Enabling a Web of Knowledge [1]

Cui Tao [a], David W. Embley [a,*], Stephen W. Liddle [b]

[a] *Department of Computer Science, Brigham Young University, Provo, UT, 84602, USA*
[b] *Information Systems Department, Brigham Young University, Provo, UT, 84602, USA*

Abstract

A major obstacle impeding progress on the "web of data" is content creation—a difficult, tedious, and time-consuming task. How do we make human-scalable, user-friendly tools to enable the web of data? Content integrity is also a major concern. How do we engender confidence in results returned from the web of data? Although seemingly unrelated, we show in this paper that it is exactly their relationship that is the key to solving both problems. As we show in this paper, we can semi-automatically derive both data and metadata from data-rich web pages to create a web of data that we then superimpose over these data-rich web pages. We link the web of data to the current web of pages, resulting in a higher-order "web of knowledge." This web of knowledge provides provenance and thus engenders the confidence necessary to raise the level of the web from "data" to "knowledge." We focus mainly on two prototype tools we have implemented: (1) TISP—a tool to automatically generate ontologies for data-rich, machine-generated web pages and annotate these pages with respect to these generated ontologies and (2) FOCIH—a tool to semi-automatically generate user-specified ontologies and annotate web pages with respect to these user-specified ontologies. We also briefly survey a suite of tools we and others are creating that have the potential to further enable this web of knowledge. As a measure of how successful these tools are and potentially can be, we summarize precision and recall results, which indicate the degree of scalability we can hope to achieve in enabling a web of knowledge.

Key words: web of knowledge, web of data, semantic web, ontology generation, semantic annotation, information extraction, information harvesting, table interpretation

1. Introduction

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 potentially many of the returned pages 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.

Figure 1 shows an example of the *Web of Knowledge* (the *WoK*) we envision. The upper-left panel shows a free-form query over the WoK. The WoK query processor highlights in green the words it recognizes. It uses these recognized words along with an OWL ontology to generate a SPARQL query over an RDF file that contains the data for the OWL ontology. The SPARQL query returns results in the lower-left panel. The WoK interface then allows users to click on individual results to retrieve the web page from which the WoK originally extracted the result and to display the page with the result highlighted. Users can also check one or more selection boxes to

* Corresponding author. Tel: +1 801 422 6470
Email addresses: ctao@cs.byu.edu (Cui Tao), embley@cs.byu.edu (David W. Embley), liddle@byu.edu (Stephen W. Liddle).

[1] Supported in part by the National Science Foundation under Grant #0414644.

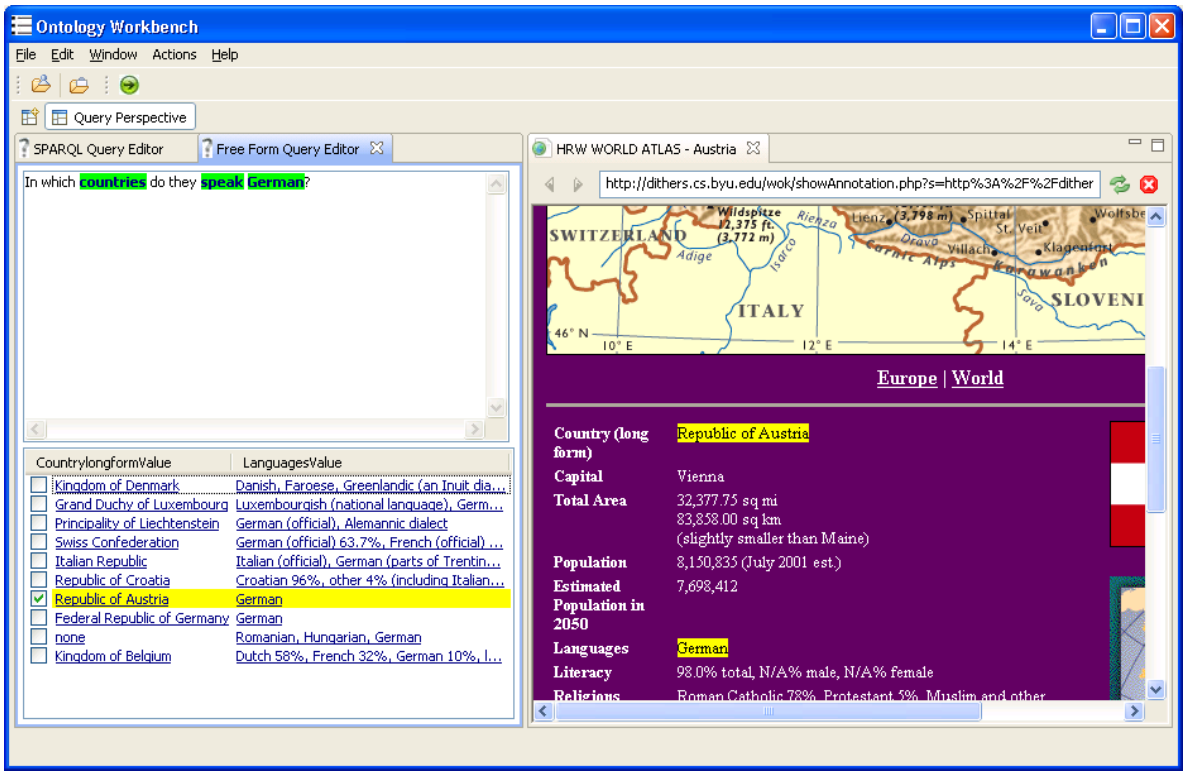

Fig. 1. Screenshot of WoK Prototype

tell the WoK that they want several results highlighted. The panel on the right in Figure 1 displays the the web page with requested results highlighted (or if the requested results are in several web pages, the panel contains a cascade of web pages, each with highlighted results).

Achieving this vision requires that we address and overcome some interesting challenges. (1) How does the WoK get its ontologies? People could manually create them, but this is highly labor intensive and does not scale to the size of the web. Creating 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 thousands of domain ontologies. In addition, different users have different views even for the same domain. Therefore, they may want their data represented and queried in different ways. To satisfy this desire, the WoK should allow for user-specific ontologies, which further multiplies the number of needed ontologies. All of this makes the ontology creation process challenging and suggests that automatic techniques or semi-automatic techniques with a high degree of automation are necessary.

(2) How do WoK ontologies get their data contents? And (3) how do source web pages become annotated with respect to WoK ontologies? Or, equivalently, how does the WoK get its provenance links to source web pages? People could manually add data for each concept in an ontology and establish relationships among the data items, and people could manually link each data item in an ontology to a data item in a source page. Clearly, however, annotating the millions of available data-rich source pages is beyond the ability of any size group of expert knowledge engineers. Moreover, pages change and are generated anew in large quantities, which makes the annotation problem even more massive.

(4) How do users query the WoK? We cannot expect users with ordinary skills and without specialized training to write database queries in the syntax of SQL or SPARQL or any other specialized query language.

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 a reality. This paper provides a step in this direction. Specifically, this paper describes our implementation of two

projects aimed at automating ontology creation and semantic annotation, and it briefly describes some of our other WoK-related projects under way including free-form query specification. Others, besides ourselves, are also working on ontology learning, information extraction, and semantic annotation, and we indicate how these efforts can also contribute to the WoK.

The two projects on which we focus in this paper are TISP (Table Interpretation for Sibling Pages) and FOCIH (Form-based Ontology Creation and Information Harvesting, pronounced *foh*·sī). TISP interprets HTML tables by comparing tables in "sibling pages"—machine-generated pages from the same web site that display data in a similar way. The Holt, Rinehart, and Winston (HRW) World Atlas web site [25], for example, has country pages like the one for Austria in Figure 1 for every country in the world. The page for the Czech Republic in Figure 2 is a sibling page of the page for Austria in Figure 1. These sibling pages contain HTML tables displaying data values for the category labels *Country (long form)*, *Capital*, *Total Area*, etc. Tables in sibling pages are "sibling tables." To interpret a table is to properly associate table category labels with table data values. TISP compares sibling tables to identify and connect nonvarying components (category labels) and varying components (data values). More generally, TISP can identify sibling tables, discover the table structure and layout of sibling tables, interpret tables using structure and layout patterns, and automatically adjust structure and layout patterns as it processes a sequence of machine-generated pages from the same web site.

With the ability to automatically interpret sibling tables, the next step is to automatically generate ontologies from these tables and annotate the data in these tables with respect to the generated ontologies. An extended version of TISP, TISP++, generates OWL ontologies from interpreted tables and then annotates the 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 table structures yield ontological concepts and relationships among these concepts, and associated data values become annotated information. The semantically annotated data leads immediately to queryable data in our envisioned WoK.

TISP and TISP++ provide a fully automatic

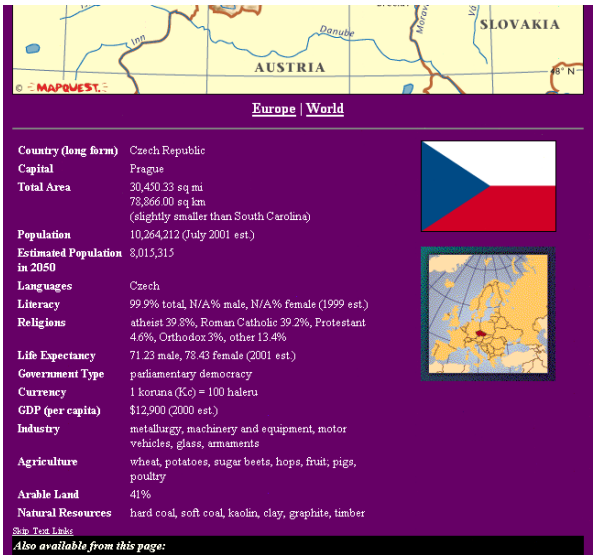

Fig. 2. A Sample Page from HRW World Atlas.

way to transform facts embedded within machine-generated sibling tables into facts accessible by standard query engines. However, the ontologies are generated based on tables as specified in a web site's 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 information gathering and querying, we created FOCIH.

