling tables correctly, and (3) whether it was able to recognize the correct pattern template or pattern combination. For the remaining sibling pages from the same web site, we tested (1) whether TISP was able to interpret tables using the recognized structure patterns, (2) whether it correctly detected the need for dynamic adjustment, and (3) whether it recognized new structure patterns correctly.

We collected 75 sibling pages from 15 different web sites in the car-advertisements domain for a total of 780 HTML tables.[4] TISP correctly discarded all uses of tables for layout and successfully paired all sibling tables. There were no nested tables in this domain. Most of the web sites contained only one table pattern, except for one site that had three different patterns. Two web sites contained tables with structure combinations. Of the 17 pairs of sibling tables, TISP recognized 16 correctly. The one pattern TISP failed to recognize correctly contained too many value cells that included the same value (values such as *unknown*, *general car*, *auto*, *dealer*, and empty spaces). TISP considered them as labels, and thus could not match

[4]The sibling pages in this domain are usually very regular. Indeed, we found no table variations in any of the sites we considered. We therefore only tested five pages per site.



Geography	Mongolia
Location:	Northern Asia, between China and Russia
Geographic coordinates:	46 00 N, 105 00 E
Map references:	Asia
Area:	*total:* 1,564,116 sq km

(a)

Crime in New York by Year

Type	1999
Murders	671
per 100,000	8.4
Rapes	1,702
per 100,000	21.3

(b)

Figure 2.11: Two Partially Misinterpreted Tables.

the table with any pre-defined pattern template or detect any pattern combination. TISP successfully interpreted all tables from the generated patterns. No adjustments were needed, neither for any path nor for any label.

For the geopolitical information domain, we tested 100 sibling pages from 10 different web sites with 884 HTML tables. TISP correctly paired 100% of all data tables and correctly discarded all layout tables. For initial pattern generation, TISP was able to recognize all 22 structure patterns successfully. As TISP processed additional sibling pages, it found one additional sibling table and correctly interpreted it. There were no path adjustments, but there were 22 label adjustments — all of them correct. For two sets of sibling tables, TISP recognized the correct patterns, but failed to recognize some implicit information that affects the meaning of the tables. Therefore it interpreted these tables only partially correctly. Figure 2.11 shows these two cases. There are actually two HTML tables in Figure 2.11a. The header *Geography Mongolia* is in one HTML table, and the rest of information is in another HTML table. Because it separated tables using HTML tags, TISP was not able to determine the relationship between these two HTML tables. TISP correctly interpreted Figure 2.11b as Pattern 3. It, however, did not recognize the relationship between *Murders* and *per 100,000* and between *Rapes* and *per 100,000.*

We collected 100 sibling pages from 10 different web sites in the molecular biology domain for a total of 862 HTML tables. Among these tables, TISP falsely classified three pairs of layout tables as data tables. TISP, however, successfully eliminated these false sibling pairs during pattern generation because it was unable to find a matching pattern. No false patterns were generated. TISP was able to recognize 28 of 29 structure patterns. TISP missed one pattern because the table contained too many empty cells. If it had considered empty cells as mismatches, TISP would have correctly recognized this pattern. As TISP processed additional sibling pages, it found 5 additional sibling tables and correctly interpreted all but one of them. The failure was caused by labels that varied across sibling tables causing them, in some cases, to look like values. There were 5 path adjustments and 12 label adjustments — all of them correct. One table was interpreted only partially correctly because TISP considered the irrelevant information *To Top* as a header.

For measuring the overall accuracy of TISP, we computed precision (P), recall (R), and an F-measure ($F = 2PR/(P{+}R)$). In its table recognition step, TISP correctly discarded 155 of 158 layout tables and discarded no data tables. It therefore achieved an F-measure of 99.0% (98.1% recall and 100% precision). TISP later discarded these three layout tables in its pattern generation step, but it also rejected two data tables, being unable to find any pattern for them. It thus achieved an F-measure of 99.4% (100% recall and 98.8% precision). For table interpretation, TISP correctly recognized 69 of 74 structure patterns. It therefore achieved a recall of 93.2%. Of the 72 structure patterns it detected, 69 were correct. It therefore achieved a precision of 95.8%. Overall the F-measure for table interpretation was 94.5% for the sites we tested.

We discuss the time performance of TISP in two phases: (1) initial pattern generation from a pair of sibling pages and (2) interpretation of the tables in the rest of the sibling pages. The time for the pattern generation given a pair of sibling pages consists of: (1) the time to read and parse the two pages and locate all the HTML tables, (2) the time for sibling table comparisons, and (3) the time to select from pre-defined structure templates and generate a pattern. The complexity of parsing and locating HTML tables is O(n), where n is the number of HTML tags. The simple tree matching algorithm has time complexity O(m_1m_2), where m_1 and m_2 are the numbers of nodes of the two sibling trees. To find the best match for each HTML table, we need to compare each table with all the HTML tables in its sibling page. The time complexity is O(km_1m_2), where k is the number of HTML tables in the sibling page. The time complexity for finding the correct pattern for each matched sibling table is O(pl), where p is the number of pattern templates and l is the number of leaf nodes in the HTML table. If pattern combinations are involved, the complexity of template discovery increases exponentially since for each subgroup we must consider every template and find the best match. We conducted the experiment on a Pentium 4 computer running at 3.2 GHz. The typical actual time needed for the pattern generation for a pair of sibling pages was below or about one second. The actual time reached a maximum of 15 seconds for a complicated web site where pages had more than 20 tables.

The time for table interpretation for a single sibling web page when no adjustment is necessary consists of: (1) the time for locating each table and (2) the time for processing the table with a pattern. The complexity of locating a table is O(p), where p is the number of path possibilities leading to the table. Each path possibility is itself logarithmic with respect to the number of nodes in the DOM tree for the pages. The complexity of matching a located table with the corresponding pattern is O(el), where e is the number of pattern entries (an entry could be either a pattern label or a pattern value) of the pattern and l is the number of leaf nodes in the HTML table's DOM tree. The time to do adjustments ranges from the time to do a simple label adjustment, which is constant, to the time required to re-evaluate all sibling tables, which is the same as the time for initial pattern generation. Overall, the typical actual time needed for interpreting tables in one page was below one second. The actual time reached a maximum of 19 seconds for a complicated web page with several tables and several adjustments.

2.7 Related Work

2.7.1 Sibling Page Comparison

Other researchers have also tried to take advantage of sibling pages. RoadRunner [13] compares two HTML pages from one web site and analyzes the similarities and dissimilarities between them in order to generate extraction wrappers. It discovers data fields by string mismatches and discovers iterators and optionals by tag mismatches. EXALG [4] uses equivalence classes (sets of items that occur with the same frequency in sibling pages) and differentiating roles to generate extraction templates for the sibling pages. DEPTA [67] compares different records in a page instead of sibling pages and tries to find the extraction template for the record. This approach first tries to find individual data records by using a few heuristics. It then uses a tree edit distance algorithm to compare different data records and tries to find the extraction region. The approach in [37] compares sibling pages to filter out general headers and footers and other constant non-data areas of a page. It then makes various comparisons among main pages and linked pages to find record segmentations.

TISP fundamentally differs from these approaches. The first three [4, 13, 67] focus on finding data fields, and the technique in [37] focuses on record segmentation. They do not discover labels or try to associate data and labels. TISP focuses on table interpretation. It looks for a table pattern in addition to data fields. Furthermore, TISP also tries to find the general structure pattern for the entire web site. It dynamically adjusts the structure pattern as it encounters new, yet-unseen structures.

2.7.2 Table Interpretation

Automated table processing is typically done in two steps: (1) table recognition — find the data table, and (2) table interpretation — find and associate labels and values. Recent surveys [22, 66] describe the vast amount of research that has been done in table processing and illustrate the challenges of automated table processing. Most of this work is about tables in imaged documents, but some is about HTML tables. Since we focus in this paper only on HTML tables, we limit the related work we discuss to only HTML table processing.

Several researchers have tried to differentiate data tables from tables for layout [9, 12, 28, 60]. They have tried to use machine learning methods [12, 60], visual level features [28, 29], and linguistic features [9]. TISP provides a unique way to do this task when sibling pages are available. By considering the match percentage between tables, we were able to filter out all the layout tables in the car and geopolitical domain and only failed to filter out three pairs of tables out of more than 800 HTML tables from the molecular biology domain (These three false positives were also filtered out during the process of pattern generation). The approaches [9, 12, 28, 60] were able to achieve F-measures of 86.5% [9], 95.5% [12], 90.0% [28], and 87.6% [60]. By way of a comparison, TISP was able to achieve an F-measure of 99.4%. TISP techniques, of course, only work when sibling pages and tables are available. Further, the experimentation was for different data sets, so these comparative results should not be construed as being definitive. The results only give an indication of about where the techniques might stand with respect to each other.

Several papers have discussed the HTML table interpretation problem. Some table interpretation systems work based on simple assumptions and heuristics (e.g. [9, 24, 25, 31, 38]). These simple assumptions (labels are either the first few rows or the first few columns) are easily broken in complex tables such as nested tables (e.g. Figure 2.1) or tables with combination structures (e.g. Figure 2.8). The approach in [43] presents a table interpretation system for automatic generation of F-logic frames for tables. It considers many linguistic features in a table such as alphabetic features, numeric features, number ranges, and data formats. It calculates differences among different regions of a table to detect the orientation of a table and to locate label cells and value cells. The average F-measure of this approach is around 50%. The technique in [55] learns lexical variants from training examples and uses a vector space model to deal with non-exact matches among labels. It also uses a few heuristics to find the association among labels and values, achieving an F-measure of 91.4%. The approach in [29] uses visual boxes instead of HTML tags to interpret HTML tables. It achieves an F-measure of 52.1% (the precision value was 57% and the recall value was 48%). By way of comparison, TISP is able to achieve an F-measure of 94.5%. Of course, TISP only works when sibling tables are available. On the other hand, when applicable, TISP has the advantage over machine learning because it is unsupervised and document and web-site independent. TISP has no need for training data and works for all domains and web sites where sibling pages with sibling tables are available. Here again, these comparative results should not be construed as being definitive since the various research groups used different data sets for experimentation. Furthermore, the measurements here are for solving variations of the same problem, not identical problems.

2.8 Concluding Remarks

In this chapter we introduced TISP, which provides a way to automatically interpret tables in hidden-web pages—pages which are almost always sibling pages. By comparing data tables in sibling pages, TISP is able to find the location of table labels and data entries, and pair them to infer the general pattern for all sibling tables from the same site. Our experiments using source pages from three different domains—car advertisements, molecular biology, and geopolitical information—indicate that TISP can succeed in properly interpreting tables in sibling pages. TISP achieved an F-measure for sibling table interpretation of 94.5%.

Several directions remain to be pursued. For TISP and table interpretation, we would like to do the following. (1) We assumed that information in one table cell is either a table label or a table value. There could be structured information within a cell, however, such as the label *via person* and the value *Michael Krause* in Figure 2.1. As a future work on table interpretation, we would like to analyze cell content to find structured information within cells. (2) Some web pages use lists as tables; we would like to consider them too. (3) We would also like to be able to deal with the case in Figure 2.11a, where we need to join adjacent HTML tables to form a single table, and we would like to improve TISP so that it can interpret tables with factored labels, where part of a label has been removed and placed in a higher level heading, such as those in Figure 2.11b.

Chapter 3

Information Conceptualization and Semantic Annotation

3.1 Introduction

One goal of the semantic web is to enable automatic targeted access to online information that can be useful to a user. This might include activities such as finding the availability and price of a specific car, reserving and ticketing travel itineraries, looking for family history information, studying basic knowledge for a scientific domain, or conducting research based on online databases. Much of this online information, indeed, the vast majority, is stored in databases on the so-called hidden web. Currently tools for retrieving and querying hidden-web data are in their infancy, and much human intervention is needed to hard-code and hand-specify their functionality.

To increase the use of the hidden-web data, we need a more automatic way of making it machine-interpretable, and we need to annotate it with a more systematic description of respective semantic domains. This goal can be realized when the information on the web is annotated by an ontology. An ontology enables domain experts to declare standardized, sharable, machine-processable knowledge.

To generate an ontology, however, is non-trivial. It not only requires domain expertise, but also requires the knowledge of conceptual modeling and an ontology language. In addition, to cover the vast amount of information available online, we might need thousands of domain ontologies. A system that can generate ontologies automatically is becoming increasingly desirable.

One possible solution for automatic ontology generation is to generate based on online tables. Online tables provide valuable information about what people think is a reasonable way to represent a domain. We can leverage tables to generate ontologies that describe the domain. In our previous research, we have implemented a system that can automatically interpret online tables found in sibling pages on the hidden web [52]. We call the system TISP (Table Interpretation for Sibling Pages). With the ability to automatically interpret hidden-web tables, we have extended TISP and implemented TISP++. TISP++ can automatically generate OWL ontologies based on hidden-web tables. Labels in nested table structures yield ontological concepts and interrelationships among these concepts.

Further, after we have a domain ontology, we can take the next step and automatically annotate hidden-web tables. Since TISP can automatically interpret these tables, the annotation becomes straightforward. TISP++ annotates each data value in a table using its corresponding label (concept). The semantically annotated data leads immediately to queryable data. Since TISP++ records information about annotated data in RDF, we can use SPARQL to query it.

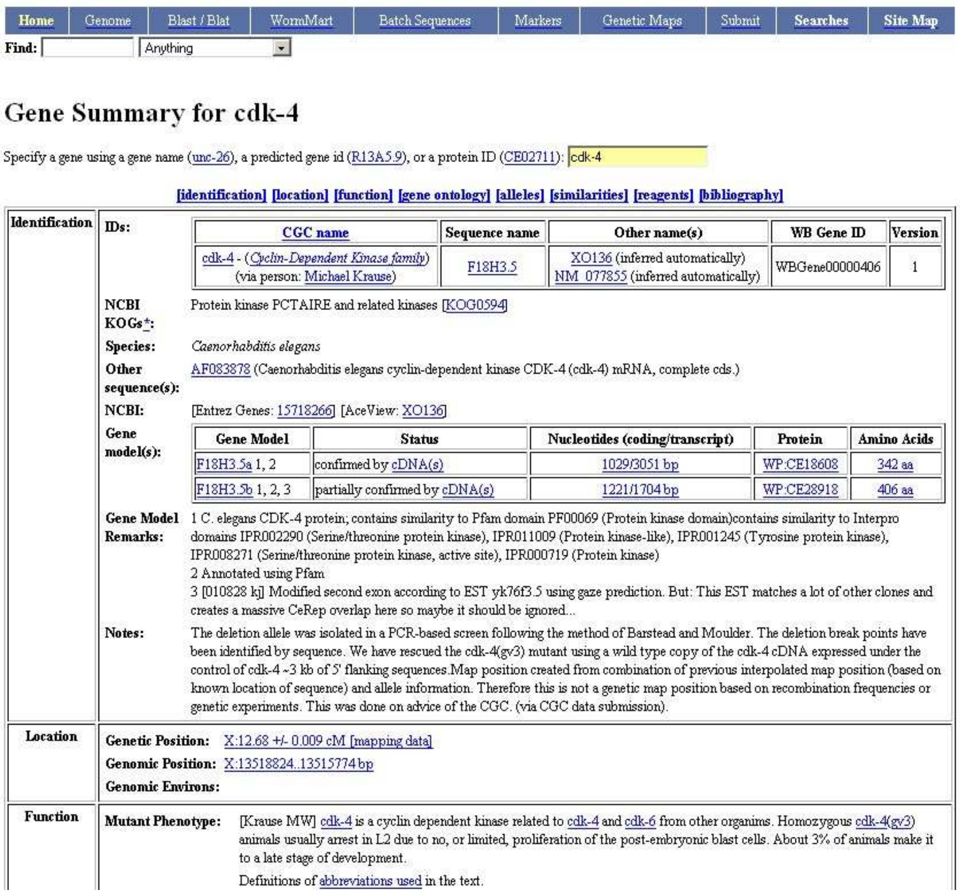
With our implementation of TISP++, we are able to automatically generate ontologies based on tables and annotate information from sibling pages in hidden-web sites with respect to the generated ontologies. The contributions of TISP++ are: (1) it provides a tool to generate ontologies automatically based on interpreted tables; (2) it provides a tool to annotate information from sibling pages automatically; (3) it makes hidden-web information publicly accessible through standard query languages.

We discuss the details of these contributions as follows. Since TISP++ depends on TISP, we first explain how TISP works. Section 3.3 discusses how TISP++ generates ontologies based on interpreted tables. Section 3.4 explains how TISP++ automatically annotates information in sibling tables with respect to generated ontlogies. Section 3.5 shows how users can query the annotated information using the user interface we have developed. Section 3.6 discusses related work. In Section 3.7, we draw conclusions and mention some possibilities for future work.

3.2 TISP

Hidden-web pages are usually generated dynamically from a pre-defined templates in response to submitted queries and are typically presented as tables. Therefore, these tables usually share the same or similar structures. We call pages from the same hidden-web site that have the same or similar structure sibling pages and the corresponding tables sibling tables. For example, the two pages in Figures 3.1 and 3.2 are sibling pages. All the pages from this web site have the same basic structure—the same top banners, the same table title (*Gene Summary for* some particular gene), and a table that contains information about the gene. If we compare the two large tables in the main part of the sibling pages, we can see that the first columns of each table are exactly the same. If we look at the cells under the *Identification* label in the two tables, both contain another tables with two columns. In both cases, the first column contains identical labels *IDs*, *NCBI KOGs*, ..., *Putative ortholog(s)*. Further, the tables under *Identification.IDs* also have identical header rows. The data rows, however, vary considerably.