FOCIH 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 an ontology—namely by creating a form to be filled in by the system as it harvests information. FOCIH allows users to create forms that describe the information they wish to harvest. Given a form, FOCIH can generate an ontology, semi-automatically "learn" to match information in a web page with the ontology, and, for machine-generated sibling pages from a web site, harvest information and annotate the pages of the web site with respect to the ontology. In addition, as an aid to initiating forms, FOCIH can reverse engineer existing tables or ontologies into 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 harvesting data and annotating information according to any view users want—either user-created, or based on TISP-interpreted

tables or on any existing ontology (currently, in our implementation, only OWL ontologies).

We make the following contributions in this paper. (1) We show how to superimpose a web of data over a web of pages and thus show how to raise the level of a web of data to the level of a web of knowledge. (2) We describe implemented, automatic and semi-automatic tools to create a web of knowledge and thus provide scalable ways to create its content. (3) We blaze a path toward the development of a suite of human-scalable, user-friendly tools to enable a web of knowledge.

We present the details of these contributions in this paper as follows. We first explain in Section 2 our underlying annotation framework—OWL files for storing ontologies and RDF files to record the annotation links between these OWL ontologies and data in HTML pages. We then explain in Sections 3 and 4 how TISP and FOCIH work—how they are able to either automatically or semi-automatically generate OWL ontologies and record annotation information for HTML pages in RDF files. Building on TISP and FOCIH as examples, we explore in Section 5 how additional work we and others are doing can contribute to the human-scalability issues that currently impede WoK creation. In Section 6 we provide some summary results to indicate the likely success of these endeavors. Finally, in Section 7 we summarize and consider future work.

2. Foundational Framework

To provide a foundational framework, we first say what what we mean by "knowledge." We then explain the essence of the foundation for a "Web of Knowledge" (a "WoK") followed by some details about the particular implementation of our WoK prototype.

Following Meadow [29], we think of the atomic elements of the WoK as symbols, which can be substrings (proper or not) of some character string or multimedia components such as images, sound bites, or frame sequences. *Data*, according to Meadow, is an attribute-value pair, which we generalize to concept-name/element pairs. The page in Figure 1 is data-rich, containing, for example, (*Capital*, *Vienna*) and (*Flag*, *<the image of the Austrian flag>*), and much, much more. To become *information*, according to Meadow, data requires context. We view information as conceptualized data—data in a conceptual model. Thus, in Figure 1 we not only have (*Capital*, *Vienna*) but also (*Country*, *Republic of Austria*) and the relationship between the two, giving the information that Vienna is the capital of Austria. *Knowledge* is the same as information, except, according to Meadow, that it carries with it some community-certified agreement that it is correct. For the scale of the web, it does not seem feasible to have committees of experts certify all facts in the web of knowledge. Rather, we rely on the current certification of facts on the web—namely, the believability of pages as they stand as assertions of more or less reputable creators of web content. Thus, our vision of the WoK is one of information (conceptualized data), extracted from web pages, with links for every data item back to the elements from which they are obtained.

Abstractly, we define the *Web of Knowledge* that we envision as a 4-tuple (O, P, D, A) where O is a set of ontologies, P is a set of web pages, D is a set of data elements, and A is a set of annotations. For our use here, we define an *ontology* for the WoK as a triple (C, R, Σ) where C is a set of concepts, and R is a set of n-ary relationships over concepts in C ($n \geq 2$), and Σ is a set of constraints. In first-order logic, each constraint in C is a one-place predicate, each n-ary relationship is an n-place predicate, and each constraint in Σ is a well-formed, closed formula. [2] We populate each ontology in the WoK with elements from D. An annotation $a \in A$ functionally maps a data element $d \in D$ to a page element $p \in P$, where p is an annotatable component of P such as a string, an image, or a sound bite. We do not require that an object $d \in D$ have an associated annotation; thus, the annotations constitute a partial function f from the union of populated ontology concepts to annotated elements in P. Further, we do not require that f be injective; thus multiple ontologies can annotate the same page element.

In our prototype implementation, we ground our ontology language in OWL. One important difference between WoK ontologies and OWL ontologies is that WoK ontologies have n-ary relationships whereas OWL ontologies only allow binary relationships. In the WoK, we want to be able to consider n-ary relationships such as $Country\text{-}Population\text{-}Year(x, y, z)$ so that we can record in a straightforward way the fact that Austria's 2001 population estimate is 8,150,835, and its 2050 estimated popu-

[2] We express ontologies in languages equivalent to description-logics [5] and thus limit them for practicality purposes.

lation is 7,698,412, as Figure 1 asserts. Using a standard transformation, we convert n-ary to binary relationships by introducing an additional concept to represent the n-ary relationship itself along with n functional, binary relationships. Thus, for our population example, we let $CountryPopulationYear(x)$ be a concept and establish the relationships

$CountryPopulationYear\text{-}Country(x, y)$,
$CountryPopulationYear\text{-}Population(x, y)$, and
$CountryPopulationYear\text{-}Year(x, y)$.

As an example, Figure 3 shows a few of the 260 lines of an OWL ontology for the information about countries contained in the HRW web site. The first few lines of the ontology declare name spaces; then beginning on Line 14, classes for the various concepts in the ontology are defined. Observe that the page itself (denoted by its site name *Hrwworldatlas*) as well as each of the category labels (concepts) in the table are OWL classes (Lines 14–18 in Figure 3). Skipping down several lines, we find binary relationships, declared in OWL as object properties. Each has a domain and range as well as an inverse relationship. Lines 50–56 show one of these binary relationships. OWL datatype properties provide for links to data values. Lines 180–183 show the declaration for the country name. Given sibling pages such as the ones in Figures 1 and 2, we can generate this OWL ontology automatically. Section 3 explains how.

For annotations in our prototype implementation, we have an OWL ontology to define them. In our annotation ontology, the top-level superclass is *AnnotatedThing*. The subclasses of *AnnotatedThing* describe the various kinds of things we can annotate, e.g., the subclasses *AnnotatedHTMLText* for HTML documents and *AnnotatedImage* for images. An object property of *AnnotatedThing* is a *CachedResource*. In our WoK implementation we cache all things we annotate—e.g., all HTML pages and all images. This ensures that WoK annotations cannot be foiled by changes in web pages, but it also means that changed pages should be reannotated to make their updated content part of the WoK. For annotated HTML text, we record the character offset of an annotated substring as well as the substring itself. For annotated images, we keep the upper-left and lower-right coordinates of the rectangular subimage being annotated. Because it is possible for an annotated thing to have component parts, we also provide for the possibility that we may need to join these component parts together to form the single thing being annotated. We may wish, for example, to consider geographic coordinate as a single longitude/latitude value. The longitude value and the latitude value, however, may appear in some HTML page in separate cells in a table or separated in the source page in some other way. The ontology, which calls for a single geographic coordinate made up of a longitude-latitude pair, needs both components to represent the single concept. Having component parts of annotated things accommodates this possibility. Besides annotating things themselves, it is also possible to add a free-form description and a free-form comment. Descriptions should describe things, whereas comments may add any additional information of interest.

To populate ontologies and link instances with annotations, we use RDF. Figure 4 shows some of the 565 lines that describe the data and annotations for the ontology in Figure 3 for both the Austria page in Figure 1 and the Czech Republic page in Figure 2. Line 10 provides the default name-space for the ontology with respect to which we are annotating the page, and Line 11 provides the name-space abbreviation "ann" for our annotation ontology. Lines 19–22 give the cached page for Austria an ID, *resource65*; the cached page for the Czech Republic similarly has an ID. Lines 31–64 contain the instances, some of which we show in Figure 4. As we identify instances for an OWL ontology class C, we simply label them C_i (literally, *C_i*) as the ith instance for the class. We associate instances with OWL classes by giving explicit types for each instance. Lines 76-78 show that the instance *Capital_1* represents a capital city. We relate instances in relationships with RDF subject-predicate-object triples. Lines 206–370 give the relationship instances: *Hrwworldatlas_1* relates to *Countrylongform_1*, *Capital_1*, *Totalarea_1*, etc. Lines 374–564 give the annotation declarations. *Countrylongform_1* is in *resource65* beginning at character location 9237, and its HTML text is "Republic of Austria". In the ontology *Hrwworldatlas* (Lines 225–227), its value is also "Republic of Austria", since it is a string.[3] In the *Hrwworldatlas* ontology, the ID *Countrylongform_1* relates to the value "Republic of Austria" through the *Countrylongform-Hrwworldatlas* relationship to the country represented by the ID "#Hrwworldatlas_1". In Section 3, we explain how we generate this RDF file automatically.