To interpret a table is to properly associate table category labels with table data values. To interpret hidden-web tables automatically, TISP compares a pair of sibling tables and looks for commonalities for labels and variations for data values. The first column of the large table, for example, is identified as a column of labels, as is the nested table that starts with label *IDs.* Within this nested table are two additional nested tables—one with columns headed by *CGC name*, *Sequence Name*, ..., and one with columns headed by the labels *Gene Model*, *Status*, ... TISP tries to match a table with one of the three pre-defined patterns in Figure 3.3 to find the association of labels and data values. The table that starts with label *CGC name*



Gene Summary for cdk-4

Specify a gene using a gene name (unc-26), a predicted gene id (R13A5.9), or a protein ID (CE02711): cdk-4

[identification] [location] [function] [gene ontology] [alleles] [similarities] [reagents] [bibliography]

Identification

IDs:

CGC name	Sequence name	Other name(s)	WB Gene ID	Version
cdk-4 - (*Cyclin-Dependent Kinase family*) (via person: Michael Krause)	F18H3.5	XO136 (inferred automatically) NM_077855 (inferred automatically)	WBGene00000406	1

NCBI KOGs*: Protein kinase PCTAIRE and related kinases [KOG0594]

Species: *Caenorhabditis elegans*

Other sequence(s): AF083878 (Caenorhabditis elegans cyclin-dependent kinase CDK-4 (cdk-4) mRNA, complete cds.)

NCBI: [Entrez Genes: 15718266] [AceView: XO136]

Gene model(s):

Gene Model	Status	Nucleotides (coding/transcript)	Protein	Amino Acids
F18H3.5a 1, 2	confirmed by cDNA(s)	1029/3051 bp	WP:CE18608	342 aa
F18H3.5b 1, 2, 3	partially confirmed by cDNA(s)	1221/1704 bp	WP:CE28918	406 aa

Gene Model Remarks: 1 C. elegans CDK-4 protein; contains similarity to Pfam domain PF00069 (Protein kinase domain)contains similarity to Interpro domains IPR002290 (Serine/threonine protein kinase), IPR011009 (Protein kinase-like), IPR001245 (Tyrosine protein kinase), IPR008271 (Serine/threonine protein kinase, active site), IPR000719 (Protein kinase)
2 Annotated using Pfam
3 [010828 kj] Modified second exon according to EST yk76f3.5 using gaze prediction. But: This EST matches a lot of other clones and creates a massive CeRep overlap here so maybe it should be ignored...

Notes: The deletion allele was isolated in a PCR-based screen following the method of Barstead and Moulder. The deletion break points have been identified by sequence. We have rescued the cdk-4(gv3) mutant using a wild type copy of the cdk-4 cDNA expressed under the control of cdk-4 ~3 kb of 5' flanking sequences.Map position created from combination of previous interpolated map position (based on known location of sequence) and allele information. Therefore this is not a genetic map position based on recombination frequencies or genetic experiments. This was done on advice of the CGC. (via CGC data submission).

Location

Genetic Position: X:12.68 +/- 0.009 cM [mapping data]

Genomic Position: X:13518824..13515774 bp

Genomic Environs:

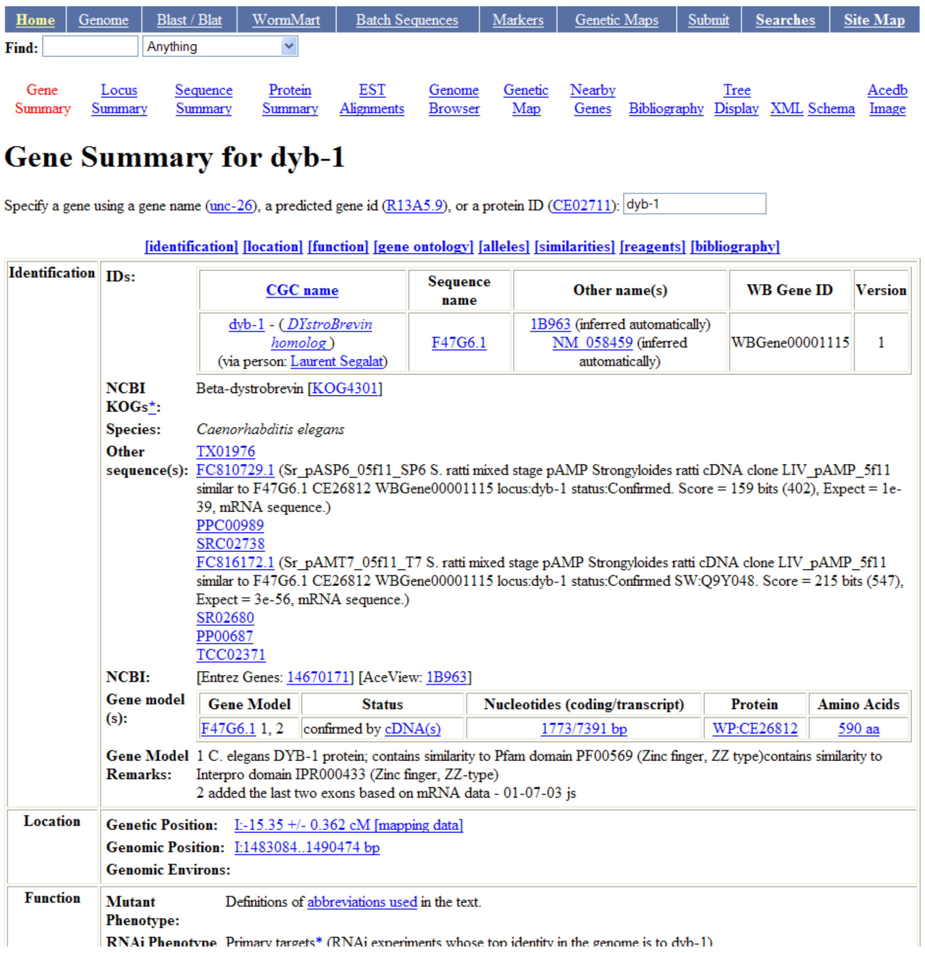
Function

Mutant Phenotype: [Krause MW] cdk-4 is a cyclin dependent kinase related to cdk-4 and cdk-6 from other organims. Homozygous cdk-4(gv3) animals usually arrest in L2 due to no, or limited, proliferation of the post-embryonic blast cells. About 3% of animals make it to a late stage of development.
Definitions of abbreviations used in the text.

Figure 3.1: A Sample Table from WormBase [62].

and the table starts with label *Gene Model*, for example, fallow Pattern 1. They have *n* labels (denoted by {L}) in the first row followed by any number of rows each with *n* data values (denoted by {V}). The table that starts with *Identification* and the label that starts with *IDs* follow Pattern 2—on each row they have a label followed by n values (often, as the case here, *n*=1). Pattern 3 is for tables with labels both across the top row and also down the left column. Combinations of these patterns are also possible. Table with two columns of label-value pairs, for example, are quite common.

With an interpreted pattern, TISP can automatically interpret the sibling tables in the remaining sibling pages on a hidden-web site. TISP can also dynamically



Home	Genome	Blast / Blat	WormMart	Batch Sequences	Markers	Genetic Maps	Submit	Searches	Site Map

Find: Anything

Gene Summary	Locus Summary	Sequence Summary	Protein Summary	EST Alignments	Genome Browser	Genetic Map	Nearby Genes	Bibliography	Tree Display	XML Schema	Acedb Image

Gene Summary for dyb-1

Specify a gene using a gene name (unc-26), a predicted gene id (R13A5.9), or a protein ID (CE02711): dyb-1

[identification] [location] [function] [gene ontology] [alleles] [similarities] [reagents] [bibliography]

Identification

IDs:

CGC name	Sequence name	Other name(s)	WB Gene ID	Version
dyb-1 - (*DYstroBrevin homolog*) (via person: Laurent Segalat)	F47G6.1	1B963 (inferred automatically) NM_058459 (inferred automatically)	WBGene00001115	1

NCBI KOGs*:	Beta-dystrobrevin [KOG4301]
Species:	*Caenorhabditis elegans*
Other sequence(s):	TX01976 FC810729.1 (Sr_pASP6_05f11_SP6 S. ratti mixed stage pAMP Strongyloides ratti cDNA clone LIV_pAMP_5f11 similar to F47G6.1 CE26812 WBGene00001115 locus:dyb-1 status:Confirmed. Score = 159 bits (402), Expect = 1e-39, mRNA sequence.) PPC00989 SRC02738 FC816172.1 (Sr_pAMT7_05f11_T7 S. ratti mixed stage pAMP Strongyloides ratti cDNA clone LIV_pAMP_5f11 similar to F47G6.1 CE26812 WBGene00001115 locus:dyb-1 status:Confirmed SW:Q9Y048. Score = 215 bits (547), Expect = 3e-56, mRNA sequence.) SR02680 PP00687 TCC02371
NCBI:	[Entrez Genes: 14670171] [AceView: 1B963]

Gene model (s):

Gene Model	Status	Nucleotides (coding/transcript)	Protein	Amino Acids
F47G6.1 1, 2	confirmed by cDNA(s)	1773/7391 bp	WP:CE26812	590 aa

Gene Model Remarks: 1 C. elegans DYB-1 protein; contains similarity to Pfam domain PF00569 (Zinc finger, ZZ type)contains similarity to Interpro domain IPR000433 (Zinc finger, ZZ-type)
2 added the last two exons based on mRNA data - 01-07-03 js

Location

Genetic Position: I:-15.35 +/- 0.362 cM [mapping data]
Genomic Position: I:1483084..1490474 bp
Genomic Environs:

Function

Mutant Phenotype: Definitions of abbreviations used in the text.
RNAi Phenotype Primary targets* (RNAi experiments whose top identity in the genome is to dyb-1)

Figure 3.2: A Second Sample Table from WormBase.

adjust an interpreted table pattern if the system encounters a sibling table with slight variation from the pattern, such as an extra column or a missing column.

3.3 Semantic Ontology Generation

TISP++ conceptualizes the information present in the tables by generating an ontology according to the structure pattern given by TISP. TISP++ uses the structure pattern in a tables as discovered by TISP to generate OWL classes, properties, and constraints according to the structure pattern. It then uses Jena [33], a semantic web framework for Java, to output the OWL ontology.

Pattern 1:

$< table > (< tbody >)? < tr > (< (td|th) > \{L\})^n \ (< tr > (< (td|th) > \{V\})^n)^+$

Pattern 2:

$< table > (< tbody >)?(< tr >< (td|th) > \{L\}(< (td|th) > \{V\})^n)^+$

Pattern 3:

$< table > (< tbody >)? < tr > (< (td|th) > \{L\})^n$
$(< tr >< (td|th) > \{L\}(< (td|th) > \{V\})^{(n-1)})^+$

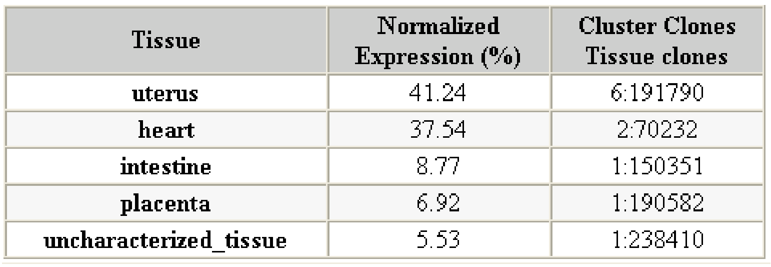
Figure 3.3: Some Basic Pre-defined Pattern Templates.

Figure 3.4 shows part of the ontology for the sibling pages interpreted by TISP from the WormBase gene repository [62] (the repository containing the pages in Figure 3.1 and 3.2). As a default, TISP++ selects the site name, "WormBase", for the ontology. This name, albeit in lower-case letters to conform to conventions becomes the name space for the ontology as Line 4 in Figure 3.4. TISP++ also uses this name to describe the generated ontology as Lines 7 and 8 in Figure 3.4 show. In addition, and most important, this name provides an anchor class to which we attach ontological concepts. Line 10 in Figure 3.4 shows the OWL class "WormBase".

For each table label, TISP++ generates an OWL class. The label name becomes the class name. To satisfy the OWL syntax, however, TISP++ elides illegal characters such as spaces and parentheses. Thus "Gene model(s):" becomes "Genemodels" as Line 13 in Figure 3.4 shows. The generated ontology also represents the relationships among the labels. TISP++ generates relationships according to the structure patterns in Figure 3.3. For Pattern 1 and Pattern 2, each value has only one associated label, and each label has only one parent label. Thus, these patterns require only binary relationships and relationship generation is straightforward. For a binary relationship between two classes A and B, TISP++ generates an OWL object property: A-B and its inverse B-A. For the property A-B, TISP++ defines A as the domain and B as the range. For example, Lines 17–23 in Figure 3.4 show the OWL object property for *WormBase-Identification.* If a label is paired with an actual value such as are the labels *Gene Model* and *Amino Acids* in Figures 3.1 and 3.2,

```
<rdf:RDF
        xmlns:owl=“http://www.w3.org/2002/07/owl#”
        xmlns:rdfs=“http://www.w3.org/2000/01/rdf-schema#”
        xmlns:base =“http://dithers.cs.byu.edu/owl/ontologies/wormbase#”... >
...
    <owl:Ontology rdf:about=“”>
        <rdfs:comment>OWL Ontology for WormBase</rdfs:comment>
        <rdfs:label>WormBase Ontology</rdfs:label>
    </owl:Ontology>
    <owl:Class rdf:ID=“WormBase”/>
    <owl:Class rdf:ID=“Identification”/>
    ...
    <owl:Class rdf:ID=“Genemodels”/>
    ...
    <owl:Class rdf:ID=“AminoAcids”/>
    ...
    <owl:ObjectProperty rdf:ID=“WormBase-Identification”>
        <owl:inverseOf>
            <owl:ObjectProperty rdf:ID=“Identification-WormBase”>
        </owl:inverseOf>
        <rdfs:domain rdf:resource=“#WormBase”/>
        <rdfs:range rdf:resource=“#Identification”/>
    </owl:ObjectProperty>
    ...
    <owl:ObjectProperty rdf:ID=“Identification-IDs”>
    ...
    <owl:ObjectProperty rdf:ID=“Identification-Genemodels”>
    ...
    <owl:ObjectProperty rdf:ID=“Genemodels-AminoAcids”>
...
    <owl:DatatypeProperty rdf:ID=“AminoAcidsValue”>
        <rdfs:range rdf:resource=“xsd;string”/>
        <rdfs:domain rdf:resource=“#AminoAcids”/>
    </owl:DatatypeProperty>
...
</rdf:RDF>
```

Figure 3.4: Partial OWL Ontology for WormBase.



Tissue	Normalized Expression (%)	Cluster Clones Tissue clones
uterus	41.24	6:191790
heart	37.54	2:70232
intestine	8.77	1:150351
placenta	6.92	1:190582
uncharacterized_tissue	5.53	1:238410

Figure 3.5: An Example Table with Pattern 3.

TISP++ generates an OWL data type property for the OWL class associated with this label. For example, data type property *AminoAcidsValue* describes the actual value for *AminoAcids*. As Lines 31–34 in Figure 3.4 show, its domain is *AminoAcids* and its range is string, by default.

For Pattern 3, each value has two associated labels. Figure 3.5 shows an example: the value *41.24* has the labels *Normalized Expression (%)* and *uterus*. Thus, the pattern requires a ternary relationship. Since OWL ontologies only allow binary relationships, we transform Pattern 3 as follows. In Figure 3.3, we consider the label L in the pattern "$(< tr >< (td|th) > \{L\}(< (td|th) > \{V\}^{(n-1)})^{+}$" as a value V. The pattern then becomes "$(< tr >< (td|th) >$\{V\}$(< (td|th) > \{V\}^{(n-1)})^{+}$" and can be further simplified to "$(< tr > (< (td|th) > \{V\})^{n})^{+}$", which is the same as Pattern 1. In Figure 3.5, for example, the contents in the first row and the first column are all labels. When TISP++ generates the ontology for this table, it considers the contents in the first row as labels, but considers the contents in the first column as values (except for the first one "Tissue" since it is in the first row). Therefore TISP++ can transform ternary relationships to binary relationships and then translate them as binary relationships to OWL.