[3] For other data types, the representations may be different, e.g., the population string 7,698,412 as HTML text and its integer value 7698412 in the ontology.

```
...
014.   <owl:Class rdf:ID="Hrwworldatlas"/>
015.
016.   <owl:Class rdf:ID="Countrylongform"/>
017.
018.   <owl:Class rdf:ID="Capital"/>
...
050.   <owl:ObjectProperty rdf:ID="Hrwworldatlas-Countrylongform">
051.     <rdfs:domain rdf:resource="#Hrwworldatlas"/>
052.     <rdfs:range rdf:resource="#Countrylongform"/>
053.     <owl:inverseOf>
054.       <owl:ObjectProperty rdf:ID="Countrylongform-Hrwworldatlas"/>
055.     </owl:inverseOf>
056.   </owl:ObjectProperty>
...
180.   <owl:DatatypeProperty rdf:ID="CountrylongformValue">
181.     <rdfs:domain rdf:resource="#Countrylongform"/>
182.     <rdfs:range rdf:resource="&xsd;string"/>
183.   </owl:DatatypeProperty>
...
```

Fig. 3. Partial OWL Ontology for HRW World Atlas.

Sometimes it is necessary to annotate elements with component parts of values that appear in multiple locations within a web page. Though unlikely for FOCIH applications, components parts of values may even come from different web pages. Annotating these more complex values in source documents requires a correspondingly more complex annotation specification. We illustrate this process with the example in Figure 5. The storage of all concatenated values is similar. Suppose that in a source file the information about *GeographicCoordinate* appears as "49 45 N", and "15 30 E" and comes from different places in the web page. In the ontology view, we want to show them as one single value "49 45 N 15 30 E". As Figure 5 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 inserts them as *hasComponent*-properties of the *GeographicalCoordinate_1* declaration.

3. TISP

Automatic ontology generation is non-trivial. Generation from arbitrary documents in the same way humans are able to read, comprehend, and formulate knowledge structures and facts, is virtually impossible, at least for the foreseeable future. Data-rich, semi-structured documents, however, may enable algorithmic solutions. As one line of research, we investigate tables as a possible source for automatic ontology generation. Online tables provide valuable information about what people think are reasonable ways to represent domains.

In our ontology-generation system, TISP, we focus on machine-generated HTML tables. Millions of these tables come from the hidden web—so-called because their data surfaces only in response to submitted queries. Because these tables are machine-generated, those from the same site usually share the same or similar structures. Pages from the same web site that have the same or similar structure are sibling pages, and the corresponding tables in these pages are sibling tables—e.g., sibling tables in Figures 1 and 3. All the pages from the HRW web site have the same basic structure—maps at the top, flags on the right, and a list of category labels and their values. The category labels have the same order, always starting with *Country (long form)*, followed by *Capital*, *Total Area*, *Population*, etc. The data values, however, vary considerably. Each country has a different name, a different capital city, etc.

To interpret machine-generated tables automatically, TISP compares a pair of sibling tables and looks for commonalities for labels and variations for data values. Given matched and mismatched strings in sibling tables, TISP looks for typical table patterns—labels as column headers, labels as row headers, or tables with both column and row headers. In Figures 1 and 3, for example, labels are row headers, which typically have a single column of values, as do the country tables in these examples. TISP also looks for combinations of these patterns such as when lengthy label-value pair tables are formatted with the bottom half of the table displayed

```
...
009.  <rdf:RDF
010.    xmlns   ="http://dithers.cs.byu.edu/owl/ontologies/hrwworldatlas#"
011.    xmlns:ann="http://dithers.cs.byu.edu/owl/ontologies/annotation#"
012.    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
...
019.  <ann:CachedResource rdf:ID="resource65">
020.    <ann:ResourceID>null</ann:ResourceID>
021.    <ann:CachedURI>http://dithers.cs.byu.edu/wok/cache.php?action=get&amp;hash=
          f3c4a774b5d8cc6d90e21deaa3639ed47dbc9b4d</ann:CachedURI>
022.  </ann:CachedResource>
...
031.  <owl:Thing rdf:ID="Hrwworldatlas_1"/>
032.  <owl:Thing rdf:ID="Countrylongform_1"/>
033.  <owl:Thing rdf:ID="Capital_1"/>
034.  <owl:Thing rdf:ID="Totalarea_1"/>
035.  <owl:Thing rdf:ID="Population_1"/>
...
063.  <owl:Thing rdf:ID="Arableland_2"/>
064.  <owl:Thing rdf:ID="Naturalresources_2"/>
...
076.  <owl:Thing rdf:about="#Capital_1">
077.    <rdf:type rdf:resource="&hrwworldatlas;Capital"/>
078.  </owl:Thing>
...
206.  <rdf:Description rdf:about="#Hrwworldatlas_1">
207.    <Hrwworldatlas-Countrylongform rdf:resource="#Countrylongform_1"/>
208.    <Hrwworldatlas-Capital rdf:resource="#Capital_1"/>
209.    <Hrwworldatlas-Totalarea rdf:resource="#Totalarea_1"/>
210.    <Hrwworldatlas-Population rdf:resource="#Population_1"/>
...
225.  <rdf:Description rdf:about="#Countrylongform_1">
226.    <CountrylongformValue rdf:datatype="&xsd;string">Republic of Austria</CountrylongformValue>
227.  </rdf:Description>
...
374.  <rdf:Description rdf:about="#Countrylongform_1">
375.    <ann:inResource rdf:resource="#resource65"/>
376.    <ann:OffsetOnHTMLPage rdf:datatype="&xsd;nonNegativeInteger">9237</ann:OffsetOnHTMLPage>
377.    <ann:HTMLText rdf:datatype="&xsd;string">Republic of Austria</ann:HTMLText>
378.  </rdf:Description>
...
```

Fig. 4. Annotation for Figure 2 for the Ontology in Figure 3.

adjacent to the top half of the table. TISP can also process complex nested tables such as the nested table in Figure 6 from the WormBase site [47]. [4]

An extended version of TISP, TISP++, leverages interpreted tables to generate ontologies that describe the domain covered by these tables. Labels in tables yield ontological concepts, and table structures yield relationships among ontological concepts. Nested table structures like the ones generated from WormBase in Figure 6 are particularly interesting. Since the structure of complex nested tables mirrors the structure of complex nested forms, a good indication of how this works is in the next section where we discuss FOCIH. Here, we explain the process for the flat HRW country tables.

Figure 3 shows part of the generated ontology for the sibling pages interpreted by TISP from the HRW WORLD ATLAS repository [25]. As a default, TISP++ selects for the ontology name "HRW-WORLDATLAS", which is the site name (contents of the HTML title tag) with all the spaces removed. This name, albeit in mixed-case letters to conform to conventions, becomes the name for the ontology. In addition, and most important, this name provides an anchor class to which we attach ontological concepts. Line 14 in Figure 3 shows the OWL class "Hrwworldatlas".

[4] For an in-depth discussion about how TISP processes both simple and complex sibling tables and sibling tables with slight differences, see [40].

```
<rdf:Description rdf:about="#GeographicCoordinate_1"> ...
 <GeographicCoordinateValue rdf:datatype="&xsd;string">49 45 N 15 30 E</GeographicCoordinateValue>
</rdf:Description>
...
<rdf:Description rdf:about="#GeographicCoordinate_1"> ...
 <ann:hasComponent rdf:resource="#GeographicCoordinateComponent_1"/>
 <ann:hasComponent rdf:resource="#GeographicCoordinateComponent_2"/>
</rdf:Description>
...
<ann:AnnotatedHTMLText rdf:ID="GeographicCoordinateComponent_1"> ...
 <ann:inResource rdf:resource="#resource3"/>
 <ann:OffsetOnHTMLPage rdf:datatype="&xsd;string">487</ann:OffsetOnHTMLPage>
 <ann:HTMLText rdf:datatype="&xsd;string">49 45 N</ann:HTMLText>
</ann:AnnotatedHTMLText>
...
<ann:AnnotatedHTMLText rdf:ID="GeographicCoordinateComponent_2"> ...
 <ann:inResource rdf:resource="#resource3"/>
 <ann:OffsetOnHTMLPage rdf:datatype="&xsd;string">530</ann:OffsetOnHTMLPage>
 <ann:HTMLText rdf:datatype="&xsd;string">15 30 E</ann:HTMLText>
</ann:AnnotatedHTMLText>
```