3.4 Semantic Annotation

After TISP++ generates an ontology according to the structure pattern of a web repository, it automatically annotates the pages from this repository with respect

```
<rdf:RDF
            xmlns:wormbase="http://www.deg.byu.edu/owl/ontologies/wormbase#"
            xmlns:ann="http://www.deg.byu.edu/owl/ontologies/annotation#"
            xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    ...
    <wormbase:WormBase rdf:ID="WormBase_1">
            <wormbase:WormBase-Identification rdf:resource="#Identification_1"/>
            <wormbase:WormBase-Location rdf:resource="#Location_1"/>
            ...
    </wormbase:WormBase>
    <wormbase:Identification rdf:ID="Identification_1">
            <wormbase:Identification-IDs rdf:resource="#IDs_1"/>
            ...
            <wormbase:Identification-Genemodels rdf:resource="#Genemodels_1"/>
            <wormbase:Identification-Genemodels rdf:resource="#Genemodels_2"/>
            ...
    </wormbase:Identification>
    <wormbase:IDs rdf:ID="IDs_1">
            <wormbase:IDs-CGCname rdf:resource="#CGCname_1"/>
            <wormbase:IDs-Sequencename rdf:resource="#Sequencename_1"/>
            ...
    </wormbase:IDs>
    ...
    <wormbase:Genemodels rdf:ID="Genemodels_1">
            ...
            <wormbase:Genemodels-Protein rdf:resource="#Protein_1"/>
            <wormbase:Genemodels-AminoAcids rdf:resource="#AminoAcids_1"/>
    </wormbase:Genemodels>
    <wormbase:Genemodels rdf:ID="Genemodels_2">
            ...
            <wormbase:Genemodels-Protein rdf:resource="#Protein_2"/>
            <wormbase:Genemodels-AminoAcids rdf:resource="#AminoAcids_2"/>
    </wormbase:Genemodels>
    ...
    <wormbase:CGCname rdf:ID="CGCname_1">
            <ann:OffsetOnHTMLPage/>5951< /ann:OffsetOnHTMLPage/>
            <ann:HTMLText/>cdk-4 ... < /ann:HTMLText/>
            <wormbase:CGCnameValue>cdk-4 ... < /wormbase:ProteinValue>
            <wormbase:CGCname-IDs rdf:resource="#IDs_1"/>
    </wormbase:CGCname>
    ...
    <wormbase:Protein rdf:ID="Protein_1">
            <ann:OffsetOnHTMLPage/>10015< /ann:OffsetOnHTMLPage/>
            <ann:HTMLText/>WP:CE18608< /ann:HTMLText/>
            <wormbase:ProteinValue>WP:CE18608< /wormbase:ProteinValue>
            <wormbase:Protein-Genemodels rdf:resource="#Genemodels_1"/>
    </wormbase:Protein>
    <wormbase:AminoAcids rdf:ID="AminoAcids_1">
            <ann:OffsetOnHTMLPage/>10152< /ann:OffsetOnHTMLPage/>
            <ann:HTMLText/>342 aa< /ann:HTMLText/>
            <wormbase:AminoAcidsValue>342 aa</wormbase:ProteinValue>
            <wormbase:AminoAcids-Genemodels rdf:resource="#Genemodels_1"/>
    </wormbase:AminoAcids>
    ...
    <wormbase:Protein rdf:ID="Protein_2">
            <ann:OffsetOnHTMLPage/>10689< /ann:OffsetOnHTMLPage/>
            <ann:HTMLText/>WP:CE28918</ann:HTMLText/>
            <wormbase:ProteinValue>WP:CE28918</wormbase:ProteinValue>
            <wormbase:Protein-Genemodels rdf:resource="#Genemodels_2"/>
    </wormbase:Protein>
    <wormbase:AminoAcids rdf:ID="AminoAcids_2">
            <ann:OffsetOnHTMLPage/>10826< /ann:OffsetOnHTMLPage/>
            <ann:HTMLText/>406 aa< /ann:HTMLText/>
            <wormbase:AminoAcidsValue>406 aa</wormbase:ProteinValue>
            <wormbase:AminoAcids-Genemodels rdf:resource="#Genemodels_2"/>
    </wormbase:AminoAcids>
...
</rdf:RDF>
```

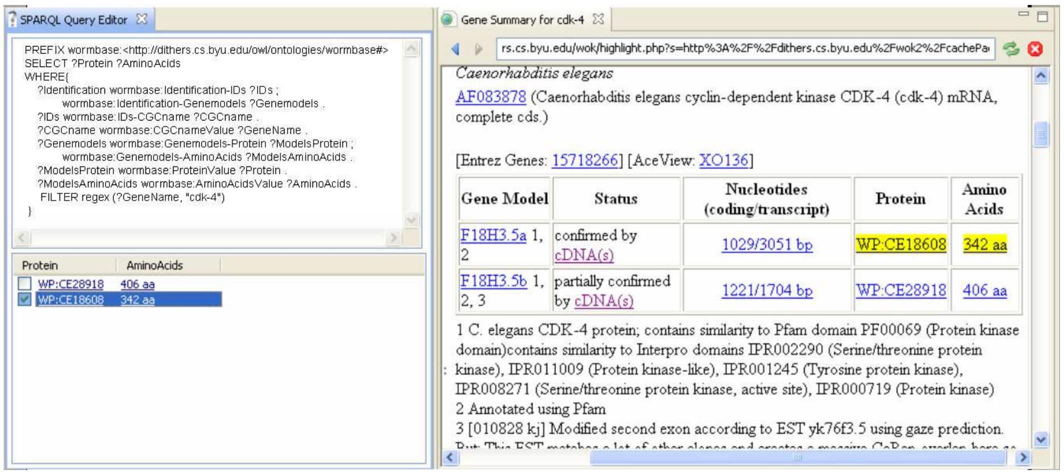
Figure 3.6: Annotation for Figure 2.1 for the Ontology in Figure 3.4.

to the generated ontology. Figure 3.6 shows a portion of the RDF file generated from the page in Figure 2.1. In an RDF annotation file, we first declare the name spaces of referenced ontologies. Line 2 in Figure 3.6 refers to the *wormbase* ontology in Figure 3.4, and Lines 3 and 4 declare additional name spaces. The *annotation* name space gives the meaning of the annotation declarations in the RDF file, and *22-rdf-syntax-ns* name space gives the meaning of the rdf declarations.

In Line 6, the URI instance *WormBase_1*, which refers to the whole table, is a URI instance for the *WormBase* class in the *wormbase* ontology in Figure 3.4. In Line 11, the URI instance *Identification_1*, which refers to the value of the label *Identification* in Figure 3.1, is a URI instance of the class *Identification* in the *wormbase* ontology. Since the table with labels *Gene Model*, *Status*, *Nucleotides (coding/trancript)*, *Protein*, and *Amino Acids* in Figure 3.1 has two data rows, TISP++ declares URI instances: *Genemodels_1* and *Genemodels_2*, one for each row (Lines 14 and 15). TISP++ also declares the relationship between two URI instances. For example, TISP++ declares the relationship *WormBase_1-Identification_1* in Lines 6 and 7. TISP++ also declares the relationships between instances and annotated values. Annotated values appear as themselves, tagged with both *HTMLText* and with their appropriate *<label>Value* (For string types the HTML text and the value are often the same. But for data types like *xsd:boolean*, the HTML text could be "True" or "False", but its value in the value space should be "T" or "F"). For example, Line 44 gives the *HTMLText* for the protein value "WP:CE18608", and Line 45 tells its *ProteinValue*, also "WP:CE18608". To identify the annotated string in the original page, TISP++ keeps track of the position where the values are located by recording the offset of the string from the beginning of the cached page, a local copy of the original page containing the values. Line 43 in Figure 3.6, for example, shows that we can locate the value "WP:CE18608" at character 10015 in the cached HTML document.

3.5 Semantic Querying

With the annotated data properly stored in an RDF file (e.g. Figure 3.6), we are now ready to query the annotated data using SPARQL [48]. A simple query illustrates how this works. If we want to find the protein and the animo-acids information for gene “cdk-4”, we can write the SPARQL query in Figure 3.7. Figure 3.7 shows the query and the result of this query.



```
PREFIX wormbase:<http://dithers.cs.byu.edu/owl/ontologies/wormbase#>
SELECT ?Protein ?AminoAcids
WHERE{
   ?Identification wormbase:Identification-IDs ?IDs ;
         wormbase:Identification-Genemodels ?Genemodels .
   ?IDs wormbase:IDs-CGCname ?CGCname .
   ?CGCname wormbase:CGCnameValue ?GeneName .
   ?Genemodels wormbase:Genemodels-Protein ?ModelsProtein ;
         wormbase:Genemodels-AminoAcids ?ModelsAminoAcids .
   ?ModelsProtein wormbase:ProteinValue ?Protein .
   ?ModelsAminoAcids wormbase:AminoAcidsValue ?AminoAcids .
    FILTER regex (?GeneName, "cdk-4")
 }
```

Protein	AminoAcids
WP:CE28918	406 aa
✓ WP:CE18608	342 aa

rs.cs.byu.edu/wok/highlight.php?s=http%3A%2F%2Fdithers.cs.byu.edu%2Fwok2%2FcachePa

Caenorhabditis elegans
AF083878 (Caenorhabditis elegans cyclin-dependent kinase CDK-4 (cdk-4) mRNA, complete cds.)

[Entrez Genes: 15718266] [AceView: XO136]

Gene Model	Status	Nucleotides (coding/transcript)	Protein	Amino Acids
F18H3.5a 1, 2	confirmed by cDNA(s)	1029/3051 bp	WP:CE18608	342 aa
F18H3.5b 1, 2, 3	partially confirmed by cDNA(s)	1221/1704 bp	WP:CE28918	406 aa

1 C. elegans CDK-4 protein; contains similarity to Pfam domain PF00069 (Protein kinase domain)contains similarity to Interpro domains IPR002290 (Serine/threonine protein kinase), IPR011009 (Protein kinase-like), IPR001245 (Tyrosine protein kinase), IPR008271 (Serine/threonine protein kinase, active site), IPR000719 (Protein kinase)
2 Annotated using Pfam
3 [010828 kj] Modified second exon according to EST yk76f3.5 using gaze prediction.

Figure 3.7: Sample SPARQL Query and Results for the Annotation in Figure 3.6

The *FILTER* statement in the SPARQL query allows the variable *?GeneName* to bind only to values containing string “cdk-4”. The *CGCnameValue* in Line 38 of Figure 3.6 satisfies this constraint. Through the property *wormbase:CGCnameValue* (Line 35), the SPARQL query associates *?GeneName* values with *?CGCname* instance values. Thus, in our example, the instance value *CGCname_1* (Line 35) binds to the variable *?CGCname*. Then, through the property *wormbase*:*IDs-CGCname*, the query finds the corresponding instances of *IDs* (in our example, *IDs_1* in Line 18). Next, through the *wormbase*:*Identification-IDs* property, the query finds the instances of *Identification* that associate with *IDs_1* (in our example, *Identification_1* in Line 11). Thereafter, through the *wormbase*:*Identification-Genemodels* property, the query finds the instances *Genemodels_1* and *Genemodels_2* (Lines 14 and 15). The query then locates the result instances, finding *Protein_1*, *Protein_2*, *AminoAcids_1*, and *AminoAcids_2* through the properties *wormbase*:*Genemodels-Protein* and *wormbase*:*Genemodels-AminoAcids* (Lines 26, 31, 27, and 32). Finally, the query returns the result values for these instances through the properties *wormbase*:*ProteinValue* and *wormbase*:*AminoAcidsValue* (Lines 45, 58, 51, and 64).

A user can select one or more returned records by checking the corresponding check boxes. TISP++ then shows the user the original source page with the values of interest highlighted. Figure 3.7 shows that we have selected the check box of the second record in the results. Thus, in the displayed page the values *WP:CE18608* and *342 aa* are highlighted. In addition to allowing a user to select a record of values, TISP++ also makes all values individually clickable. When a user clicks on an individual value, TISP++ displays the source page with the corresponding value highlighted.

As a result of TIPS++ processing, which includes table understanding, ontology generation, and semantic annotation, we make hidden web data present in HTML tables completely accessible by a standard query system. In addition, TISP++ also semantically annotates the data with respect to the generated OWL ontology, so that the data becomes machine-understandable and automatically manipulable by computer agents.

3.6 Related Work

3.6.1 Ontology Generation

In recent years, many researchers have tried to facilitate ontology generation. Manual editing tools such as Protégé [41] and OntoWeb [49] have been developed to help users create and edit ontologies. It is not trivial, however, to learn ontology modeling languages and complex tools in order to manually create ontological description for information repositories.

Because of the difficulties involved in manual creation, researchers have developed semi-automatic ontology generation tools. Most efforts so far have been devoted to automatic generation of ontologies from text files. Tools such as OntoLT [7], Text2Onto [11], OntoLearn [40], and KASO [61] use machine learning methods to generate an ontology from arbitrary text files. These tools usually require a large training corpus and use various natural language processing algorithms to derive features to learn ontologies. The results, however, are not very satisfactory [42].

Tools such as SIH [53], TANGO [56], and the one developed by Pivk [42] use structured information (HTML tables) as a source for learning ontologies. Structured information makes it easier to interpret new items and relations. The approach in [42] tries to discover semantic labels for table regions and generate an ontology based on a table's structure. But how this process is done and what format the generated ontologies have is not discussed in the paper. SIH [53] and TANGO [56] are two ongoing projects we are currently working on. SIH tries to generate user-specified ontologies depending on user-generated forms. TANGO generates ontologies by analyzing related tables in a specific domain, generating an ontology according to each table, and then merging these ontologies to a general ontology for the domain. TISP++ can generate OWL ontologies fully automatically. It, however, only generates an ontology for a single set of sibling pages. It does not merge ontologies generated from different web sites, nor does it provide for user-specified ontologies. In addition, TISP++ generates ontologies in only one simple way, while TANGO aims at generating more sophisticated ontologies.

3.6.2 Semantic Annotation

Existing semantic annotation systems can be classified into pattern-based systems and machine learning-based systems. Pattern-based systems such PANKOW [10] and Armadillo [17] find entities by discovering patterns. The pattern are either discovered manually or induced semi-automatically with a set of initial manually tagged seed patterns. Systems such as SemTag [14], AeroDAML [36], and KIM [44] use a set of pre-defined rules to locate the information of interest. OWL-AA [15, 16] uses a domain-specified extraction ontology to locate semantic entities. Systems such as S-CREAM [30] and MnM [57] use machine learning algorithms and natural language processing methods to locate semantic entities. All of these approaches require some pre-defined information. Pattern-based approaches need a set of initial seed patterns. Rule-based approaches need a set of pre-defined rules. Extraction-ontology-based approaches need domain ontologies. And machine learning-based approaches need a training corpus. TISP++, however, does not require a training corpus or pre-defined domain knowledge and relies only on the definition of a few typical table-pattern templates, which are all domain-independent.

3.7 Concluding Remarks

TISP++ uses TISP to semantically annotate web pages, turning their embedded facts into externally accessible facts. Given an interpreted table, TISP++ automatically generates an OWL ontology depending on the table's structure and then semantically annotates the data in the table with respect to this generated ontology. By doing so, all the data present in the sibling tables becomes accessible through a standard query interface.

Several directions remain to be pursued. For TISP++ and semantic ontology generation and semantic annotation, we would like to do the following. (1) It would be helpful if we could generate more sophisticated ontologies that cover more complex semantics such as n-ary relationship sets, generalization/specialization, and aggregation. This will likely require us to combine TISP with TAT [34], which abstracts tables into a rich encoding of table features based on Wang notation, and to combine TISP and TAT with MOGO [39], which uses rich table encodings and available semantic encodings to discover n-ary relationship sets, generalization/specialization, and aggregation. (2) TISP++ harvests information only with respect to the ontological view of the hidden-web site on which it is applied. We plan to allow users to harvest information from multiple hidden-web sites according to a personalized view. Our users would have the option to choose what data in a table they want to include and how this data should be organized. This involves matching user-specific views with TISP-generated table patterns. (3) Average web users need a more user-friendly query system, so that they can find data of interest without knowing SPARQL. We plan to provide our users with a natural-language-based query interface (e.g. like [3]) and a form-based query interface (e.g. like [20]).

Chapter 4

Form-based Ontology Creation and Information Harvesting

4.1 Introduction

Web designers use their own views to present online information. A web site can provide a large amount of information about a topic in great detail. But sometimes a user only needs a relatively small and specific part of the information. Web resources typically do not allow users to query with their own view or even do not allow for user queries at all. Users have to browse the pages, filter out the unneeded information, and find the information of interest. In another related scenario, the information a user wants may be spread across several different sources. No source, by itself, is capable of providing the information. In this case, users often need to search several online repositories and gather information of interest manually. Both of these tasks are tedious and time-consuming.

More and more researchers have realized the importance of information gathering for the purpose of personalization [8, 35]. In [35], Kapetanios proposed personal contents and collective intelligence, which coincides with our personalized ontology creation and information harvesting idea. We take into consideration both a user's contributions to knowledge creation and sharing, and the computer's role in facilitating the knowledge sharing and learning process in collaborative environments. We call our system FOCIH (Form-based Ontology Creation and Information Harvester, pronounced *foh*·sī). FOCIH is a system that can harvest information for users from different hidden web sites with respect to user-specific views, specified by ordinary forms.

Our first step toward personalized information harvesting is to provide users with a tool with which they can give their view of a domain without knowledge of conceptual modeling or ontology languages. We observe that forms are a natural way for humans to collect information. As an everyday activity, people create forms and ask others to fill in the form. In this way, specified information can be gathered. Inspired by this observation, we designed and implemented FOCIH, which allows a user to generate a form that describes the information a user wishes to harvest.

The next step is to generate ontologies according to user-created forms. The form labels become concepts in an ontology. Then based on the structure of different form components, FOCIH generates different kinds of relationship sets and constraints. Also as an interesting and convenient way to initiate forms, we reversed ontology generation from forms so that FOCIH can generate forms from ontologies (limited currently to the case of TISP-generated ontologies [51]). Users can then alter the form, if desired, to obtain a personalized view via the form.

The final step is to annotate and harvest information with respect to the view represented by the form. We harvest from pages, all from the same hidden-web site. After creating a form, a user finds a sample page from a web-site of interest and highlights and fills in the information of interest from the sample page into the form. The user filled-in values provide FOCIH with valuable information about how to locate source information in the sample document. Using location patterns, FOCIH is able to harvest information from other sibling pages from the same hidden-web site automatically. It thus annotates all the information on these pages with respect to the ontology. Once the pages are annotated, users can query the information. In our implementation, users can query generated RDF tuples with SPARQL queries [48] or with free-form queries [58].

With our implementation of FOCIH, users are able to create forms, generate ontologies based on forms, and harvest information from sibling pages in hidden-web sites. The contributions of FOCIH are: (1) it provides users with a way to create an ontology without knowing conceptual modeling or ontology languages; (2) it provides a tool to generate ontologies automatically based on forms; (3) for some cases, it initializes forms for users; and (4) it facilities automatic user-specific information harvesting and annotation from tables in sibling pages in hidden-web sites.

We present the details of these contributions as follows. Section 4.2 discusses related work for ontology generation as well as personalized ontologies and information gathering. Section 4.3 introduces OSM which we use as the target ontology language into which FOCIH forms are cast. Section 4.4 describes how to use the FOCIH GUI to create a form that describes a user's view of a domain and how to annotate a sample page in that domain with respect to the user's view by filling out the created form. Section 4.5 explains how FOCIH generates ontologies based on user-created forms. Section 4.6 discusses how FOCIH determines path and instance recognition which allows it to harvest information with respect to the created form. Section 4.7 describes how FOCIH harvests information from multiple sites and semantically annotates the harvested information with respect to personalized views. Section 4.8 shows how FOCIH can create initial form views for users based on TISP interpreted hidden-web site. It also illustrates a scenario in which a user initializes a form from a hidden-web site, customizes it for personal use, and then uses it to harvest information from other sites as well as the site from which the form originated. In Section 4.9, we make concluding remarks.