Fig. 5. Sample RDF Annotation for Instance Concatenation

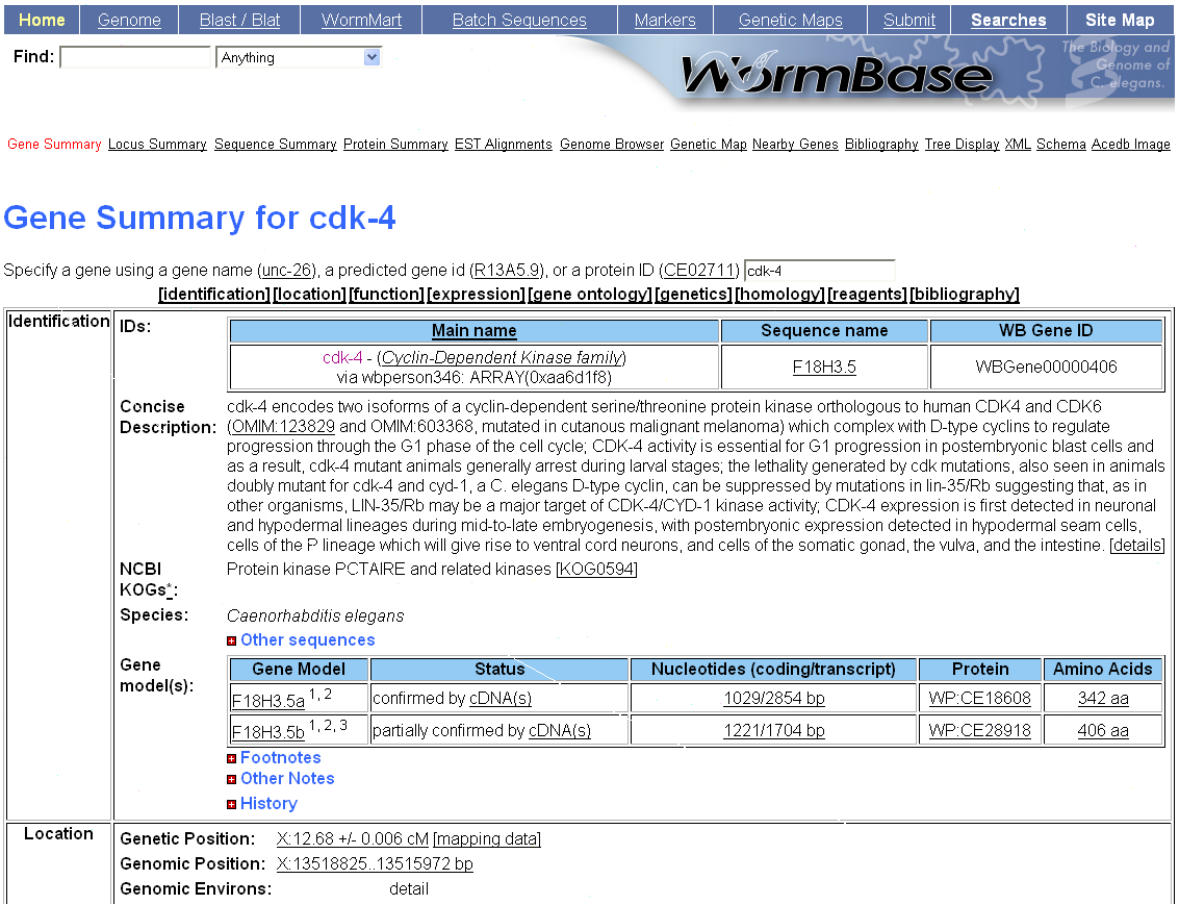

Fig. 6. A Sample Table from WormBase [47].

For each table label, TISP++ generates an OWL class. The label name becomes the class name. To satisfy the OWL syntax, however, TISP++ elides characters such as spaces and parentheses. Thus "Country (long form)" becomes "Countrylongform" as Line 16 in Figure 3 shows. The generated ontology also represents the relationships among the labels. 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 50–56 in Figure 3 show the OWL object property for *Hrwworldatlas-Countrylongform*. If a label is paired with an actual value, TISP++ generates an OWL data type property for the OWL class associated with this label. For example, data type property *Countrylongform* describes the actual value for *Countrylongform*. As Lines 180–183 in Figure 3 show, its domain is *Countrylongform* and its range is string.

After TISP++ generates an ontology according to the structure pattern of a web repository, it automatically annotates the pages from this repository with respect to the generated ontology. Since the interpretation declares the structure pattern, the annotation is straightforward. TISP++ generates an RDF file to record the information as Figure 4 shows. It knows the resources—the pages it processes (e.g., Lines 19–22 in Figure 4), and it knows where on each page it obtains values and thus the character offset for each annotated value in each page (e.g., Line 376 in Figure 4). It picks up each value and makes it a *Thing* (e.g., Lines 31–64). Based on discovered label-value pairs in the table structure, it knows the concept class in which to put each value (e.g., Lines 76–78). The discovered relationships among label-value pairs in the table structure lead to knowing which relationship instances to establish—e.g., the relationship *Hrwworldatlas-Countrylongform* with the instance (*Hrwworldatlas_1*, *Countrylongform_1*) in Lines 206–207. The annotation in Lines 374-378, as an example, ties the instance *Countrylongform_1* to its lexical instance value "Republic of Austria" which starts at character position *9237* in *resource65*.

With the annotated data properly stored in an RDF file (e.g. Figure 4), we can immediately query the annotated data using SPARQL [37]. Although possible to use the WoK in this way, most WoK users will not learn SPARQL and, without some other means to query the WoK, would find it unusable. As Figure 1 shows, users can also query the WoK with free-form queries. We explain how this works in Section 5.

4. FOCIH

TISP++ provides an automatic solution for creating WoK content when sibling tables are available. The generated ontologies, however, represent information only in the same way as the original tables present it and only for the information present in machine-generated tables in sibling pages. How can we provide scalable ways for users to annotate any information they wish according to any view they deem reasonable?

To enable these possibilities for the WoK, we 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 people to collect information. As an everyday activity, people create forms and ask others to fill them in. In this way, specified information can be gathered. FOCIH mimics this everyday activity and allows users to generate a form that describes the information they wish to harvest and then provides a way for users to gather this information from HTML web pages. As information is harvested, FOCIH also annotates it for the WoK.

Given a form, FOCIH generates a corresponding ontology. Form labels become concepts in an ontology. Based on the structure of form components, FOCIH generates relationships and constraints. After FOCIH generates an ontology from a given form, a user can annotate and harvest information with respect to the view represented by the form. Users can harvest information from any arbitrary page, but FOCIH is particularly designed to work with machine-generated pages all from the same site. After creating a form, a user can choose a sample page from a web site of interest and highlight and fill in the information of interest from the sample page into the form. The user-entered values provide FOCIH with 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 site automatically. It can thus annotate all the information in these pages with respect to the ontology.

The form-creation mode of operation provides users with an intuitive method for defining different kinds of form features. FOCIH has five basic form elements from which users construct forms: *single-*

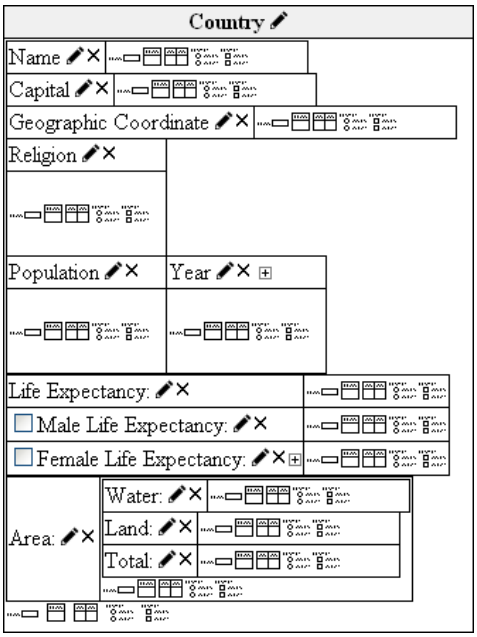

Fig. 7. A Sample Form.

label/single-value element, *single-label/multiple-value element*, *multiple-label/multiple-value element*, *mutually-exclusive choice element*, and *non-exclusive choice element*. Users can also choose to nest forms inside any of the form elements and can thus represent complex, nested hierarchies of information.

Figure 7 shows an example of form creation. Suppose we are interested in basic information about countries (their names, locations, populations, etc.). We begin with a blank form with an empty title and edit the title, making *Country* the base-form title. To add form elements, users click on a chosen form-element icon. Since these form elements appear at the form's top level and also nested in every form element, users can add form elements in sequence or can create forms nested inside other form elements. Continuing with our example, we want each country to have one name, one capital, and one central geographic coordinate. We thus add three single-label/single-value elements to the form and label them *Name*, *Capital*, and *Geographic Coordinate* as Figure 7 shows. Since we know there might be one or more religions in a country, we choose to use a single-label/multiple-value form element and label it *Religion*. We want to keep track of the population of a *Country* for each of several years for which it might be available. Therefore, we create a multiple-label/multiple-entry field—*Population-Year* pairs for the country, as Figure 7 shows. (Although not needed for our example here, users can append additional columns by clicking on the plus icon.) We are also interested in the life expectancy for people in each country depending on 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 be *Male Life Expectancy* and *Female Life Expectancy*. *Land*, *Water*, and *Total Area* are also of interest. Each *Country* has an *Area*, and the areas of interest are: *Land*, *Water*, and *Total*. We thus nest each kind as a single-label/single-value element within the single-label/single-value element for *Area* as Figure 7 shows.

In the form-filling mode of operation, we annotate a page from a web site with respect to a created form by (simply) filling in the form. FOCIH provides users with an interface in which they can (1) open a web page from which they want to collect information, (2) highlight the value or values of interest for each form field, and (3) copy and paste those values into created forms.

Figure 8 shows an example of annotating values using a form. The left-hand side shows the filled-in form for the sample web page on the right-hand side. To annotate the string "Prague" for the form field "Capital", for example, we highlight the string "Prague" by dragging the mouse over it and then click on the pencil icon in the single-entry *Capital* field. For multiple-entry fields, FOCIH allows users to copy several values into the form field. Religions and population estimates in Figure 8 are examples. For population values, users are responsible to copy values appropriately in rows—10,264,212 and 2001 in the same row and 8,015,315 and 2050 in the same row as Figure 8 shows. Users can also concatenate two or more highlighted values when filling a form by clicking on the plus icon. For example, suppose a web site presents *Geographic Coordinate* information by listing longitude and latitude separately, perhaps in two different cells of a table. A user can first highlight the longitude value and then click on the pencil icon and then concatenate the latitude value with the longitude value to form a single (compound) value by clicking on the plus icon.