4.2 Related Work

We know of no system that allows both for ease of personal ontology creation and for search and retrieval of facts in documents with respect to a personalized ontology.

Researchers, however, have shown much interest in easying the burden of ontology creation and in personalized information gathering with respect to an ontology.

4.2.1 Ontology Creation

In recent years, many researchers have tried to facilitate ontology generation. Manual editing tools such as Protege [41] and OntoWeb [49] have been developed to help users create and edit ontologies. It is far from trivial, however, to learn ontology modeling languages and complex tools in order to manually create ontological descriptions for information repositories.

Because of the difficulties involved in manual creation, researchers have developed semi-automatic ontology generation tools. Most efforts so far have been devoted to automatic generation of ontologies from text files. Tools such as OntoLT [7], Text2Onto [11], OntoLearn [40], and KASO [61] use machine learning methods to generate an ontology from arbitrary text files. These tools usually require a large training corpus and use various natural language processing algorithms to derive features to learn ontologies. The results, however, are not very satisfactory [42]. Tools such as OntoBuilder [26], TANGO [56], the one developed by Pivk [42], and the one introduced in [5], use structured information (HTML tables and forms) as a source for learning ontologies semi-automatically from forms. Structured information makes it easier to interpret new items and relations. These approaches, however, derive concepts and relationships among concepts from source data, not from users. FOCIH, on the other hand, allows users to provide their own views and generate user-specified ontologies.

4.2.2 Personalized Ontologies and Personalized Information Gathering

Much research has been conducted on personalized information gathering and search based on ontologies. Most of the these approaches focus on information retrieval.

They try to retrieve documents that satisfy users' needs based on ontology models that describe user interests. Approaches such as [46, 54, 65] use personalized ontologies as training sets for information-retrieval techniques that retrieve documents of interest. In [45], the authors propose an approach for personalized search by matching personalized ontologies with page ontologies. Their system generates personalized ontologies based on user browsing patterns. By matching a user ontology with a page ontology, the system can retrieve more semantically relevant documents. WEBCLUSTERS [47] classifies search results into a meaningful hierarchy of topics based on a taxonomic ontology that represents the perspective of the user performing the search. The personalized ontologies are usually created manually by experts or interactively with users via a knowledge base. These systems, however, only return documents and cannot return actual facts. Users still have to traverse the returned the documents to get the information of interest.

4.3 OSM Ontologies

We use OSM [21] as the semantic data model for an extraction ontology. The advantage of OSM is that it has a high-level graphical representation that directly translates to predicate calculus. Thus, when appropriately limited, it translates in a straightforward way to OWL [2] and to various description logics [48]. Even more important than these advantages, however, an OSM ontology can support data extraction from source documents [21, 24].

Figure 4.1 shows a graphical view of a sample ontology. The components of OSM include object sets, relationship sets, and constraints over these object and relationship sets. An object set in an OSM ontology represents a set of objects which may either be lexical or non-lexical. A dashed box represents a lexical object set and a solid box represents a non-lexical object set. A lexical object set contains concrete values. For example, "Moscow" is a possible value of the *Capital* object set in Figure 4.1. A non-lexical object set describes an abstract concept, such as *Country* in Figure 4.1. Object identifiers represent non-lexical objects.

Lines among object sets represent the relationship sets among them. An arrow indicates functional from domain to range. For example, if we wish to assert that a *Country* can have at most one *Name*, we make the relationship set functional from *Country* to *Name*. A small circle at one end of a line indicates optional, meaning that objects in the connecting object set may or may not participate in relationship for the relationship set. For example, a *Country* may or may not have Geographic Coordinators given for it. OSM also supports n-ary relationships with multiple lines connecting the object sets involved. For example, *Country*, *Population*, and *Year* constitute a ternary relationship set.

In addition to constraints on relationship sets, OSM also supports is-a relationships, also called generalization/specilization. Graphically displayed, a triangle represents generalization/specialization with the generalization connected to an apex of the triangle and the specializations connected to the opposite base. In Figure 4.1, *Male Life Expectancy* and *Female Life Expectancy* are two kinds of specializations of *Life Expectancy*. Constraints on generalization/specialization are also possible. A plus symbol in the triangle would mean that the specializations are mutually exclusive; a union symbol would mean that the generalization is a union of the specializations; together a plus symbol and union symbol would mean that the specializations partition the generalization.

4.4 Form Creation and Annotation

We have a proof-of-concept GUI for FOCIH that runs online.[1] This GUI has two modes of operation: form creation and form annotation. The form-creation mode allows users to create forms with different structures based on the way they want

[1]http://www.deg.byu.edu/wok/formBuilder.php

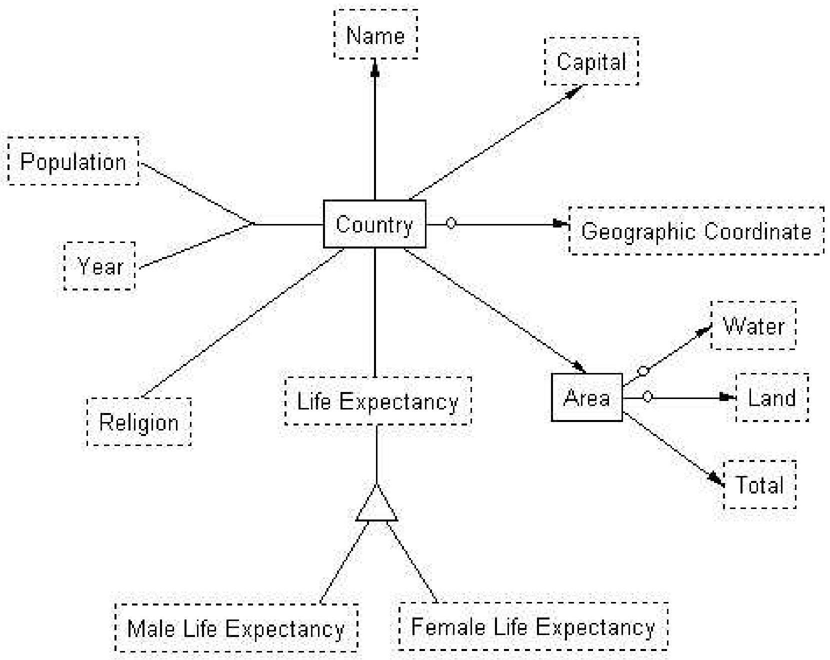
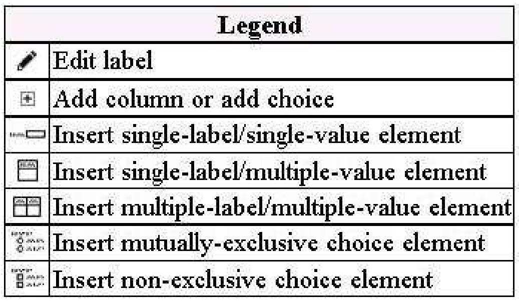


Figure 4.1: The Graphical View of a Sample Ontology

to organize their information. The form-annotation mode allows users to annotate pages with respect to created forms.

4.4.1 Form Creation

The form-creation mode provides users with an intuitive method for defining different kinds of form features. FOCIH has five basic form elements from which users can choose: *single-label/single-value element*, *single-label/multiple-value element*, *multiple-label/multiple-value element*, *mutually-exclusive choice element*, and *non-exclusive choice element*. Figure 4.2 shows the legend of buttons for the FOCIH GUI. Initially, a user starts with a blank form with an empty title. The set of insert icon buttons appears inside the blank form and the edit-label icon appears in the empty title. By clicking on the edit-label icon, a user can add or edit the title. By clicking on one of the icons in the blank form, the user can add new form elements to the form. When a user clicks on , for example, a single-label/multiple-value



Legend	
✎	Edit label
⊞	Add column or add choice
▭	Insert single-label/single-value element
▤	Insert single-label/multiple-value element
⊞	Insert multiple-label/multiple-value element
○	Insert mutually-exclusive choice element
☐	Insert non-exclusive choice element

Figure 4.2: The Legend for FOCIH GUI Form Creation Mode

element appears. The insert icon set then appears below the single-label/multiple-value element so that a user can add additional form elements. The icon set also appears inside the single-label/multiple-value element, so that a user can add other form elements nested inside the single-label/multiple-value element. By clicking on ⊞, a user can add additional columns to a multiple-column element and additional choices to a choice element.

Figure 4.3 shows an example of form creation. Suppose we are interested in basic information about countries (their names, locations, populations, etc.). In our example, we choose *Country* as the base-form title. In our view, we want each country to have one name, capital, and geographic coordinate, so we add three single-label/single-value elements to the form and label them *Name*, *Capital*, and *Geographic Coordinate*. Since we know there might be one or more religions in a country, we choose to use a single-label/mulitple-value form element and label it *Religion*. We want to keep track of population of a *Country* for each of several years. Therefore, we create a multiple-label/multiple-entry field as Figure 4.3 shows. We are also interested in the life expectancy for people in each country depending on the gender. Since the same life-expectancy values can be for either gender, we use a non-exclusive choice form element under *Life Expectancy* and make the choices to be *Male Life Expectancy* and *Female Life Expectancy*. *Land*, *Water*, and *Total Area* are also of interest. Each *Country* has an *Area*, and the *Area* has the three properties *Land*, *Water*, and *Total*.

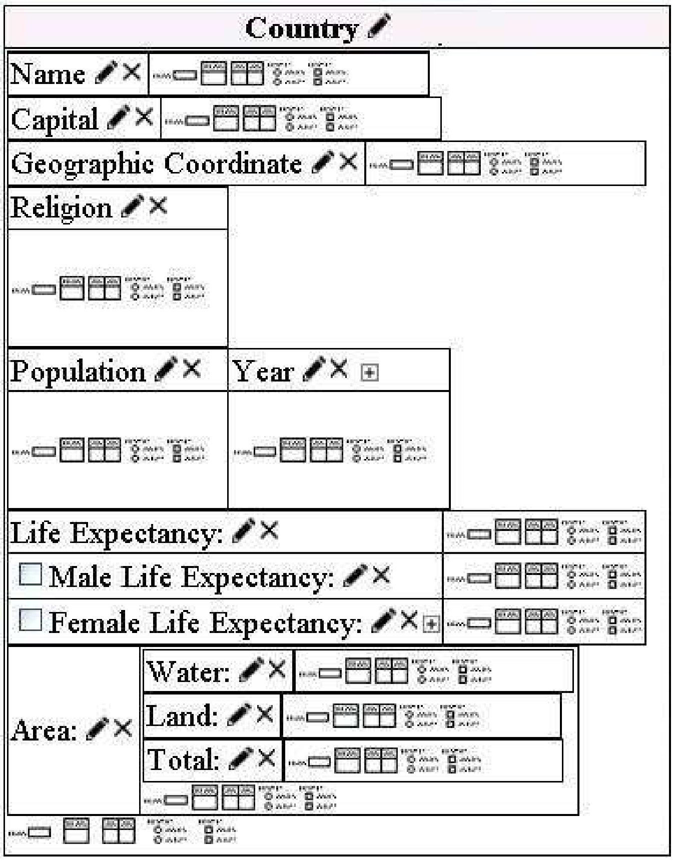


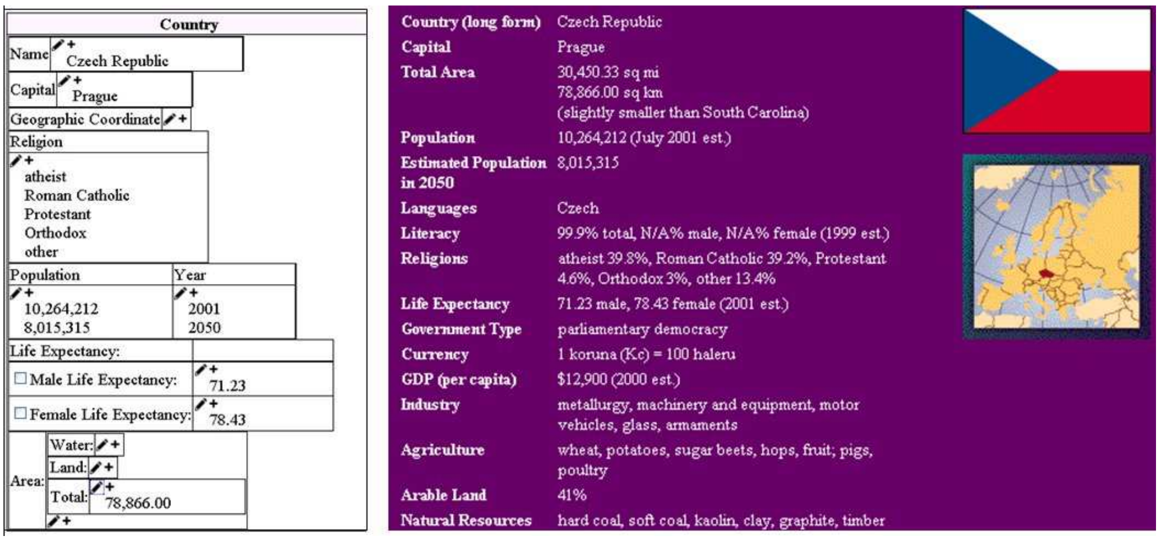
Figure 4.3: A Sample Form

We thus nest each of these properties as single-label/single-value elements within the single-label/single-value element for *Area* as Figure 4.3 shows.

4.4.2 Form Annotation

We annotate a page from a web site with respect to a created form by filling in the form. FOCIH provides users with a GUI in which they can open a page in the web site from which they want to collect information, highlight the value or values of interest for each form field, then fill the values into created forms.

Figure 4.4 shows an example of annotating values using a form. The left hand side shows the filled-in form in the annotation mode. The right hand side shows a sample web page in the domain. For example, to annotate the string “Prague” under “Capital” in the source as a capital, we highlight the string “Prague” by dragging the



Country

Name	✏ + Czech Republic
Capital	✏ + Prague
Geographic Coordinate	✏ +
Religion	✏ + atheist, Roman Catholic, Protestant, Orthodox, other

Population	Year
✏ + 10,264,212	✏ + 2001
8,015,315	2050

Life Expectancy:	
☐ Male Life Expectancy:	✏ + 71.23
☐ Female Life Expectancy:	✏ + 78.43

Area:		
	Water:	✏ +
	Land:	✏ +
	Total:	✏ + 78,866.00
	✏ +	

Country (long form)	Czech Republic
Capital	Prague
Total Area	30,450.33 sq mi 78,866.00 sq km (slightly smaller than South Carolina)
Population	10,264,212 (July 2001 est.)
Estimated Population in 2050	8,015,315
Languages	Czech
Literacy	99.9% total, N/A% male, N/A% female (1999 est.)
Religions	atheist 39.8%, Roman Catholic 39.2%, Protestant 4.6%, Orthodox 3%, other 13.4%
Life Expectancy	71.23 male, 78.43 female (2001 est.)
Government Type	parliamentary democracy
Currency	1 koruna (Kc) = 100 haleru
GDP (per capita)	$12,900 (2000 est.)
Industry	metallurgy, machinery and equipment, motor vehicles, glass, armaments
Agriculture	wheat, potatoes, sugar beets, hops, fruit; pigs, poultry
Arable Land	41%
Natural Resources	hard coal, soft coal, kaolin, clay, graphite, timber

Figure 4.4: An Filled in Form with a Source Data Page

mouse over it and clicking on the ✏ icon in the single-entry *Capital* field—FOCIH automatically adds 'Prague" to the form field under *Capital*.

We can also add more values in one form entry. For example, there are several religions in the Czech Republic. We highlight each of the values "atheist", "Roman Catholic", "Protestant", "Orthodox", and "other" in the source page individually and click on the ✏ icon one by one. Then the these five values appear in the corresponding form field as five data instances.

We can also concatenate two or more highlighted values when filling a form by clicking the click the + icon. For example, suppose there is a web site that provides us with *Geographic Coordinate* information by listing longitude and latitude separately. We first highlight the longitude value and then click on the ✏ icon. To concatenate the latitude values, we highlight it and click on the + icon. Then the longitude and latitude value will appear as one concatenated data instance in the *Geographic Coordinate* form field.

4.5 Ontology Generation

For a created form, FOCIH can generate an ontology inferred from the form. Figure 4.1 shows a generated ontology for the form in Figure 4.3. Based on the from title, FOCIH generates a new ontology and a non-lexical concept with this title as the name. Thus, for the form in Figure 4.3, FOCIH generates the concept Country as Figure 4.1 shows. Every label in the form also represents a concept in the corresponding ontology; the label is the name for the concept. Form concepts with nested components become non-lexical object sets. Thus, *Area* is a non-lexical object set. Form concepts without nested components become lexical object sets. Thus, *Name*, *Capital*, *Geographical Coordinate*, *Religion*, *Population*, *Year*, *Water*, *Land*, and *Total* are all lexical. As a consistency requirement, generalization/specializtion concepts must all be lexical or must all be non-lexical. To meet this requirement, FOCIH declares all the object sets involved in a generalization/specialization to be lexical if there are no nested component other than the nesting of generalization/specialization components themselves; otherwise all concepts are non-lexical. Since there are no non-generalization/specialization form components nested under, *Life Expectancy*, *Male Life Expectancy*, and *Female Life Expectancy* are all lexical.

FOCIH generates relationship sets among the concepts as follows.