From a created form, FOCIH can generate an OWL ontology inferred from the form. FOCIH first generates a WoK ontology, the ontology defined as a 4-tuple Section 2. It then converts the ontology to OWL. Among other advantages of an intermediate WoK ontology is the direct correspondence between

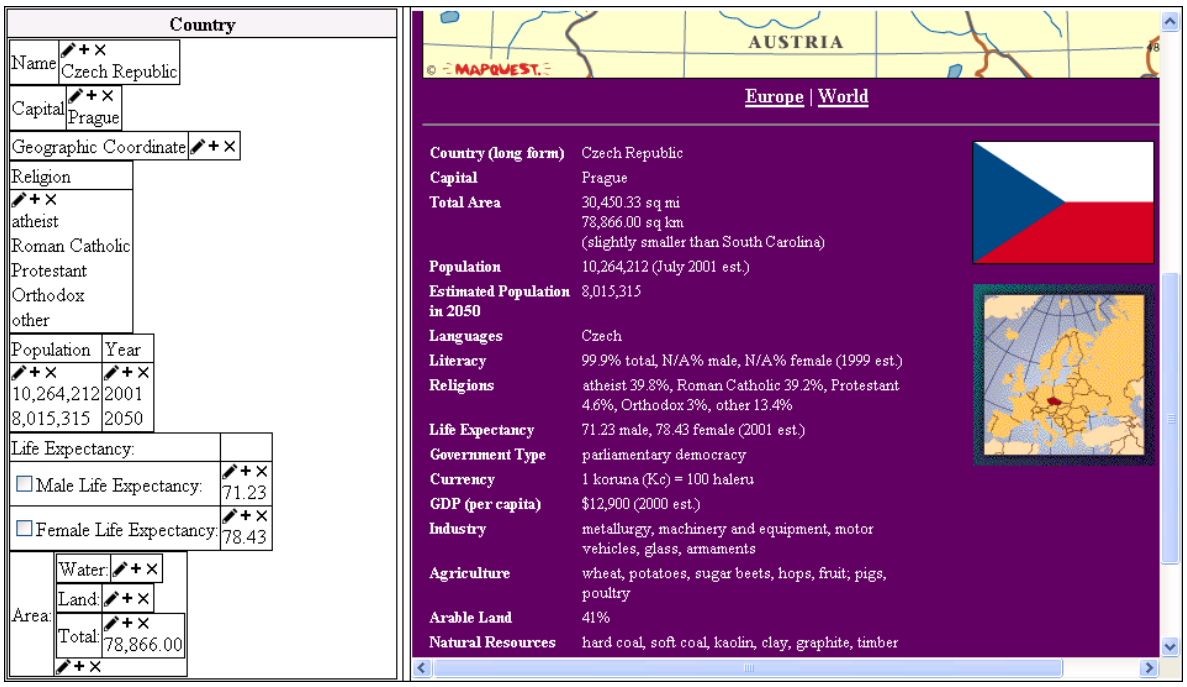

Fig. 8. A Filled-in Form for a Source Data Page.

form elements and WoK ontologies for such features as n-ary relationships and nested forms. Figure 9 graphically shows the result of converting the form in Figure 7 and its filled-in version in Figure 8 to an intermediate WoK ontology.

Every label in the form represents a concept in the WoK ontology—the label becomes the name for the concept. In the graphical representation for a WoK ontology, named boxes represent concepts—dashed boxes for lexical concepts and solid boxes for non-lexical concepts. Lexical concepts are for form-fields with data values, and non-lexical concepts are for form-fields of concepts with nested forms, including the base-form framework in which the top-level form is nested. Thus, *Country* and *Area* are non-lexical concepts in our example, whereas *Name*, *Capital*, and all the rest are lexical concepts. In WoK ontologies the instances in both lexical and non-lexical concepts are object identifiers, but lexical identifiers reference lexical values while non-lexical concepts do not.

How FOCIH generates relationships among the concepts depends on the choice of form elements and their layout with respect to each other. In the graphical representaiton of WoK ontologies, lines represent relationships. Lines with arrowheads are functional from tail concept to head concept. A small letter "o" ("o" for optional) near the connection between a concept C and a relationship R denotes that the participation in relationship R of instances in C is optional.

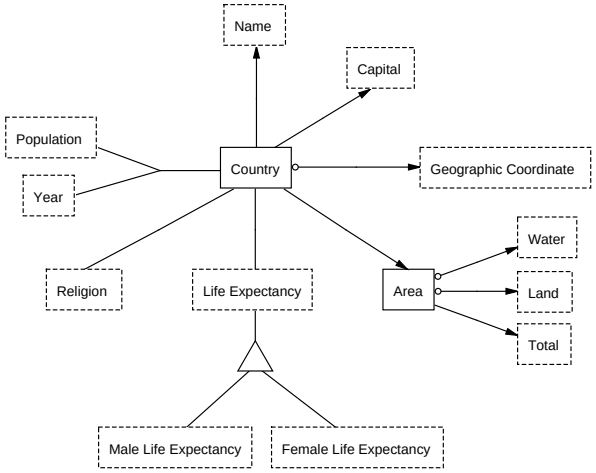

Fig. 9. Graphical View of the Generated Ontology

– Between each form element E and a single-label/single-value form element S nested within E, FOCIH generates a functional, binary relationship from E to S. Thus, since all top-level form elements are nested inside the form's titled framework, FOCIH generates functional, binary relationships from *Country* to *Name*, from *Country* to *Capital*, from *Country* to *Geograph-*

ical Coordinate, and from *Country* to *Area* as Figure 9 shows. For the elements nested inside *Area* FOCIH also generates functional, binary relationships.

- Between each concept C and each single-label/multiple-value form element S nested inside C, FOCIH generates a non-functional binary relationship between C and S. Thus FOCIH accommodates the possibly many *Religions* for each *Country* as Figure 9 shows.
- Between each concept C and each multiple-label/multiple-value form element M nested inside C, FOCIH generates either an n-ary relationship or a set of binary relationships. If M is not the only form element in the form, FOCIH generates an n-ary relationship, otherwise it generates a set of binary relationships. Thus, FOCIH generates an n-ary relationship among *Country*, *Population*, and *Year*. [5]
- Between each concept C and each choice form element E nested inside C, FOCIH generates a non-functional binary relationship between C and E. Thus FOCIH generates a non-functional binary relationship between *Country* and *Life Expectancy* as Figure 9 shows.
- 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. A triangle denotes a WoK ontology generalization/specialization with the apex connected to the generalization and the opposite base connected to the specializations. For exclusive choice elements, a plus symbol appears in the triangle, and imposes a pair-wise-disjoint constraint on the specialization concepts. For the example in Figure 8, FOCIH 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 nested generalizations must match upper-level specialization labels.

Although FOCIH is able to generate all concepts, all relationships among concepts, and all generalization/specialization hierarchies, it can generate only some of the constraints that might be desirable. FOCIH knows, for example, that relationship constraints from parent concept 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 relationship 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 9, the *Name-Country* relationship is not bijective. FOCIH, however, can later modify constraints based on observations as FOCIH harvests information from source documents. The optional specifications on the three relationships in Figure 9 appear initially because FOCIH observes that the first page from which it harvests information (i.e., the page in Figure 8) has no *Geographic Coordinate*, no *Water* area, and no *Land* area.

From the generated, 4-tuple WoK ontology, we can generate an OWL ontology. Figure 10 shows part of the 228-line generated OWL ontology for the form in Figure 7. Every concept in the WoK ontology becomes an OWL class. In Figure 10, Lines 14–46 include all these declarations. All specialization concepts in the WoK ontology become subclasses. Lines 24–28 in Figure 10 show the declaration for the specialization *MaleLifeExpectancy*. All relationships in the WoK ontology resolve into object properties. The binary relationships have a domain and range and an inverse as Lines 50–56 in Figure 10 show. A functional declaration (e.g., Line 58) accompanies those relationships that are functional. Each n-ary relationship ($n \geq 3$) yields a new concept and n functional binary relationships. Line 46 in Figure 10 shows the new concept *CountryPopulationYear*, and Lines 98–106 show one of the functional binary relationships—the one for *Country*. Finally, all lexical concepts have a datatype property for their values. Lines 223–226 show the datatype property for the country-name value. The name for a datatype property is the concept name concatenated with "Value". Line 223 in Figure 10 shows that the name of the value for the *Name* concept is *NameValue*.

[5] Our example does not illustrate the case of a multiple-label form element by itself with no other form elements. As an example, consider a form that has a multiple-label form element by itself nested inside a form framework whose title is *Country*. The labels could be *Name*, *Capital*, and *Population*, and the rows in the multiple-label field would be various country names along with their capitals and populations. In this case, FOCIH would generate these functional binary relationships: from *Country* to *Name*, *Country* to *Capital*, and *Country* to *Population*.

```
...
014.   <owl:Class rdf:ID="Country"/>
015.
016.   <owl:Class rdf:ID="Area"/>
...
024.   <owl:Class rdf:ID="LifeExpectancy"/>
025.
026.   <owl:Class rdf:ID="MaleLifeExpectancy">
027.     <rdfs:subClassOf rdf:resource="#LifeExpectancy"/>
028.   </owl:Class>
...
046.   <owl:Class rdf:ID="CountryPopulationYear"/>
...
050.   <owl:ObjectProperty rdf:ID="Country-Area">
051.     <rdfs:domain rdf:resource="#Country"/>
052.     <rdfs:range rdf:resource="#Area"/>
053.     <owl:inverseOf>
054.       <owl:ObjectProperty rdf:ID="Area-Country"/>
055.     </owl:inverseOf>
056.   </owl:ObjectProperty>
057.
058.   <owl:FunctionalProperty rdf:about="#Area-Country"/>
...
098.   <owl:ObjectProperty rdf:ID="CountryPopulationYear-Country">
099.     <rdfs:domain rdf:resource="#CountryPopulationYear"/>
100.    <rdfs:range rdf:resource="#Country"/>
101.    <owl:inverseOf>
102.      <owl:ObjectProperty rdf:ID="Country-CountryPopulationYear"/>
103.    </owl:inverseOf>
104.  </owl:ObjectProperty>
105.
106.  <owl:FunctionalProperty rdf:about="#CountryPopulationYear-Country"/>
...
223.  <owl:DatatypeProperty rdf:ID="NameValue">
224.    <rdfs:domain rdf:resource="#Name"/>
225.    <rdfs:range rdf:resource="&xsd;string"/>
226.  </owl:DatatypeProperty>
```

Fig. 10. Partial Generated OWL Ontology for the Form in Figure 7.

Harvesting information for a single hand-annotated page is immediate. FOCIH simply puts the values and relationships among values in the filled-in form into the generated WoK ontology. As an example, Figure 11 shows several of the 304 lines of the generated RDF for the HRW web page in Figure 2. Transforming values from filled-in FOCIH forms into RDF is straightforward. Concept values become *owl:Thing*s (e.g., Lines 29–30 in Figure 2). Each value is also associated with an OWL ontology class corresponding to the concept (e.g., Lines 59-67). Instances in specializations also appear in their generalizations since the meaning of a specialization is that its instance set is a subset of its generalization instance set. Observe therefore that the life-expectancy things declared in Lines 29–30 are both in the *LifeExpectancy* class (Lines 61 and 66), which is the generalization, as well as in the specialization classes—*MaleLifeExpectancy* (Line 60) and *FemaleLifeExpectancy* (Line 65). Also in a straightforward way, relationship instances become relationship instances in a generated RDF file. Lines 123–126 show some relationship instances. Lines 146–148 show a relationship between *MaleLifeExpectancy_1* and its typed value. Finally, adding the annotation information for lexical values is also straightforward. Lines 226–230 record the information for the annotation of *MaleLifeExpectancy_1*: its cached web page (*resource49*), its HTML text string (*71.23*), and its character offset on this cached page (*9624*), which is known as a result of having been highlighted, copied, and pasted into the form.

More interesting than just storing hand-annotated information, and more useful in terms of scalability, is FOCIH's ability to recognize patterns for harvesting information from an annotated page and thus to be able to automatically harvest the same kind of information from machine-generated sibling

```
...
029.   <owl:Thing rdf:ID="MaleLifeExpectancy_1"/>
030.   <owl:Thing rdf:ID="FemaleLifeExpectancy_1"/>
...
059.   <owl:Thing rdf:about="#MaleLifeExpectancy_1">
060.     <rdf:type rdf:resource="&country;MaleLifeExpectancy"/>
061.     <rdf:type rdf:resource="&country;LifeExpectancy"/>
062.   </owl:Thing>
063.
064.   <owl:Thing rdf:about="#FemaleLifeExpectancy_1">
065.     <rdf:type rdf:resource="&country;FemaleLifeExpectancy"/>
066.     <rdf:type rdf:resource="&country;LifeExpectancy"/>
067.   </owl:Thing>
...
123.   <rdf:Description rdf:about="#Country_1">
124.     <Country-Area rdf:resource="#Area_1"/>
125.     <Country-LifeExpectancy rdf:resource="#MaleLifeExpectancy_1"/>
126.     <Country-LifeExpectancy rdf:resource="#FemaleLifeExpectancy_1"/>
...
146.   <rdf:Description rdf:about="#MaleLifeExpectancy_1">
147.     <MaleLifeExpectancyValue rdf:datatype="&xsd;string">71.23</MaleLifeExpectancyValue>
148.   </rdf:Description>
...
226.   <rdf:Description rdf:about="#MaleLifeExpectancy_1">
227.     <ann:inResource rdf:resource="#resource49"/>
228.     <ann:OffsetOnHTMLPage rdf:datatype="&xsd;string">9624</ann:OffsetOnHTMLPage>
229.     <ann:HTMLText rdf:datatype="&xsd;string">71.23</ann:HTMLText>
230.   </rdf:Description>
...
```

Fig. 11. RDF Annotation for Generalization/Specialization

pages for a site. FOCIH accomplishes this task by recognizing both paths to instances within HTML DOM trees and the instances themselves. 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.

A user-highlighted value can be the entire DOM-tree node (e.g., "Prague" in Figure 8) or a proper subpart of the string that constitutes the DOM-tree node (e.g., just the populated value in Figure 8). [6] 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 patterns, 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 [15,18]. An *individual pattern* has left and right contexts and a regular-expression instance recognizer. For example, for the highlighted area value "78,866.00", the left context can be "\b" (word boundary) and the right context can be "sq km", the regular-expression recognizer can be decimal number, and the appearance number can be 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 8 has as its left context *sos*, as its right context *eos*, as its regular-expression recognizer ".*" (any string), and as its delimiter "([,;]\s*)"

[6] 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.

(either comma/space or semicolon/space).

To determine patterns, FOCIH first determines whether a pattern is an individual pattern or a list pattern. If only one highlighted value from a DOM-tree node 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 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 DOM-tree node. If FOCIH recognizes some of the context as an instance of one of the regular-expression recognizers in its library of recognizers, FOCIH substitutes the recognized substring in the context by the recognizer. If a highlighted substring can be recognized by a regular-expression recognizer in its 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.

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 8 is an example. For this list FOCIH creates the delimiter expression "[,;]". For more complex cases such as the religions list in Figure 8, 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.

With path recognition and instance recognition, FOCIH can locate the information of interest from all the sibling pages for a site and store it in an RDF file. Assuming FOCIH has harvested information from a number of HRW web pages, we can then query the resulting RDF file with a SPARQL query like the one in Figure 12, that asks for the life expectancy of males in the Czech Republic. In this example, the filter expression finds all instances of *NameValue* that contain the string "Czech". Then through the property *NameValue*, SPARQL can locate all the *Name* instances we are looking for, in our example, *#Name_1*. Further through the property *Country-Name*, SPARQL locates *#Country_1*. Finally through the property *Country-LifeExpectancy*, SPARQL can find the instances *#FemaleLife-Expectancy_1* and *#MaleLifeExpectancy_1*. Then following the property *MaleLifeExpectancyValue*, SPARQL finds the value we are looking for.

In order to display query results within the context of the underlying original source web pages, the WoK system can rewrite the SPARQL query to include annotation information. Figure 13 shows the rewritten query. The original query only retrieves values for male life expectancy and country name. The rewritten query also retrieves offset, annotated string, and source-document URI fields for both male life expectancy and country name. Using this information, our query tool can present the source document side by side with the SPARQL query and its results, as Figure 14 shows. The example in Figure 14 shows only two columns of results (*MaleLife-ExpectancyValue* and *NameValue*) because the query tool automatically hides annotation-layer output fields. Note that the tool uses the annotation-layer information to create hyperlinks from result data to the corresponding original source pages. In Figure 14, the user has clicked on the male life expectancy result, which caused the tool not only to highlight that result (71.23), but also to display the cached source page from which the result came; the annotated value is also highlighted in the web page. Clicking on a checkbox next to a results row causes the tool to highlight all corresponding result values in the source page(s).

5. A Suite of WoK-Creation Tools

In this section we explore additional work we and others are doing that can contribute to the WoK, and, in particular, how the work can contribute to

```
PREFIX country:<http://dithers.cs.byu.edu/owl/ontologies/country#>
SELECT ?MaleLifeExpectancyValue ?NameValue
WHERE {
   ?Country country:Country-LifeExpectancy ?MaleLifeExpectancy ;
          country:Country-Name ?Name  .
   ?MaleLifeExpectancy country:MaleLifeExpectancyValue ?MaleLifeExpectancyValue .
   ?Name country:NameValue ?NameValue .
   FILTER regex (?NameValue, "czech", "i")
}
```

Fig. 12. Sample SPARQL Query

```
PREFIX country:<http://dithers.cs.byu.edu/owl/ontologies/country#>
PREFIX ann: <http://dithers.cs.byu.edu/owl/ontologies/annotation#>
SELECT ?MaleLifeExpectancyValue ?MaleLifeExpectancyAnnOffset ?MaleLifeExpectancyAnnText
   ?MaleLifeExpectancyAnnURI ?NameValue ?NameAnnOffset ?NameAnnText ?NameAnnURI
WHERE {
   ?Country country:Country-LifeExpectancy ?MaleLifeExpectancy ;
          country:Country-Name ?Name  .
   ?MaleLifeExpectancy country:MaleLifeExpectancyValue ?MaleLifeExpectancyValue;
      ann:OffsetOnHTMLPage ?MaleLifeExpectancyAnnOffset ;
      ann:HTMLText ?MaleLifeExpectancyAnnText ;
      ann:inResource ?MaleLifeExpectancyAnnResource .
   ?MaleLifeExpectancyAnnResource
      ann:CachedURI ?MaleLifeExpectancyAnnURI .
   ?Name country:NameValue ?NameValue;
      ann:OffsetOnHTMLPage ?NameAnnOffset ;
      ann:HTMLText ?NameAnnText ;
      ann:inResource ?NameAnnResource .
   ?NameAnnResource
      ann:CachedURI ?NameAnnURI .
   FILTER regex (?NameValue, "czech", "i")
}
```

Fig. 13. Rewritten SPARQL Query

the human-scalability issues surrounding the WoK. Analyzing the human activities involved in FOCIH, we ask how each can be automated, or at least improved so as to mitigate the human effort involved. We briefly explore:

– Automatic adjustment to paths and to list and context patterns during information harvesting (Subsection 5.1);
– Automatic initial form generation (Subsection 5.2); and
– Automatic initial form filling (Subsection 5.3).