- *Single-label/single-value form elements.* Between the form-title concept T and each top-level single-label/single-value form element S, FOCIH generates a functional binary relationship set from T to S. Thus, FOCIH generates functional relationship sets from *Country* to *Name*, from *Country* to *Capital*, from *Country* to *Geographical Coordinate*, and from *Country* to *Area* as Figure 4.1. Similarly, between each form element E and a single-label/single-value form element S nested in side of E, FOCIH also generates a functional binary relationship set from E to S. Thus, FOCIH generates functional relationships form *Area* to *Water*, from *Area* to *Land*, and from *Area* to *Total*.

- *Single-label/multiple-value form elements.* Between each form-title concept T and each single-label/multiple-value concept M, FOCIH generates a non-functional binary relationship set between T and M. Thus FOCIH accommodates the possibly many *Religions* for each *Country* as Figure 4.1 shows. Although our running example has no single-value/multiple-value form elements nested inside other form elements, FOCIH also creates non-functional binary relationship sets between a parent form element and each nested child single-label/multiple-value form element.

- *Multiple-label/multiple-value form elements.* Between the form-title concept and each multiple-label form element as well as between each form element and a multiple-label concept nested inside of it, FOCIH generates either an n-ary relationship set or a set of binary relationship sets. If the multiple-label element is not the only form element in the form, FOCIH generates an n-ary relationship set, otherwise it generates a set of binary relationship sets between the form-title concept and each of the concepts in the multiple-label element. Thus, FOCIH generates an n-ary relationship set among *Country*, *Population*, and *Year*. Our running example does not illustrate the case of a multiple-label form element by itself with no other form elements. As an example consider a multiple-label form element by itself nested inside a form whose title is *Country*. The labels might be *Name*, *Capital*, *Population (2005 est.)*, and *Size (sq. km.)*, and the rows in the multiple-label field would be various country names along with their capitals, populations, and sizes. In this case, FOCIH would generate four functional binary relationship sets: from *Country* to *Name*, from *Country* to *Capital*, from *Country* to *Population (2005 est.)*, and from *Country* to *Size (sq. Km.)*.

- *Choice form elements.*

  FOCIH generates a non-functional binary relationship set between the form-title concept and a top-level choice form element. Thus FOCIH generates a non-functional binary relationship set between *Country* and *Life Expectancy* as Figure 4.1 shows. Similar to other nested form elements, nested choice form elements have the same relationships to their parent form elements as do top-level choice form elements to the form title concept.

  For both mutually-exclusive and non-exclusive choice elements, FOCIH generates a generalization/specialization relationship with the header label as the generalization concept and each of the labels on the selection list as specialization concepts. From the example in Figure 4.4, FOCIH therefore generates a non-exclusive choice element for the generalization/specialization with *Life Expectancy* as the generalization and *Male Life Expectancy* and *Female Life Expectancy* as specializations. Nesting choice form elements inside of choice specification elements extends the generalization/specialization hierarchy. Header labels of of nested generalizations must match upper-level specialization labels. We could, for example, extend the hierarchy by nesting *Male Life Expectancy 40-60* and *Male Life Expectancy 60+* under the upper-level specialization *Male Life Expectancy.* In this case, FOCIH would generate concepts for these specializations which would appear as specialization concepts for the generalization *Male Life Expectancy* in Figure 4.3.

  FOCIH imposes no constraints on generalization/specialization for non-exclusive form elements. For mutually-exclusive form elements, FOCIH adds a plus symbol to the triangle to designate the mutual exclusion. This, however, would be inappropriate for our example because we know that as life-expectancy values are harvested, some male and female life-expectancy values may be the same—thus, the male and female values are not mutually exclusive.

As for determining the lexicality of generalization/specialization hierarchies, the OSM rule requires concepts in the entire hierarchy to be all lexical or all non-lexical. FOCIH generates the hierarchy with all lexical concepts if all leaf form elements in the hierarchy expect a single value as is the case in Figure 4.3. Otherwise, FOCIH generates all concepts in the entire hierarchy as non-lexical.

Although FOCIH is able to generate all concepts, all relationship sets, and all generalization/specialization hierarchies, it can generate only some of the constraints that might be desirable. FOCIH knows that relationship-set constraints from parent content to child concept should be functional when the child concept is a single-label/single-value form element. From a form specification alone, however, FOCIH is not able to determine whether the inverse direction of a binary relationship set is functional. Names of countries, for example, might be unique and therefore functionally determine countries. In these cases, FOCIH initially imposes no constraints. Thus, in Figure 4.1, the *Name-Country* relationship set is not bijective. FOCIH, however, can later modify constraints based on observation as FOCIH harvests information from source documents. The optional (i.e., non-mandatory) constraints on the three relationship set in Figure 4.1 appears initially because FOCIH observes that the first page from which it harvests information (i.e., the page in Figure 4.4 has no *Geographic Coordinate*, no *Water* area, and no *Land* area). FOCIH is reticent, however, to establish constraints where it observes non-violations such as after harvesting from several pages and seeing that capital-city names are unique. Instead, after gathering sufficiently many examples, FOCIH may ask its human users to confirm its educated guess.

4.6 Path and Instance Recognition

To fill in the form, users manually transfer values of interest from the source document to the target entry blanks in the form. Each target form entry blank can contain one or more data instances. To enter an instance, a user highlights the instance with a mouse and then clicks on the ✎ icon in the field in which the value should appear. For example, in Figure 4.4, the form element with label *Religion* is a multiple-value form element, which we fill in with five data instances. Additionally, each instance itself can contain one or more highlighted values. For example, the instance for the *Geographic Coordinate* label could contain two highlighted values one for longitude and one for latitude perhaps from two different data cells in a table from a source page. To indicate that multiple component values should be concatenated together to form a single value, after clicking on the ✎ icon to add the first component, a user clicks on the ➕ icon to concatenate subsequent components. When filling in multiple-label/multiple-value form elements like the *Population-Year* form element, users must be careful to put related values in the same row. For example, value "10,264,212" in the form in Figure 4.4 should go to the same row as "2001".

Although users fill in the form manually, they only need to do this manual transformation once for a single page from a hidden-web site. To harvest information from the remaining pages in the same hidden-web site, FOCIH determines the layout pattern for target instance values in the first page and uses these patterns to extract target instance values from remaining pages. FOCIH accomplishes this task by using path recognition and instance recognition. Neither path recognition nor instance recognition is trivial. Path recognition requires FOCIH to be able to identify the path in the HTML DOM-tree leading to the node that contains each highlighted string. Instance recognition requires FOCIH to be able to identify the substrings in one or more DOM-tree nodes that constitute the instance values.

4.6.1 Path Recognition

Path recognition is about locating the DOM-tree node of a user-highlighted value from a source hidden-web page by using the structural layout pattern of the hidden-web site. Knowing the path, FOCIH can then automatically locate corresponding DOM-tree nodes (sibling nodes) that contain values of interest from the remaining pages in the same hidden-web site. For example, if a user highlights "Czech Republic" in the sample page in Figure 4.4, FOCIH needs to collect all the values under "Country (long form)" from the remaining pages of the same web site.

Since hidden-web pages are usually sibling pages with regular structure, we can usually locate the corresponding DOM-tree node in another sibling page by following the same XPath from the root to the current node. As discussed in Chapter 2, however, XPath does not always locate sibling nodes even though sibling pages usually have the same base structure because some variations might exist. Some sibling pages might have additional or missing tables, and some sibling tables might have additional or missing labels. Thus, sometimes following the XPath does not locate the sibling node for which we are looking.

TISP [52], however, can deal with these variations in sibling pages. Given a table in a sample page, TISP can automatically locate sibling tables of given table in all other sibling pages. In addition, since TISP can interpret sibling tables automatically, FOCIH can obtain the sibling nodes from each of the sibling tables by first locating the corresponding labels and then locating the value nodes for the labels. For example, to obtain all the sibling nodes for *Name* of *Country* in the sibling pages of the page in Figure 4.4, TISP first automatically interprets the table. FOCIH now knows that the first column of this table is a column of labels and that the label "Country (long form)" is the label for the values to be collected. Then for the remaining sibling pages, TISP can locate the corresponding sibling tables. In each table, FOCIH looks for the label ""Country (long form)" and then locates the corresponding value nodes.

4.6.2 Instance Recognition

A user-highlighted value can be the entire DOM-tree node (e.g, "Prague" in Figure 4.4) or a proper subpart of the string that constitutes the DOM-tree node (e.g., just the populated value in Figure 4.4).[2] In the latter case, FOCIH needs to know how to find the right subpart within the DOM-tree node. Moreover, since a value can be composed of one or more highlighted values from one or more DOM-tree nodes (e.g., when longitude and latitude are in separate DOM-tree nodes), FOCIH needs to know how to compose values from different substrings of different nodes from the source page.

Considering these possibilities, we observe that there are two kinds of patterns: (1) individual patterns for entire strings, proper substrings, and string components and (2) list patterns. Particularly, for list patterns, but also as context for individual pattern, FOCIH has a default list of delimiters: ",", ";", "|", " /", "\", "(", ")", "[", "]", "{", "}", *sos* (start of string) and *eos* (end of string). FOCIH also has a library of regular-expression recognizers for values in common formats, such as numbers, numbers with commas, decimal numbers, positive/negative integers, percentages, dates, times, and currencies [19, 23]. An *individual pattern* has left and right context, a regular-expression instance recognizer, and an optional appearance number of the substring. For example, for the highlighted area value "78,866.00", the left context can be "\b" (word boundary) and the right context can be "\ssq", the regular-expression recognizer can be decimal number, and the appearance number is 2 (the second decimal number in the string). A *list pattern* has a left context, a right context, a regular-expression recognizer, and a delimiter. The list of agriculture products in Figure 4.4 would have as its left context *sos*, as its right context *eos*, as

[2]If an identified DOM-tree node is not already a string with no internal formatting tags, FOCIH removes the tags and converts the DOM-tree node to a simple string

its regular-expression recognizer ".*" (any string), and as its delimiter "(,\s*)|(;\s*)" (either a comma space or a semicolon space).

We now explain how FOCIH detects patterns. FOCIH first determines whether a pattern is an individual pattern or a list pattern. Given a DOM-tree node and all the highlighted values in that node, FOCHI groups the highlighted values that go to the same form entry together. If there is only one highlighted value that goes to a form entry, FOCIH recognizes it as an individual pattern; and if there are many highlighted values that go to a form entry, FOCIH recognizes it as a list pattern.

For both individual and list patterns, FOCIH next determines the context information, and the regular expression pattern of the substrings of interest. To determine the left or the right context of a highlighted value in a DOM-tree node, FOCIH initially takes the substring that is on the left or on the right of the highlighted substring until it reaches other highlighted values or the beginning or the end of the whole node string. FOCIH can further generalize the context. First, if some of the context is recognizable as instance of one of the regular-expression recognizers, FOCIH substitutes the recognized substring in the context by the recognizer. Second, FOCIH can generalize the context information when it sees more sibling-node contents during its harvesting phase of operation. Sometimes FOCIH cannot locate the context information in a newly encountered sibling content. This usually means that the initial context is too specific in the original sample page. FOCIH then tries to generalize the the context by comparing context strings with the pattern and allowing non-delimiter characters to be replaced by an expression that permits any characters.

If a highlighted substring can be recognized by a regular-expression recognizer in our library, FOCIH uses it as the regular-expression recognizer for the pattern. If not, then the instance recognizer is an expression that recognizes any string. In this case, proper recognition depends on the left and right context, and for individual values, perhaps also the appearance number, and for lists also the delimiter.

Sometimes, with the same context and regular-expression recognizer, FOCIH locates more than one substring from a DOM-tree node, but an individual pattern should only recognize one substring. In this case, an appearance number can help. An appearance number n tells FOCIH which substring of the several substrings recognized by the context and regular-expression recognizer is the one the user wants. We assume that the sibling pages are highly regular and all the values should appear in the same order over sibling pages. So if the highlighted value appears as the nth recognized value, FOCIH assumes that it should extract the nth value in the remaining sibling pages.

For delimiters in list patterns, FOCIH compares the substrings between highlighted values. Looking particularly for delimiters in our list of delimiters, FOCIH attempts to identify a simple delimiter-separated list. It then constructs a regular expression for the delimiter. The agriculture list in Figure 4.4 is an example. For this list FOCIH creates the delimiter expression “, |;”. For more complex cases such as the religions list in Figure 4.4, the list separator is not merely a simple delimiter. In the religions list a percentage plus a comma separate the names of the religions, and the delimiter expression should be “\s*\d[1-2](.\d*)?%,\s*”. FOCIH generates this delimiter expression by (1) discovering that the percentage recognizer in the library recognizes part of every substring between highlighted values, (2) observing that a comma follows every percentage, and (3) noticing that the combination of the percentage and the comma covers the substrings. In general, FOCIH checks substrings for library instance recognizers and standard delimiters as illustrated in the religions example; when this is insufficient to cover all of the substrings, FOCIH adds general character recognizers, as necessary, to cover the substrings.

4.7 Semantic Annotation

With path recognition and instance recognition, FOCIH can locate the information of interest from all the sibling pages for a hidden-web site and represent it with respect to the generated ontology. Since FOCIH has already "understood" each page, we can immediately and automatically semantically annotate values for each page in the site using the ontology as the annotation ontology. This means that we can transform a source page to a semantic web page, which is machine-understandable [6]. FOCIH annotates each page and saves the annotated information in an RDF file.

Figures 4.5, 4.6, 4.7, 4.8, 4.9, 4.10, and 4.11 show the annotated information for the page in Figure 4.4 with respect to the ontology in Figure 4.1. Lines 2–4 in Figure 4.5 gives the name spaces in the RDF file. The *ann* name space (Line 2) describes the annotation tags, which we explain as we continue our discussion. The *country* name space (Line 4) describes our *Country* ontology (Figure 4.1). Both *ann* and *country* are OWL ontologies—*ann* created by us for the purpose of annotation as discussed here and *country* generated from the OSM description in Figure 4.1.

Lines 5–19 define an instance *Country_1*, for the *Country* class. Line 5 introduces the instance identifier for the annotated web page in Figure 4.4. An *InResource* tag tells us that the source file is *#resource1*. Lines 7–18 tell us the properties of *Country_1* and list the instances as RDF triples. Since *Country-Name*, *Country-Capital*, and *Country-Area* are all functional binary relationship sets, there is only one instance of *Name*, *Capital*, and *Area*, each related to *Country_1*. Since *Country-Religion* is a non-functional binary relationship set, there can be more than one instance of *Religion* for each instance of *Country*. Through the property *Country-Religion*, *Country_1* connects to *#Religion_1* through *#Religion_5*, which are the five instances of *Religion* that appear in Figure 4.4. There is an n-ary relationship set between *Country*, *Population*, and *Year*. Since OWL only supports binary relationship sets, FOCIH generates a new class, *CountryPopulationYear*, to represent this

```
<rdf:RDF
    xmlns:ann="http://dithers.cs.byu.edu/owl/ontologies/annotation#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:country="http://dithers.cs.byu.edu/owl/ontologies/country#">
```

Figure 4.5: Sample RDF Annotation for Name Spaces

```
    <country:Country rdf:ID="Country_1">
        <ann:inResource rdf:resource="#resource1"/>
        <country:Country-Name rdf:resource="#Name_1"/>
        <country:Country-Capital rdf:resource="#Capital_1"/>
        <country:Country-Religion rdf:resource="#Religion_1"/>
        <country:Country-Religion rdf:resource="#Religion_2"/>
        <country:Country-Religion rdf:resource="#Religion_3"/>
        <country:Country-Religion rdf:resource="#Religion_4"/>
        <country:Country-Religion rdf:resource="#Religion_5"/>
        <country:Country-CountryPopulationYear rdf:resource="#CountryPopulationYear_1"/>
        <country:Country-CountryPopulationYear rdf:resource="#CountryPopulationYear_2"/>
        <country:Country-LifeExpectancy rdf:resource="#MaleLifeExpectancy_1"/>
        <country:Country-LifeExpectancy rdf:resource="#FemaleLifeExpectancy_1"/>
        <country:Country-Area rdf:resource="#Area_1"/>
    </country:Country>

```

Figure 4.6: Sample RDF Annotation for *Country*

relationship set. Through the property *Country-CountryPopulationYear*, *Country_1* connects to *#CountryPopulationYear_1* and *#CountryPopulationYear_2*, which are the two instances of the *CountryPopulationYear* relationship set in Figure 4.4. Through the property *Country-LifeExpectancy*, *Country_1* also connects to *#MaleLifeExpectancy_1*, which is an instance of *MaleLifeExpectancy*, and *#FemaleLifeExpectancy_1*, which is an instance of *FemaleLifeExpectancy*. Since *MaleLifeExpectancy* and *FemaleLifeExpectancy* are subclasses of *LifeExpectancy*, they inherit the property *Country-LifeExpectancy* from their parent class.