Each of these efforts involves significant development work and significant work to integrate it into the WoK. Surprisingly, these efforts also lead to a direction for user-friendly WoK query specification. We therefore briefly explore these opportunities in Subsection 5.4. Finally, in Subsection 5.5 we briefly explain how related work, mostly from other researchers, can contribute to the WoK vision.

5.1. *Automating Harvesting Adjustments*

Even though machine-generated, sibling pages are not as uniform as might be expected. For example, the list of religions for Israel is

Jewish 80.1%, Muslim 14.6% (mostly Sunni Muslim), Christian 2.1%, other 3.2% (1996 est.)

where, in addition to percentage-comma separators, the list also includes parenthetical remarks. Expecting only percentage-comma separators, FOCIH does not harvest information properly from this list.

To address this and similar problems, we run FOCIH in three different modes of operation. (1) Harvest from pages, one-by-one: FOCIH displays each page and fills in the form the best it can; then, a user can either confirm that FOCIH has done so correctly, or can alter the annotation by deleting any values FOCIH filled incorrectly and adding any values FOCIH may have missed and thus fix any problems that arise. (2) Harvest, stopping only when FOCIH recognizes that it fails to properly extract some information for some page: again, a user can fix any

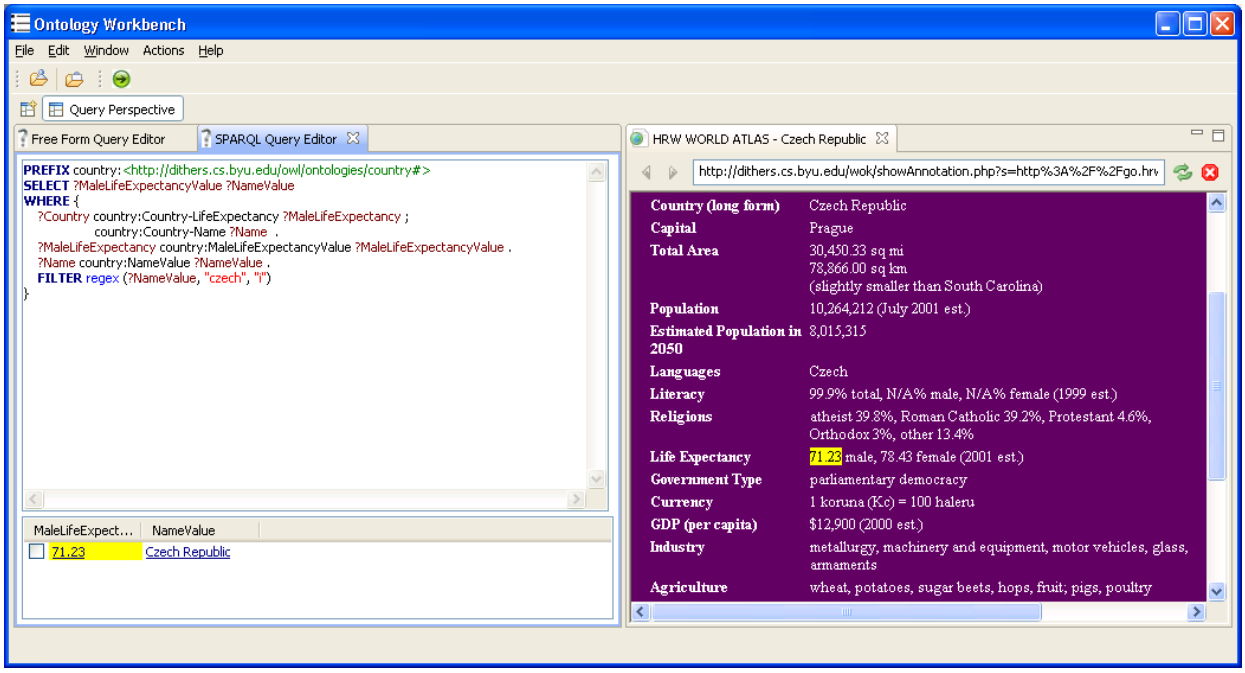

Fig. 14. Sample Query and Results for FOCIH.

incorrect annotation. (3) Harvest without stopping: FOCIH does the best it can, but may make mistakes.

To help with scalability, FOCIH can learn from the corrections a user makes. It can adjust context expressions and delimiter expressions for lists. For our example of religions in Israel, the delimiter expression should not only be percentage/comma delimiters but also percentage/parenthetical-remark/comma delimiters. FOCIH should adjust its context-recognition expressions and its list-delimiter expressions as it harvests information.

5.2. *Automating Form Generation*

For many applications, ontologies already exist. If we could generate forms automatically from these ontologies, we could then use FOCIH to do the annotation and information harvesting. FOCIH users could either use the generated forms directly, or they could modify them starting from a given base to suit their needs.

We have designed and implemented algorithms to reverse-engineer WoK ontologies into XML-schema specifications [1,30]. Since XML nests information like FOCIH does for forms, we can immediately generate nested form specifications. Thus, whenever we can reverse-engineer ontology specifications into a WoK ontology, we can generate a FOCIH form corresponding to the ontology. In our current prototype implementation we can reverse-engineer sibling tables via TISP into WoK ontologies. We also have a prototype implementation that transforms OWL ontologies into WoK ontologies. We have not yet integrated our code for all these systems together, but doing so would let us initialize FOCIH forms given either TISP-resolvable sibling pages such as the one in Figure 6 or from the many available OWL ontologies [33].

5.3. *Automating Initial Form Filling*

The problem of manual labeling for the first (or first and only) web page is a barrier to scalability. Is there a way we can get the system to do the initial labeling for us? Yes, with extraction ontologies, but at a cost—the cost of building these extraction ontologies.

A WoK ontology augmented with instance recognizers is an *extraction ontology*. Instance recognizers contain regular expressions for each lexical concept that recognize common textual items such as dates, times, prices, and numbers. They also contain lexicons that match with items such as countries, cities, and protein names and functions. In addition they make use of context keywords, units, left and right context information, and expected cardinal-

ity distributions to aid in recognizing instances for ontological concepts. Much can and has been said about how to build and use these instance recognizers embedded within extraction ontologies (e.g., [13,16] among several others).

Building instance recognizers is laborious. We have four answers to this legitimate observation. (1) We can build an extensive library of common instance recognizers that can be used directly or specialized for use in some domain. (2) We need not have perfect recognizers because we can augment extraction ontologies with pattern-based extractors—once a few values have been recognized, pattern-based extractors can determine the pattern of a semi-structured page and thereby correctly recognize values beyond those observed directly by instance recognizers. (3) We can automatically update lexicon recognizers by adding additional values found by pattern-based extractors. And (4) we can bootstrap our way up by using FOCIH to help build instance recognizers and lexicons by letting users provide a few sample mappings from a page to an ontology, and from these sample mappings, generate pattern-based annotators and then update recognizers with the newly found information.

5.4. *WoK Query Specification*

As argued earlier, users should not be asked to learn SPARQL in order to be able to query the WoK. Instead, we should provide for free-form query specification as Figure 1 shows. The key to making free-form queries work is not natural-language processing (at least not in the usual sense of natural-language processing), but rather is the application of extraction ontologies to the queries themselves. The essence of the idea is to (1) use an extraction ontology to identify constants, keywords, and keyword phrases in a free-form query; (2) find the ontology that matches best; and (3) embed the query in the implicit hypergraph of the ontology yielding (3a) a join over the relationship paths connecting identified concepts, (3b) a selection on identified constants modified by identified operators, and (3c) a projection on mentioned concepts. (See [3] for a complete explanation about mapping free-form queries to formal database query languages.)

In Figure 1 "countries" is a keyword identifying the country concept in the generated ontology. It is also a keyword identifying the country-name concept in the generated ontology. "Speak" is a keyword for the language concept in the ontology, and "German" is a constant value in the lexicon that lists languages.

Thus, our WoK prototype is able to generate the SPARQL query in Figure 15. Observe that the query in Figure 15 is the augmented SPARQL query. Thus, when this query executes, it produces results as Figure 1 shows. When users click on results, the system intercepts the request, uses the information to find and highlight requested information in cached pages and displays them to users as Figure 1 shows.

Anyone can readily pose free-form queries. To be successful, however, a user does have to guess which keywords, values, and constraint expressions might be available in an extraction ontology for the domain of interest. This is similar to users having to guess keywords and values for current search-engine queries. Since arbitrary free-form queries may not always be successful, we also plan to provide a form-based query language. Based on the ontology and our ability to generate FOCIH forms, we can instead generate ordinary HTML forms typically used to pose queries on the web. In this way the system can guide users in formulating queries for a domain.

5.5. *Related Work*

Developing the WoK we envision is a huge task. Its framework allows for the inclusion of related work on many topics: sibling-page comparison, table interpretation, ontology generation, and automatic semantic annotation. Much more can be done to enhance the human-scalability of the envisioned WoK.