Figures 4.7, 4.8, and 4.10 show annotation declaration for value instances of *Name*, *Capital*, *Religion*, *MaleLifeExpectancy*, and *FemaleLifeExpectancy*. In each annotation declaration, the *inResource* tag tells which source document the information comes from. The *OffsetOnHTMLPage* tag tells where to locate the specific substring in the source document, and the *HTMLText* tag records the component's text. The *<label>Value* tag tells us the value of the instance, which for string is usually the

```
<country:Name rdf:ID="Name_1">
    <ann:inResource rdf:resource="#resource1"/>
    <ann:OffsetOnHTMLPage>8765</ann:OffsetOnHTMLPage>
    <ann:HTMLText>Czech Republic</ann:HTMLText>
    <country:NameValue>Czech Republic</country:NameValue>
    <country:Name-Country rdf:resource="#Country_1"/>
</country:Name>

<country:Capital rdf:ID="Capital_1">
    <ann:inResource rdf:resource="#resource1"/>
    <ann:OffsetOnHTMLPage>8896</ann:OffsetOnHTMLPage>
    <ann:HTMLText>Prague</ann:HTMLText>
    <country:CapitalValue>Prague</country:CapitalValue>
    <country:Capital-Country rdf:resource="#Country_1"/>
</country:Capital>

```

Figure 4.7: Sample RDF Annotation for Single-Label/Single-Value Elements

```
<country:Religion rdf:ID="Religion_1">
    <ann:inResource rdf:resource="#resource1"/>
    <ann:OffsetOnHTMLPage>9806</ann:OffsetOnHTMLPage>
    <ann:HTMLText>atheist</ann:HTMLText>
    <country:ReligionValue>atheist</country:ReligionValue>
    <country:Religion-Country rdf:resource="#Country_1"/>
</country:Religion>

<country:Religion rdf:ID="Religion_2">
    <ann:inResource rdf:resource="#resource1"/>
    <ann:OffsetOnHTMLPage>9821</ann:OffsetOnHTMLPage>
    <ann:HTMLText>Roman Catholic</ann:HTMLText>
    <country:ReligionValue>Roman Catholic</country:ReligionValue>
    <country:Religion-Country rdf:resource="#Country_1"/>
</country:Religion>

<country:Religion rdf:ID="Religion_3">
    <ann:inResource rdf:resource="#resource1"/>
    <ann:OffsetOnHTMLPage>9843</ann:OffsetOnHTMLPage>
    <ann:HTMLText>Protestant</ann:HTMLText>
    <country:ReligionValue>Protestant</country:ReligionValue>
    <country:Religion-Country rdf:resource="#Country_1"/>
</country:Religion>

<country:Religion rdf:ID="Religion_4">
    <ann:inResource rdf:resource="#resource1"/>
    <ann:OffsetOnHTMLPage>9860</ann:OffsetOnHTMLPage>
    <ann:HTMLText>Orthodox</ann:HTMLText>
    <country:ReligionValue>Orthodox</country:ReligionValue>
    <country:Religion-Country rdf:resource="#Country_1"/>
</country:Religion>

<country:Religion rdf:ID="Religion_5">
    <ann:inResource rdf:resource="#resource1"/>
    <ann:OffsetOnHTMLPage>9873</ann:OffsetOnHTMLPage>
    <ann:HTMLText>other</ann:HTMLText>
    <country:ReligionValue>other</country:ReligionValue>
    <country:Religion-Country rdf:resource="#Country_1"/>
</country:Religion>

```

Figure 4.8: Sample RDF Annotation for Multiple-Label/Mutiple-Value Elements

```
<country:CountryPopulationYear rdf:ID=“CountryPopulationYear_1”>
    <ann:inResource rdf:resource=“#resource1”/>
    <country:CountryPopulationYear-Year rdf:resource=“Year_1”/>
    <country:CountryPopulationYear-Population rdf:resource=“Population_1”/>
    <country:CountryPopulationYear-Country rdf:resource=“#Country_1”/>
</country:CountryPopulationYear>

<country:CountryPopulationYear rdf:ID=“CountryPopulationYear_2”>
    <ann:inResource rdf:resource=“#resource1”/>
    <country:CountryPopulationYear-Year rdf:resource=“Year_2”/>
    <country:CountryPopulationYear-Population rdf:resource=“Population_2”/>
    <country:CountryPopulationYear-Country rdf:resource=“#Country_1”/>
</country:CountryPopulationYear>

<country:Population rdf:ID=“Population_1”>
    <ann:inResource rdf:resource=“#resource1”/>
    <ann:OffsetOnHTMLPage>9224</ann:OffsetOnHTMLPage>
    <ann:HTMLText>10,264,212</ann:HTMLText>
    <country:PopulationValue>10264212</country:PopulationValue>
    <country:Population-CountryPopulationYear rdf:resource=“#CountryPopulationYear_1”/>
</country:Population>

<country:Year rdf:ID=“Year_1”>
    <ann:inResource rdf:resource=“#resource1”/>
    <ann:OffsetOnHTMLPage>9241</ann:OffsetOnHTMLPage>
    <ann:HTMLText>2001</ann:HTMLText>
    <country:YearValue>2001</country:YearValue>
    <country:Year-CountryPopulationYear rdf:resource=“#CountryPopulationYear_1”/>
</country:Year>

<country:Population rdf:ID=“Population_2”>
    <ann:inResource rdf:resource=“#resource1”/>
    <ann:OffsetOnHTMLPage>9389</ann:OffsetOnHTMLPage>
    <ann:HTMLText>8,015,315</ann:HTMLText>
    <country:PopulationValue>8015315</country:PopulationValue>
    <country:Population-CountryPopulationYear rdf:resource=“#CountryPopulationYear_2”/>
</country:Population>

<country:Year rdf:ID=“Year_2”>
    <ann:inResource rdf:resource=“#resource1”/>
    <ann:OffsetOnHTMLPage>9348</ann:OffsetOnHTMLPage>
    <ann:HTMLText>2050</ann:HTMLText>
    <country:YearValue>2050</country:YearValue>
    <country:Year-CountryPopulationYear rdf:resource=“#CountryPopulationYear_2”/>
</country:Year>

```

Figure 4.9: Sample RDF Annotation for *n*-ary Relationship Sets

```
    <country:MaleLifeExpectancy rdf:ID=“MaleLifeExpectancy_1”>
        <ann:inResource rdf:resource=“#resource1”/>
        <ann:OffsetOnHTMLPage>10009</ann:OffsetOnHTMLPage>
        <ann:HTMLText>71.23</ann:HTMLText>
        <country:MaleLifeExpectancyValue>71.23</country:MaleLifeExpectancyValue>
        <country:LifeExpectancy-Country rdf:resource=“#Country_1”/>
    </country:MaleLifeExpectancy>

    <country:FemaleLifeExpectancy rdf:ID=“FemaleLifeExpectancy_1”>
        <ann:inResource rdf:resource=“#resource1”/>
        <ann:OffsetOnHTMLPage>10021</ann:OffsetOnHTMLPage>
        <ann:HTMLText>78.43</ann:HTMLText>
        <country:FemaleLifeExpectancyValue>78.43</country:FemaleLifeExpectancyValue>
        <country:LifeExpectancy-Country rdf:resource=“#Country_1”/>
    </country:FemaleLifeExpectancy>

```

Figure 4.10: Sample RDF Annotation for Generalization/Specialization

```
    <country:Area rdf:ID=“Area_1”>
        <ann:inResource rdf:resource=“#resource1”/>
        <country:Area-Total rdf:resource=“Total_1”/>
        <country:Area-Country rdf:resource=“#Country_1”/>
    </country:Area>

    <country:Total rdf:ID=“Total_1”>
        <ann:inResource rdf:resource=“#resource1”/>
        <ann:OffsetOnHTMLPage>9044</ann:OffsetOnHTMLPage>
        <ann:HTMLText>78,866.00</ann:HTMLText>
        <country:TotalValue>78866.00</country:TotalValue>
        <country:Total-Area rdf:resource=“#Area_1”/>
    </country:Total>

</rdf:RDF>
```

Figure 4.11: Sample RDF Annotation for Nested Form Elements

same as the *HTMLText* value, but for RDF types such as *integer*, and *date-time*, the value is reformatted for the type.

Figure 4.9 shows declarations for instances *#CountryPopulationYear_1* and *#CountryPopulationYear_2*. Each of these two instances also connects to an instance of *Year*, an instance of *Population*, and an instance of *Country* as Lines 79–81 and 86–88 show. Lines 91–121 show the annotation declarations of these instances. Observe that the *PopulationValue* instances (Lines 95 and 111) are reformatted as integers. FOCIH makes this adjustment base on its having recognized these values with library recognizers, which have built in converters for RDF types.

Lines 139-143 in Figure 4.11 show the declarations of *#Area_1*. *#Area_1* connects to *#Total_1* through the property *Area-Total*. Lines 145-151 show the annotation declaration for *Total_1*.

Annotating a value whose component parts appear in different places in a source document requires a more complex annotation specification. As an example, Figure 4.12 shows how FOCIH keeps track of values concatenated from component parts in an annotated RDF file. All concatenated values are stored similarly. Suppose in an original file, the information about *GeographicCoordinate* appears as "49 45 N", and "15 30 E" and comes from different places in the web page or even from different web pages.[3] In the ontology view, we want to show them as one single value "49 45 N 15 30 E". As Figure 4.12 shows, FOCIH stores the concatenated value as *GeographicalCoordinate_1* and then generates two objects *GeographicCoordinateComponent_1* and *GeographicCoordinateComponent_2*, one for each component part of the value and makes them as *hasComponent*-properties of *GeographicalCoordinate_1* declaration.

After storing the information in an RDF file, we query it. Figure 4.13 shows a sample SPARQL query in the Twinkle interface [1]. Our sample query finds the

[3]Although unlikely for FOCIH applications, component parts of values may appear in different web pages.

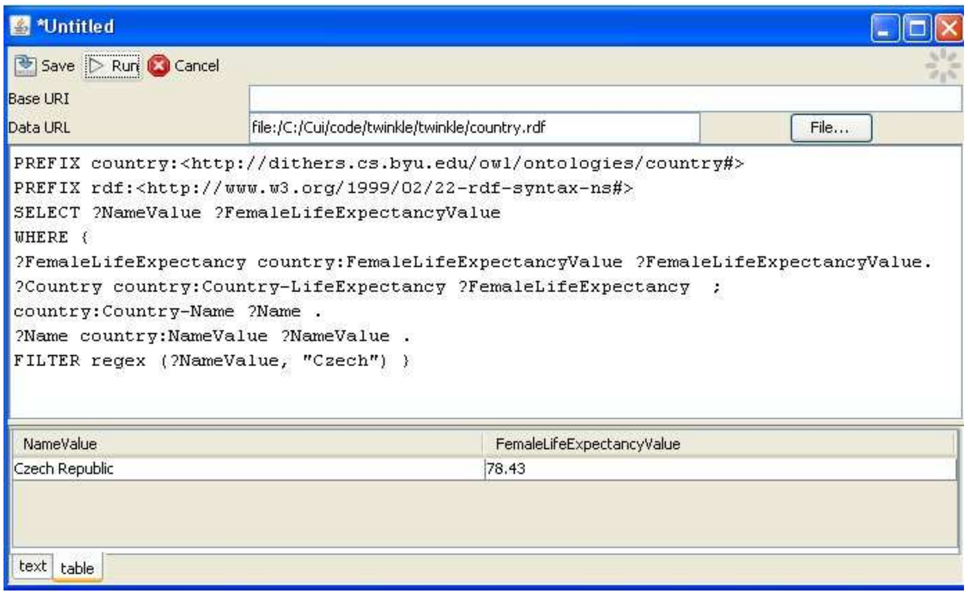
```
<country:GeographicCoordinate rdf:ID="GeographicCoordinate_1"> ...
    <country:GeographicCoordinate-Country rdf:resource="#Country_1"/ >
    <country:GeographicCoordinateValue>49 45 N 15 30 E</country:GeographicCoordinateValue>
    <ann:hasComponent rdf:resource="#GeographicCoordinateComponent_1"/ >
    <ann:hasComponent rdf:resource="#GeographicCoordinateComponent_2"/ >
</country:GeographicCoordinate>
...
<ann:AnnotatedHTMLText rdf:ID="GeographicCoordinateComponent_1"> ...
    <ann:inResource rdf:resource="#resource3"/ >
    <ann:OffsetOnHTMLPage>487</ann:OffsetOnHTMLPage>
    <ann:HTMLText>49 45 N</ann:HTMLText>
</ann:AnnotatedHTMLText>
...
<ann:AnnotatedHTMLText rdf:ID="GeographicCoordinateComponent_2"> ...
    <ann:inResource rdf:resource="#resource3"/ >
    <ann:OffsetOnHTMLPage>530</ann:OffsetOnHTMLPage>
    <ann:HTMLText>15 30 E</ann:HTMLText>
</ann:AnnotatedHTMLText>
```

Figure 4.12: Sample RDF Annotation for Instance Concatenation

Czech female life expectancy. Using the filter statement, SPARQL can find all the instances of *NameValue* that contain string "Czech". Then through the property *NameValue*, SPARQL can locate all the *Name* instances we are looking for, in our example, *#Name_1*. Further though the property *Country-Name*, SPARQL locates *#Country_1*. Finally through the property *Country-LifeExpectancy*, SPARQL can find the instances *#FemaleLifeExpectancy_1* and *#MaleLifeExpectancy_1*. Then follow the property *FemaleLifeExpectancyValue*, SPARQL can find the value we are looking for.

4.8 Initializing Forms for FOCIH with TISP

The purpose of FOCIH is to help users who do not know conceptual modeling to create ontologies of their own, and then to harvest information with respect to these created ontologies. We want to make this process as convenient as we can. One way we can help is to make the form creation process automatic or semi-automatic rather than manual. Sometimes users of one hidden-web site S_1 would like to view the data of another hidden-web site with respect to the organization of data in S_1. If we could reverse engineer the organization of S_1 into a form, we could then allow users to fill in



file:/C:/Cui/code/twinkle/twinkle/country.rdf

```
PREFIX country:<http://dithers.cs.byu.edu/owl/ontologies/country#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?NameValue ?FemaleLifeExpectancyValue
WHERE {
?FemaleLifeExpectancy country:FemaleLifeExpectancyValue ?FemaleLifeExpectancyValue.
?Country country:Country-LifeExpectancy ?FemaleLifeExpectancy  ;
country:Country-Name ?Name .
?Name country:NameValue ?NameValue .
FILTER regex (?NameValue, "Czech") }
```

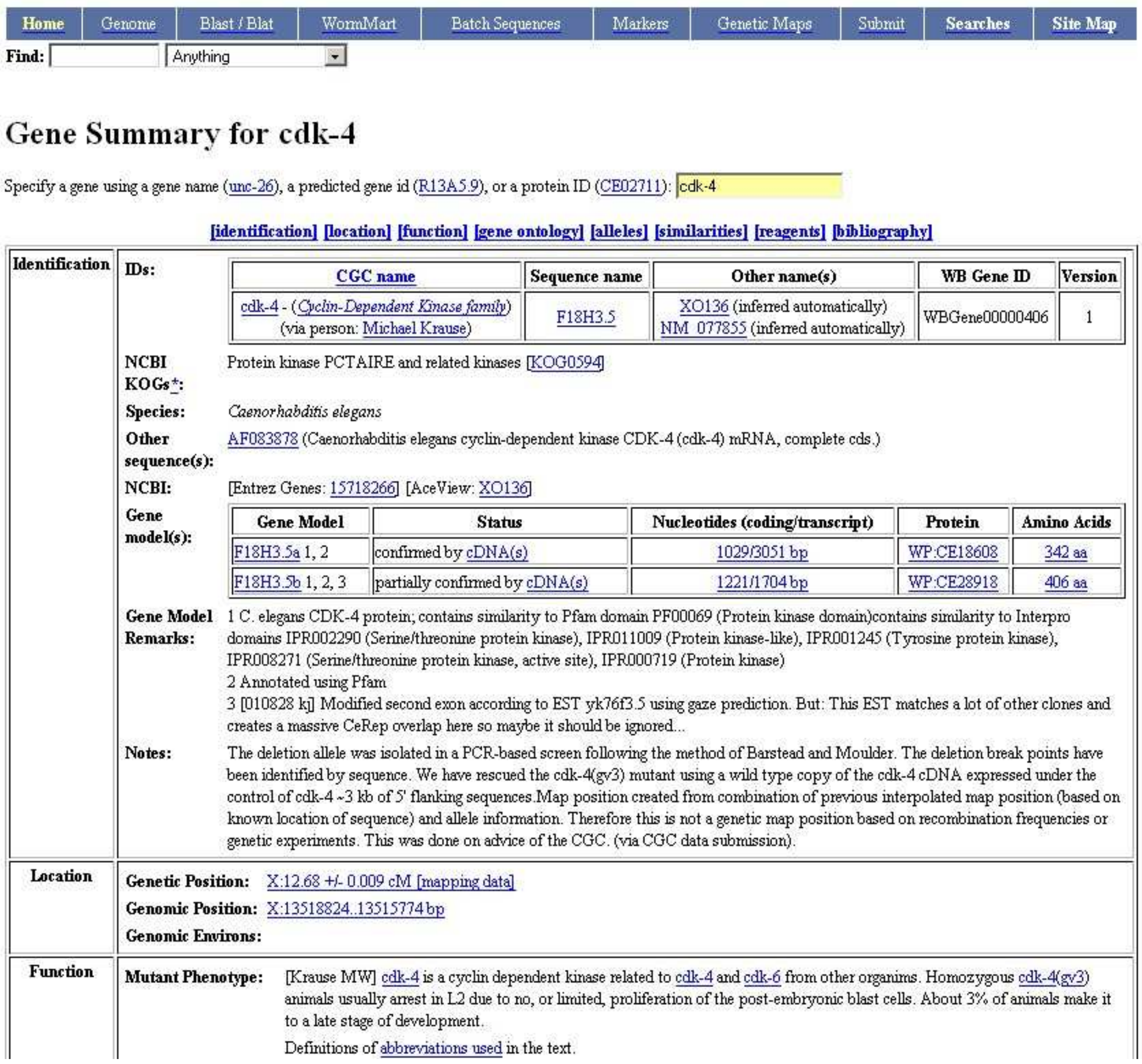
NameValue	FemaleLifeExpectancyValue
Czech Republic	78.43

Figure 4.13: A Sample SPARQL Query using Twinkle [1].

the form with values from a page of S_2. We could then harvest the information from all the pages of S_2 with respect to the view of S_1. As we explain in this section TISP used in conjunction with FOCIH allows us to provide users with this possibility.

Often tables are "mirror images" of forms. When they are and when they only use FOCIH equivalent layout structures, we can immediately generate FOCIH forms for them. To generate forms from tables, FOCIH first applies TISP to interpret sibling tables and determine their layout structures. When their layout structure satisfied the constraints of FOCIH forms, FOCIH recasts them as forms. To obtain a table for the form, FOCIH uses the name of the web site as the form title.

As a fairly complex example, Figure 4.14 shows a page with a nested table and Figure 4.15 shows the generated form. The overall table is the one with labels *Identification*, *Location*, and *Function*. Since there is only one column of values associated with the labels, FOCIH generates three single-label/single-value form elements, one for each label as Figure 4.15 shows. Inside of the value cell for label *Identification*, there is another table with eight labels starting with label *IDs:*. Thus, nested inside the entry field of the form element with label *Identification*, FOCHI generates eight



Gene Summary for cdk-4

Specify a gene using a gene name (unc-26), a predicted gene id (R13A5.9), or a protein ID (CE02711): cdk-4

[identification] [location] [function] [gene ontology] [alleles] [similarities] [reagents] [bibliography]

Identification

IDs:

CGC name	Sequence name	Other name(s)	WB Gene ID	Version
cdk-4 - (*Cyclin-Dependent Kinase family*) (via person: Michael Krause)	F18H3.5	XO136 (inferred automatically) NM_077855 (inferred automatically)	WBGene00000406	1

NCBI KOGs*: Protein kinase PCTAIRE and related kinases [KOG0594]

Species: *Caenorhabditis elegans*

Other sequence(s): AF083878 (Caenorhabditis elegans cyclin-dependent kinase CDK-4 (cdk-4) mRNA, complete cds.)

NCBI: [Entrez Genes: 15718266] [AceView: XO136]

Gene model(s):

Gene Model	Status	Nucleotides (coding/transcript)	Protein	Amino Acids
F18H3.5a 1, 2	confirmed by cDNA(s)	1029/3051 bp	WP:CE18608	342 aa
F18H3.5b 1, 2, 3	partially confirmed by cDNA(s)	1221/1704 bp	WP:CE28918	406 aa

Gene Model Remarks: 1 C. elegans CDK-4 protein; contains similarity to Pfam domain PF00069 (Protein kinase domain)contains similarity to Interpro domains IPR002290 (Serine/threonine protein kinase), IPR011009 (Protein kinase-like), IPR001245 (Tyrosine protein kinase), IPR008271 (Serine/threonine protein kinase, active site), IPR000719 (Protein kinase)
2 Annotated using Pfam
3 [010828 kj] Modified second exon according to EST yk76f3.5 using gaze prediction. But: This EST matches a lot of other clones and creates a massive CeRep overlap here so maybe it should be ignored...

Notes: The deletion allele was isolated in a PCR-based screen following the method of Barstead and Moulder. The deletion break points have been identified by sequence. We have rescued the cdk-4(gv3) mutant using a wild type copy of the cdk-4 cDNA expressed under the control of cdk-4 ~3 kb of 5' flanking sequences.Map position created from combination of previous interpolated map position (based on known location of sequence) and allele information. Therefore this is not a genetic map position based on recombination frequencies or genetic experiments. This was done on advice of the CGC. (via CGC data submission).

Location

Genetic Position: X:12.68 +/- 0.009 cM [mapping data]
Genomic Position: X:13518824..13515774 bp
Genomic Environs:

Function

Mutant Phenotype: [Krause MW] cdk-4 is a cyclin dependent kinase related to cdk-4 and cdk-6 from other organims. Homozygous cdk-4(gv3) animals usually arrest in L2 due to no, or limited, proliferation of the post-embryonic blast cells. About 3% of animals make it to a late stage of development.

Definitions of abbreviations used in the text.

Figure 4.14: A Sample Table from WormBase [62].

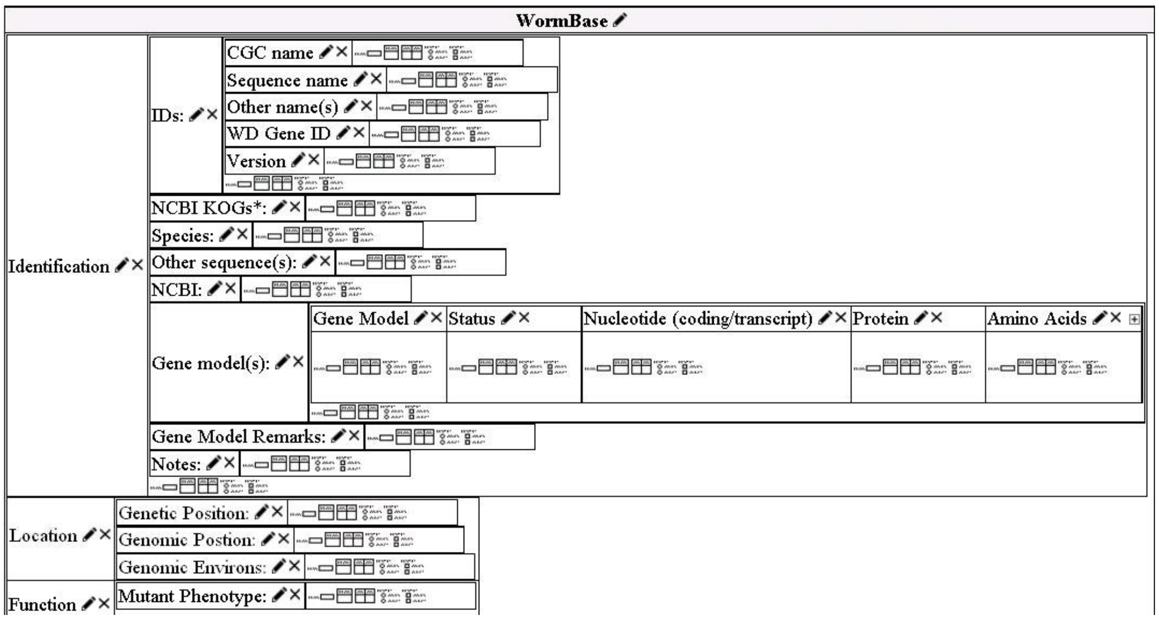
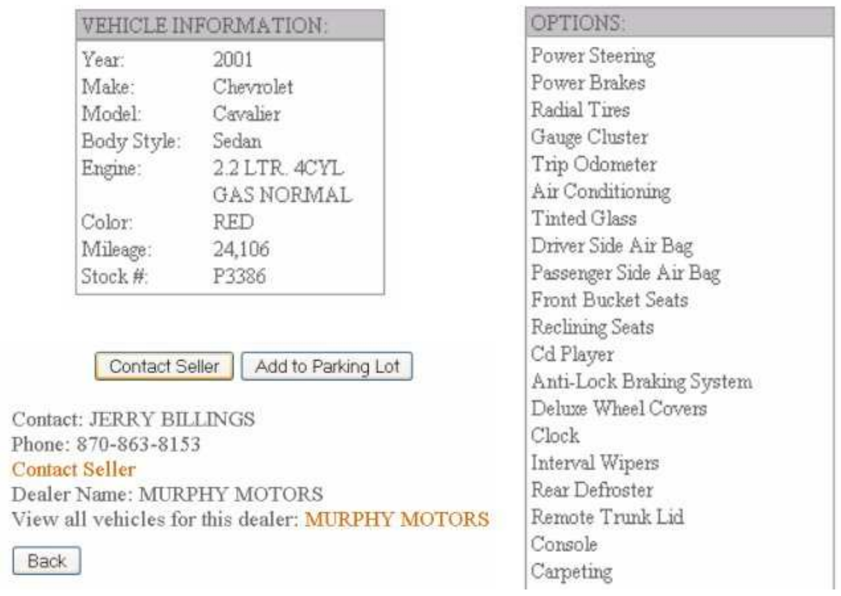


Figure 4.15: Generated Form for Table in Figure 4.14 (Partial)

single-label/single-value form elements, one for each label. Further, inside of the value cell for label *IDs:*, there are a table with five labels starting with the label *CGC name* and each label has only one value associated with it. Thus, inside the entry field of the form element with label *IDs:*, FOCHI generates five single-label/single-value form elements, one for each label. Inside of the value cell for label *Gene model(s):*, there is a table with five labels starting with the label *Gene Model* and this table has two rows of values. For this case, FOCHI then generates a multiple-label/multiple-value form element. There are more tables in the figure, we only list these three to illustrate how FOCIH forms correspond to tables.

After having a form generated for FOCIH via TISP, users may wish to alter the form before harvesting additional information. To illustrate this idea and to show how a user can create a form from one site and harvest from another, we give an example.

Suppose we are interested in buying a car, and we are trying to find information about used cars from different online dealer web sites or car inventory web sites. We care about the make, the model, the engine information, the color, and the mileage


VEHICLE INFORMATION:	
Year:	2001
Make:	Chevrolet
Model:	Cavalier
Body Style:	Sedan
Engine:	2.2 LTR. 4CYL GAS NORMAL
Color:	RED
Mileage:	24,106
Stock #:	P3386

Contact: JERRY BILLINGS
Phone: 870-863-8153
Contact Seller
Dealer Name: MURPHY MOTORS
View all vehicles for this dealer: MURPHY MOTORS

OPTIONS:
Power Steering
Power Brakes
Radial Tires
Gauge Cluster
Trip Odometer
Air Conditioning
Tinted Glass
Driver Side Air Bag
Passenger Side Air Bag
Front Bucket Seats
Reclining Seats
Cd Player
Anti-Lock Braking System
Deluxe Wheel Covers
Clock
Interval Wipers
Rear Defroster
Remote Trunk Lid
Console
Carpeting

Figure 4.16: A Sample Table for Form Generation

on the car. We might search around and find the web page in Figure 4.16. We might also wish all the web sites would present their information like this web site does.

To make this happen, we apply TISP/FOCIH to generate a form automatically for us so we can harvest information from all the car web sites with respect to this view. The result is in Figure 4.17.

Looking at the generated form in Figure 4.17 further, we see that we do not really need to the *Stock#* and that the *OPTIONS* are too detailed for us. So we decide to remove them from the form. Further, we care about a specific breakdown of the engine information—the number of cylinders, the size of the engine, and the fuel injection type. Thus we nest three single-label/single-value form elements inside *Engine* with labels *Cylinder*, *Engine Size*, and *Fuel Injection* as Figure 4.18 shows.

Next we select a sample page for each of the additional web sites from which we want to harvest information and fill in the modified form for each sample page. Figures 4.19, 4.20, and 4.21 show examples. FOCIH harvests the requested information from all the sibling pages in these web sites and semantically annotates each page.

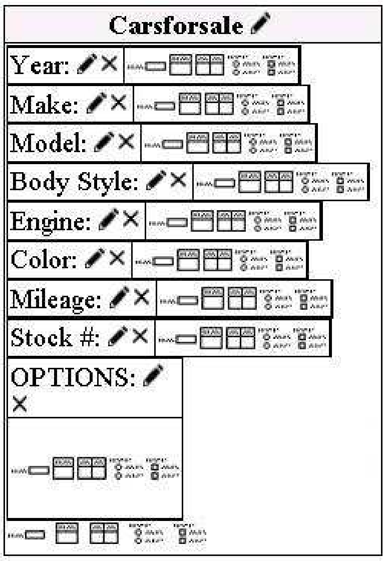


Figure 4.17: Generated Form for Table in Figure 4.16

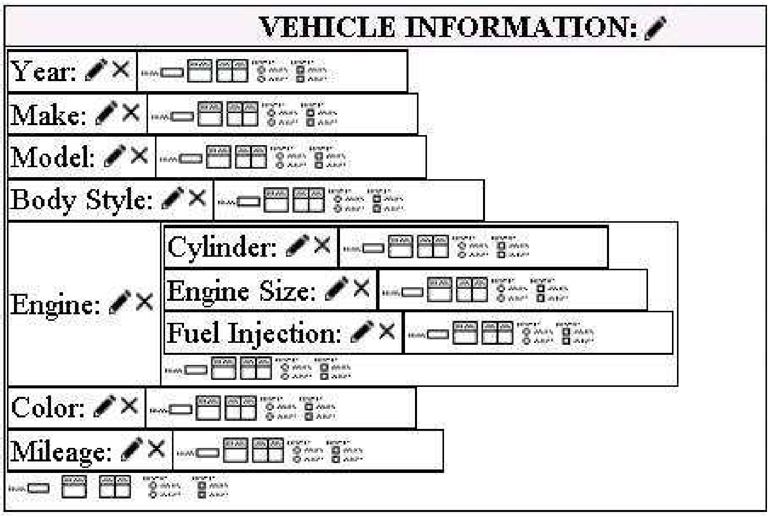
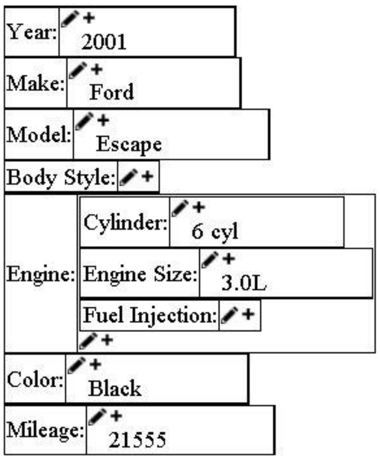
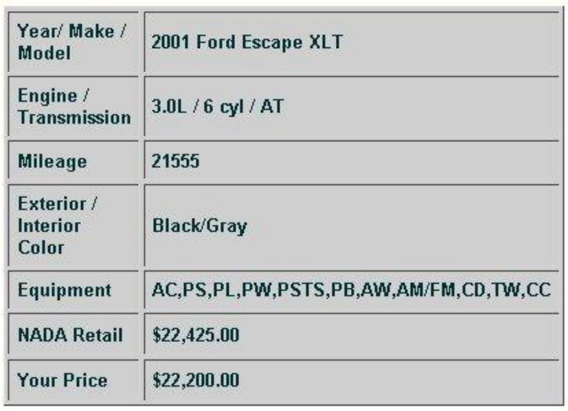


Figure 4.18: The User Modified Form Request According the Form in Figure 4.16



Year:	2001
Make:	Ford
Model:	Escape
Body Style:	
Engine: Cylinder:	6 cyl
Engine: Engine Size:	3.0L
Engine: Fuel Injection:	
Color:	Black
Mileage:	21555

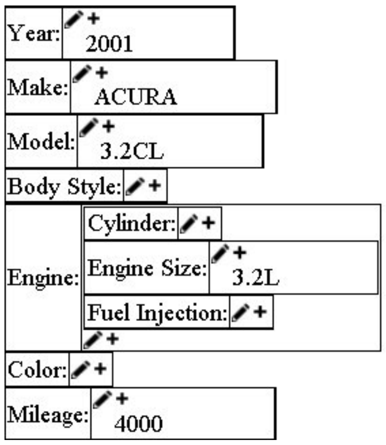
(a)



Year/ Make / Model	2001 Ford Escape XLT
Engine / Transmission	3.0L / 6 cyl / AT
Mileage	21555
Exterior / Interior Color	Black/Gray
Equipment	AC,PS,PL,PW,PSTS,PB,AW,AM/FM,CD,TW,CC
NADA Retail	$22,425.00
Your Price	$22,200.00

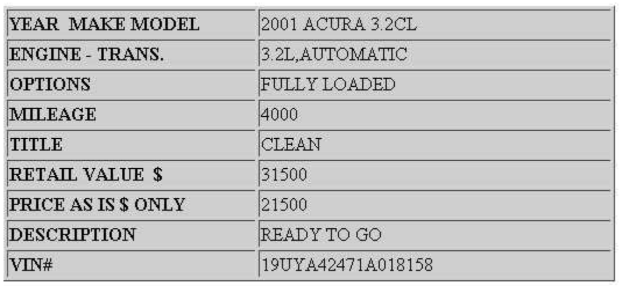
(b)

Figure 4.19: Form Filling Results for Second Sample Car Page



Year:	2001
Make:	ACURA
Model:	3.2CL
Body Style:	
Engine: Cylinder:	
Engine: Engine Size:	3.2L
Engine: Fuel Injection:	
Color:	
Mileage:	4000

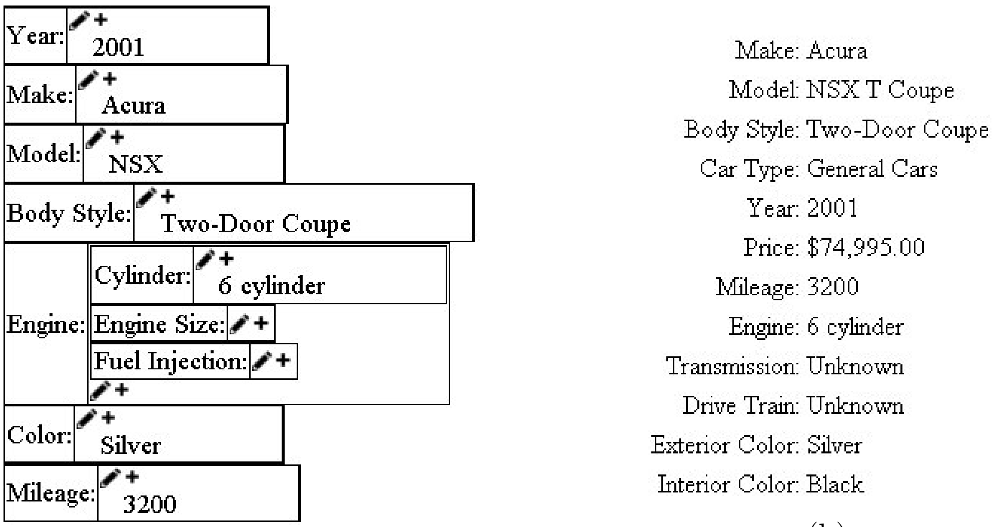
(a)



YEAR MAKE MODEL	2001 ACURA 3.2CL
ENGINE - TRANS.	3.2L,AUTOMATIC
OPTIONS	FULLY LOADED
MILEAGE	4000
TITLE	CLEAN
RETAIL VALUE $	31500
PRICE AS IS $ ONLY	21500
DESCRIPTION	READY TO GO
VIN#	19UYA42471A018158

(b)

Figure 4.20: Form Filling Results for Third Sample Car Page



Year:	2001	
Make:	Acura	
Model:	NSX	
Body Style:	Two-Door Coupe	
Engine:	Cylinder:	6 cylinder
	Engine Size:	
	Fuel Injection:	
Color:	Silver	
Mileage:	3200	

(a)

Make:	Acura
Model:	NSX T Coupe
Body Style:	Two-Door Coupe
Car Type:	General Cars
Year:	2001
Price:	$74,995.00
Mileage:	3200
Engine:	6 cylinder
Transmission:	Unknown
Drive Train:	Unknown
Exterior Color:	Silver
Interior Color:	Black

(b)

Figure 4.21: Form Filling Results for Fourth Sample Car Page

4.9 Conclusion

FOCIH provides users who do not know conceptual modeling or ontology languages a way to create ontologies that represent their own particular views for various domains. FOCIH lets users design a form that captures their view of a domain; then, from a designed form, FOCIH generates an ontology. FOCIH also allows users to say that the display tables of a hidden-web site capture their view of a domain; then via TISP, FOCIH generates a corresponding form which users may alter to more precisely capture their view of a domain. In either case, the user has a form representing an ontological description of a domain. Given a form, users can show FOCIH how to fill in the form for a typical page in a hidden web site. From a filled in form from a typical page, FOCIH infers path and instance-recognition expressions so that it is able to locate information to fill in the form from the pages in the rest of the hidden web site. FOCIH can thus harvest information from the hidden-web site with respect to the generated ontologies. While harvesting, FOCIH semantically annotates the information of interest and generates an RDF file. By doing so, the data of interest present in the sibling pages in the hidden-web site become accessible through a standard query interface, and users can query the information based on their own specified view.

Several directions remain to be pursued. (1) We can augment form specifications so that they can generate even richer ontologies. In particular, we want to add an aggregation constraint so that users can capture subpart/superpart hierarchies. (2) Besides generating forms from sibling pages, there are other ways to make form creation convenient. We also want to provide users with the option to convert an existing ontology (e.g, an OWL ontology) to a form. There are at least two benefits of doing so. First, if a user wants to annotate information with respect to an OWL ontology, this option provides users with a convenient way through forms to annotate all the information of interest in a hidden-web site with respect to the selected ontology. Second, an existing ontology can provide users with a starting point to describe a domain. Users can make modifications as desired. (3) We would also like to facilitate the form-filling process. After a user manually fills in a created form based on one page, FOCIH can harvest information automatically from all other pages in the same hidden-web site. But for a new web site, the user has to fill in the form again for another sample page. An extraction ontology [23] can help with the form filling. After FOCIH harvests information from one web site, we have many sample values for each concept in the ontology. These sample values could provide FOCIH with enough information to generate extraction ontologies. These extraction ontologies could then be used to automatically locate information of interest from a sample web page without users having to manually locate them for FOCIH.

Chapter 5

Conclusions

5.1 Summary of Contributions

In this dissertation, we introduce TISP, TISP++, and FOCIH. TISP and TISP++ contribute to automatic table interpretation, ontology generation, and semantic annotation. TISP provides a solution to automatic interpretation for sibling tables as typically found on the hidden web. TISP++ offers a way to automate ontology generation given an interpreted table and enables automatic semantic annotation for interpreted tables. FOCIH contributes to personalized ontology creation and information harvesting. It gives users a way, without knowing ontology languages, to create an ontology, and it can also harvest information with respect to a user-created ontological view.

Seen as part of a larger whole, TISP++, FOCIH, and TISP/FOCIH contribute ways to help create a web of knowledge and superimpose it over the current web of pages. With a web of knowledge superimposed over web pages, users can ask free-form questions, receive answers to their questions, and check generated answers by viewing original pages from which the answers were extracted.

- TISP++ can transform data stored in sibling tables in the hidden web to a web of knowledge fully automatically without any user interaction. Based on the structure of sibling tables, TISP++ can generate ontologies automatically; and based on the association between table labels and values, TISP++ can automatically annotate these values. By doing so, TISP++ makes the information present in hidden-web tables publicly accessible by common queries. All the queryable information as well as the annotation information is linked together in generated RDF files. This way, TISP++ enables a web of knowledge for a certain segment of hidden web sites—namely those that present Their data as HTML tables, one for each site object.

- One problem with TISP++, however, is that the generated ontology and the annotated data are only represented in the way the original table represents it. FOCIH provides a way for users to declare their own ontologies and annotate web pages for the web of knowledge according to their declared ontology. FOCIH provides users with an interface where they can generate different form components to represent their view of a domain. If a user likes a particular way a table presents data, converted-from-table forms, FOCIH generates an ontology. Then based on a user's filled-in information, FOCIH can harvest and annotate information for the web of knowledge with respect to that particular view.

5.2 Future Work

With these contributions, this dissertation serves as a foundational pillar for turning the current web of pages into a web of knowledge. Having come this far, it is clear that several future-work items are immediately on the horizon. (1) Currently, TISP only works with information stored in sibling tables. We would like to extend our work to automatically harvest and semantically annotate information stored in not only in tables, but also in sibling pages in general. (2) We want to learn more about reverse-engineering ontologies to forms. Given ontologies in different ontology languages such as OWL, FOCIH should be able to automatically generate forms for users to fill in. This would provide a way for FOCIH to harvest and annotate information with respect to an existing ontology. (3) We see the possibility of being able to convert the generated ontologies into extraction ontologies. Based on the information that FOCIH harvests for each concept in an ontology, we obtain valuable information for instance recognizers. By adding these instance recognizers to each concept in a generated ontology, the ontology becomes an extraction ontology. How well it operates depends on how good the instance recognizers are. As additional information is harvested, it should be possible to automatically enhance these instance recognizers.

Bibliography

[1] Twinkle: A SPARQL Query Tool. http://www.ldodds.com/projects/twinkle/, 2008.

[2] W3C Web Ontology Language. http://www.w3.org/TR/owl-features/, 2008.

[3] M.J. Al-Muhammed. *Ontology Aware Software Service Agents: Meeting Ordinary User Needs on the Semantic Web.* PhD thesis, Brigham Young University, 2007.

[4] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD'03)*, pages 337–348, San Diego, California, 2003.

[5] S.M. Benslimane, M. Malki, M.K. Rahmouni, and D. Benslimane. Extracting personalised ontology from data-intensive web application: an HTML forms-based reverse engineering approach. *Informatica*, 18(4):511–534, 2007.

[6] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, pages 28–37, May 2001.

[7] P. Buitelaar, D. Olejnik, and M. Sintek. Ontolt: A Protégé plug-in for ontology extraction from text based on linguistic analysis. In *Proceedings of the First European Semantic Web Symposium (ESWS'04)*, pages 31–44, Heraklion, Greece, May 2004.

[8] A. Bundy and F. McNeill. Representation as a fluent: An AI challenge for the next half century. *IEEE Intelligent Systems*, 21(3):85–87, 2006.

[9] H. Chen, S. Tsai, and J. Tsai. Mining tables from large scale HTML texts. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING'00)*, pages 166–172, Saarbrücken, Germany, July–August 2000.

[10] P. Cimiano, S. Handschuh, and S. Staab. Towards the self-annotating web. In *Proceedings of the 13th International Conference on World Wide Web (WWW'04)*, pages 462–471, New York, New York, May 2004.

[11] P. Cimiano and J. Völker. Text2Onto—a framework for ontology learning and data-driven change discovery. In *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB'05)*, pages 227–238, Alicante, Spain, June 2005.

[12] W.W. Cohen, M. Hurst, and L.S. Jensen. A flexible learning system for wrapping tables and lists in HTML documents. In *Proceedings of the 11th International World Wide Web Conference (WWW'02)*, pages 232–241, Honolulu, Hawaii, May 2002.

[13] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards automatic data extraction from large web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 109–118, Rome, Italy, September 2001.

[14] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, K.S. Mccurley, S. Rajagopalan, and A. Tomkins. A case for automated large-scale semantic annotation. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):115–132, December 2003.

[15] Y. Ding, D.W. Embley, and S.W. Liddle. Automatic creation and simplified querying of semantic web content: An approach based on information-extraction ontologies. In *Proceedings of the First Asian Semantic Web Conference (ASWC'06)*, pages 400–414, Beijing, China, September 2006.

[16] Y. Ding, D.W. Embley, and S.W. Liddle. Enriching OWL with instance recognition semantics for automated semantic annotation. In *Proceedings of the First International Workshop on Ontologies and Information Systems for the Semantic Web (ONISW'07)*, pages 160–169, Auckland, New Zealand, November 2007.

[17] A. Dingli, F. Ciravegna, and Y. Wilks. Automatic semantic annotation using unsupervised information extraction and integration. In *Proceedings of the Third International Conference on Knowledge Capture (K-CAP'03), Workshop on Knowledge Markup and Semantic Annotation*, pages 514–519, Sanibel Island, Florida, October 2003.

[18] The W3C architecture domain. http://www.w3.org/dom/, 2005.

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

[20] D.W. Embley. NFQL: the natural forms query language. *ACM Transactions on Database Systems*, 14(2):168–211, 1989.

[21] D.W. Embley, D.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 pages. *Data & Knowledge Engineering*, 31(3):227–251, November 1999.

[22] D.W. Embley, M. Hurst, D. Lopresti, and G. Nagy. Table processing paradigms: A research survey. *International Journal of Document Analysis and Recognition*, 8(2-3):66–86, June 2006.

[23] D.W. Embley, Y.S. Jiang, and Y.-K. Ng. Record-boundary discovery in Web documents. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD'99)*, pages 467–478, Philadelphia, Pennsylvania, May–June 1999.

[24] D.W. Embley, C. Tao, and S.W. Liddle. Automatically extracting ontologically specified data from HTML tables with unknown structure. In *Proceedings of the 21st International Conference on Conceptual Modeling (ER'02)*, pages 322–327, Tampere, Finland, October 2002.

[25] D.W. Embley, C. Tao, and S.W. Liddle. Automating the extraction of data from HTML tables with unknown structure. *Data & Knowledge Engineering*, 54(1):3–28, July 2005.

[26] A. Gal, G. Modica, and H. Jamil. Ontobuilder: fully automatic extraction and consolidation of ontologies from web sources. In *Proceedings of the 20th International Conference on Data Engineering (ICDE'04)*, page 853, Boston, Massachusetts, March–April 2004.

[27] D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematics Monthly*, 69(1):9–14, 1962.

[28] W. Gatterbauer and P. Bohunsky. Table extraction using spatial reasoning on the CSS2 visual box model. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, pages 1313–1318, Boston, Massachusetts, July 2006.

[29] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, and B. Pollak. Towards domain-independent information extraction from web tables. In *Proceedings of*

the 16th International World Wide Web Conference (WWW'07), pages 71–80, Banff, Canada, May 2007.

[30] S. Handschuh, S. Staab, and F. Ciravegna. S-CREAM — Semi-automatic CREAtion of Metadata. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW'02)*, pages 358–372, Siguenza, Spain, October 2002.

[31] W. Holzinger, B. Krüpl, and M. Herzoge. Using ontologies for extracting product features from web pages. In *Proceedings of the Fifth International Semantic Web Conference (ISWC'06)*, pages 286–299, Athens, Georgia, November 2006.

[32] P.G. Ipeirotis, L. Gravano, and M. Sahami. Probe, count, and classify: categorizing hidden web databases. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD'01)*, pages 67–78, Santa Barbara, California, May 2001.

[33] Jena — A Semantic Web Framework for Java. http://jena.sourceforge.net/, 2008.

[34] P. Jha and G. Nagy. Wang notation tool: Layout independent representation of tables. In *Proceedings of the 19th International Conference on Pattern Recognition (ICPR08)*, Tampa, Florida, December 2008. (in press).

[35] E. Kapetanios. Quo vadis computer science: From turing to personal computer, personal content and collective intelligence. *Data & Knowledge Engineering*, 67(2):286–292, 2008.

[36] P. Kogut and W. Holmes. AeroDAML: Applying information extraction to generate DAML annotations from web pages. In *Proceedings of the of the First International Conference on Knowledge Capture (K-CAP'01) Workshop on Knowledge Markup and Semantic Annotation*, pages 190–197, Victoria, British Columbia, 2001.

[37] K. Lerman, L. Getoor, S. Minton, and C. Knoblock. Using the structure of web sites for automatic segmentation of tables. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD'04)*, pages 119–130, Paris, France, June 2004.

[38] S. Lim and Y. Ng. An automated approach for retrieving heirarchical data from HTML tables. In *Proceedings of the Eighth International Conference on*

Informaiton and Knowledge management (CIKM'99), pages 466–474, Kansas City, Missouri, November 1999.

[39] S. Lynn and D.W. Embley. Semantically conceptualizing and annotating tables. In *Proceedings of the Third Asian Semantic Web Conference*, Bangkok, Thailand, December 2008. (in press).

[40] R. Navigli, P. Velardi, A. Cucchiarelli, and F. Neri. Quantitative and qualitative evaluation of the OntoLearn ontology learning system. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 1043–1050, Geneva, Switzerland, August 2004.

[41] N.F. Noy, M. Sintek, S. Decker, M. Crubezy, R.W. Fergerson, and M. Musen. Creating semantic web contents with Protègè-2000. *IEEE Intelligent Systems*, 16(2):60–71, March–April 2001.

[42] A. Pivk. Automatic ontology generation from web tabular structures. *AI Communications*, 19(1):83–85, 2006.

[43] A. Pivk, P. Cimiano, and Y. Sure. From tables to frames. In *Proceedings of the Third International Semantic Web Conference (ISWC'04)*, pages 166–181, Hiroshima, Japan, November 2004.

[44] B. Popov, A. Kiryakov, D. Ognyanoff, D. Manov, and A. Kirilov. KIM — a semantic platform for information extraction and retrieval. *Natural Language Engineering*, 10(3-4):375–392, 2004.

[45] S. Sendhilkumar and T. V. Geetha. Personalized ontology for web search personalization. In *Proceedings of the First Bangalore Annual Compute Conference (Compute'08)*, pages 1–7, Bangalore, India, January 2008.

[46] A. Sieg, B. Mobasher, and R. Burke. Web search personalization with ontological user profiles. In *Proceedings of the 16th ACM Conference on Conference on Information and Knowledge Management (CIKM'07)*, pages 525–534, Lisbon, Portugal, November 2007.

[47] A. Singh and K. Nakata. Hierarchical classification of web search results using personalized ontologies. In *Proceedings of the Third International Conference on Universal Access in Human-Computer Interaction*, Las Vegas, Nevada, July 2005.

[48] SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/, 2008.

[49] P. Spyns, D. Oberle, R. Volz, J. Zheng, M. Jarrar, Y. Sure, R. Studer, and R. Meersman. OntoWeb—a semantic web community portal. In *Proceedings of the 5th International Conference on Practical Aspects of Knowledge Management (PAKM'02)*, pages 189–200, Vienna, Austria, December 2002.

[50] K.-C. Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3):422–433, 1979.

[51] C. Tao and D.W. Embley. Automatic hidden-web table interpretation, conceptualization, and semantic annotation. *Data & Knowledge Engineering.* (in press).

[52] C. Tao and D.W. Embley. Automatic hidden-web table interpretation by sibling page comparison. In *Proceedings of the 26th International Conference on Conceptual Modeling (ER'07)*, pages 560–581, Auckland, New Zealand, November 2007.

[53] C. Tao and D.W. Embley. Seed-based generation of personalized bio-ontologies for information extraction. In *Proceedings of the First International Workshop on Conceptual Modelling for Life Sciences Applications (CMLSA'07)*, pages 74–84, Auckland, New Zealand, November 2007.

[54] X. Tao, Y. Li, N. Zhong, and R. Nayak. Ontology mining for personalizedweb information gathering. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI'07)*, pages 351–358, Silicon Valley, California, November 2007.

[55] A. Tengli, Y. Yang, and N.L. Ma. Learning table extraction from examples. In *Proceedings the 20th International Conference on Computational Linguistics (COLING'04)*, pages 987–993, Geneva, Switzerland, August 2004.

[56] Y.A. Tijerino, D.W. Embley, D.W. Lonsdale, Y. Ding, and G. Nagy. Toward ontology generation from tables. *World Wide Web: Internet and Web Information Systems*, 8(3):251–285, September 2004.

[57] M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna. MnM: Ontology driven semi-automatic and automatic support for semantic

markup. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW'02)*, pages 379–391, Siguenza, Spain, October 2002.

[58] M. Vickers. Ontology-based free-form query processing for the semantic web. Master's thesis, Brigham Young University, 2006.

[59] X. Wang. *Tabular Abstraction, Editing, and Formatting.* PhD thesis, Univeristy of Waterloo, 1996.

[60] Y. Wang and J. Hu. A machine learning based approach for table detection on the web. In *Proceedings of the 11th International Conference on World Wide Web (WWW'02)*, pages 242–250, Honolulu, Hawaii, May 2002.

[61] Y. Wang, J. Völker, and P. Haase. Towards semi-automatic ontology building supported by large-scale knowledge acquisition. In *AAAI Fall Symposium On Semantic Web for Collaborative Knowledge Acquisition*, volume FS-06-06, pages 70–77, Arlington, Virginia, October 2006.

[62] Worm base! http://www.wormbase.org, 2005.

[63] XML Path Language (XPath). http://www.w3.org/TR/xpath, 2006.

[64] W. Yang. Identifying syntactic differences between two programs. *Software Practice and Experience*, 21(7):739–755, 1991.

[65] L. Yong and G. Li. Research and realization of personalized search engine based on ontology. In *Proceedings of the IFIP International Conference on Network and Parallel Computing Workshops (NPC'07)*, pages 1016–1020, Dalian, China, September 2007.

[66] R. Zanibbi, D. Blostein, and J.R. Cordy. A survey of table recognition. *International Journal of Document Analysis and Recognition*, 7(1):1–16, 2004.

[67] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *Proceedings of the 14th International Conference on World Wide Web (WWW'05)*, pages 76–85, Chiba, Japan, May 2005.