Sibling Page Comparison. Besides our own work, several researchers have also tried to take advantage of sibling pages. RoadRunner [11] 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) to generate extraction templates for the sibling pages. DEPTA [49] compares different records in a page (sibling records) instead of sibling pages and tries to find the extraction template for the record. The approach in [27] 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

```
PREFIX country:<http://dithers.cs.byu.edu/owl/ontologies/country#>
PREFIX ann: <http://dithers.cs.byu.edu/owl/ontologies/annotation#>
SELECT ?CountrylongformValue ?CountrylongformAnnOffset ?CountrylongformAnnText ?CountrylongformAnnURI
       ?LanguagesValue ?LanguagesAnnOffset ?LanguagesAnnText ?LanguagesAnnURI
WHERE {
      ?Country country:Country-Countrylongform ?Countrylongform ;
               country:Country-Languages ?Languages .
      ?Countrylongform country:CountrylongformValue ?CountrylongformValue;
           ann:OffsetOnHTMLPage ?CountrylongformAnnOffset ;
           ann:HTMLText ?CountrylongformAnnText ;
           ann:inResource ?CountrylongformAnnResource .
      ?CountrylongformAnnResource
           ann:CachedURI ?CountrylongformAnnURI .
      ?Languages country:LanguagesValue ?LanguagesValue;
           ann:OffsetOnHTMLPage ?LanguagesAnnOffset ;
           ann:HTMLText ?LanguagesAnnText ;
           ann:inResource ?LanguagesAnnResource .
      ?LanguagesAnnResource
           ann:CachedURI ?LanguagesAnnURI .
      FILTER regex (?LanguagesValue, "german", "i")
}
```

Fig. 15. Generated SPARQL Query

record segmentations.

Table Interpretation. Not to slight the vast amount of work on table processing [17,48], we mention here only work on HTML table interpretation. Several researchers try to differentiate data tables from tables for layout [7,10,21,45]. They use machine-learning methods [10,45], visual level features [21,22], and linguistic features [7]. Other papers [7,19,20,24,28] discuss the HTML table interpretation problem based on simple assumptions and heuristics. The approach in [35] 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 technique in [41] 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. The approach in [22] uses visual boxes instead of HTML tags to interpret HTML tables.

Ontology Generation. In recent years, many researchers have tried to facilitate ontology generation. Excellent editing tools such as Protégé [32] and OntoWeb [38] have been developed to help users create and edit ontologies. Because of the expertise required and the difficulties involved in manual creation, researchers have turned to semi-automatic ontology generation tools. Most efforts so far have been devoted to automatic generation of ontologies from text files. Tools such as OntoLT [6], Text2Onto [9], OntoLearn [31], and KASO [46] 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. To date, the results have not been very satisfactory [34]. Tools such as TANGO [42], and the one developed by Pivk [34] 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 [34] tries to discover semantic labels for table regions and generate an ontology based on a table's structure.

Automatic Semantic Annotation. Existing semantic annotation systems can be classified into pattern-based systems and machine-learning-based systems. Pattern-based systems such PANKOW [8] and Armadillo [14] find entities by discovering patterns. The patterns are either discovered manually or induced semi-automatically with a set of initial manually tagged seed patterns. Systems such as SemTag [12], AeroDAML [26], and KIM [36] use a set of pre-defined rules to locate information of interest. Systems such as S-CREAM [23] and MnM [43] use machine-learning algorithms and natural-language processing methods to locate semantic entities.

6. Scalability

To indicate the likely success of these endeavors to increase human-scalability, we provide some summary results. Details of these results are in [39] for TISP and FOCIH, in [16] for extraction ontologies, and in [3,44] for free-form queries.

We tested TISP in three domains: geopolitical information, molecular biology, and car sales. We considered more than 2,000 tables found in 275 sibling pages across 35 web sites. 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.

The performance of TISP++ depends on the performance of TISP. Given that TISP can interpret tables correctly, TISP++ can automatically generate ontologies and annotate information in interpreted tables correctly. Disagreements may arise, however, about how to reverse-engineer some table structures into ontologies. There is more than one reasonable way, for example, to map tables with both row and column headers into WoK ontologies.

FOCIH can always correctly generate ontologies according to user-created forms. Here, as with TISP, users may disagree among the possible, reasonable mappings. For both TISP and FOCIH, default choices are likely to suffice, but options should be available for users who might prefer alternate views.

How well FOCIH can automatically harvest information from sibling pages depends on how uniform the pages are. As an indication of what might be expected, we tested FOCIH's ability to do instance recognition by considering a number of different web pages. In these web pages, we encountered 71 instance-recognition situations. FOCIH was able to recognize all 25 full-string matches it encountered. It successfully recognized 29 of 38 partial-string matches and 6 of 8 list patterns. In these trials, the overall accuracy of instance recognition was 84.5%.

Extraction ontologies can likely help with human-scalability issues, but only if they can perform reasonably well. Over the course of many years, we have developed our ontology-based information-extraction tool and have tested it on various domains, each with dozens of real-world web pages. Based on approximately 20 domains with which we have experimented we summarize our experience as follows. In simple, unified domains we typically achieve close to 100% precision and recall for almost all fields, while in more complicated or loosely unified domains, the precision and recall for some fields falls off dramatically. For example, we have worked on a genealogical application in which we wanted to extract information from obituaries as they appear in standard newspapers. For this complex domain, our information-extraction engine was only able to achieve about 74% precision for identifying relatives of a deceased person and only about 82% recall for recognizing funeral addresses. In general, however, in nearly 20 domains that contain in total over 200 different class concepts, our extraction engine typically achieves at least 80% accuracy for both precision and recall values on most fields. For over half of the domains, the precision and recall values were above 90%.

For free-form query processing, we have been able to gather some results to indicate how well this part of the WoK might perform. In [44], we describe an experiment in which four subjects each provided five queries on five domains (car ads, real estate, countries, movies, and diamonds) for a total of 100 queries. The recall for identifying concept values to be returned was 89% and for correctly generating conditional operators was 75% while the corresponding precision values were respectively 89% and 88%. Overall, the system interpreted 47% of the queries with perfect accuracy while interpreting an additional 49% with partial accuracy for a total of 96% with some reasonable accuracy.

In another experiment with a more advanced free-form query processor, we gathered additional results [3]. Testing for system performance in finding the predicates of a formal representation for free-form service requests and values for predicate arguments, we considered service requests belonging to the following domains: scheduling appointments with medical doctors, purchasing cars, and renting apartments. We asked subjects to make free-form, natural-language-like service requests belonging to these domains using their own words, but to only

ask conjunctive queries with positive literals.[7] We received a total of 31 requests, which included 548 constraints and 170 constant values. Overall, recall results averaged 98% for predicates and 95% for arguments of predicates, and precision results approached 100% for both predicates and arguments. When the system selects the right ontology for a service request, the system almost cannot select irrelevant predicates and arguments for an ontology that is narrowly focused on the service and thus has little in the way of alternative choices that may be ambiguous.

7. Summary

In this paper, we explained our vision of a web of knowledge—a web of data superimposed over a web of pages. To actually realize this vision, human scalability issues are at the forefront. We cannot expect web content providers to annotate pages they create or cause to be generated without assistance. We therefore strive to create tools for semi-automatic ontology generation and semi-automatic annotation of page content with respect to generated ontologies. Our desire is to make these semi-automatic processes as automatic as possible, thereby shifting the burden as much as possible away from human effort in tedious and time-consuming work. For WoK users, we also address human scalability issues, realizing that the effort required to learn formal query languages is beyond the time, patience, and expertise most users of the WoK have.

We focused particularly on TISP and FOCIH, two systems directed specifically at mitigating the problems of WoK realization. We also explained how our previous work on extraction ontologies plays an interesting role in helping to make the WoK realizable. TISP provides a solution to automatic interpretation for sibling tables as typically found in many machine-generated web pages. Based on TISP, TISP++ offers a way to automate ontology generation given an interpreted table and enables automatic semantic annotation for interpreted tables. FOCIH supports 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. Automated initial form generation from existing ontologies and extraction ontologies that can automatically initialize annotation specification can further enhance FOCIH and shift the burden of ontology creation and web-page annotation even further toward full automation. Extraction ontologies also offer a key that opens the way for free-form query specification, which thus opens the door to WoK usage by ordinary people, untrained in sophisticated, technical query languages.

Results from preliminary scalability experimentation are encouraging. If with 90% accuracy or better, TISP-like systems can interpret tables and turn them into ontologies, FOCIH-like systems can allow users to harvest information with respect to personalized ontological views, and extraction ontologies can recognize and extract instances in free-form queries and in data-rich web pages, WoK creation and usage may be feasible. Moreover, as a community, encouraging results can bolster further development and spur us on to create the kinds of tools necessary to enable the WoK.

As for our own future work, several 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, not only in tables but also in sibling pages in general. (2) So that we can draw from the large numbers of existing ontologies, we intend to integrate our various projects regarding reverse-engineering ontologies to forms. This would provide a way for FOCIH to harvest and annotate information with respect to existing ontologies. (3) We see the possibility of being able to semi-automatically convert generated ontologies into extraction ontologies. Based on the information that FOCIH harvests for each concept in an ontology, we can create the beginnings of a lexicon for these instance recognizers or recognize common data items and thus automatically select instance recognizers for concepts. 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. (4) Finally, we must also worry about making the WoK itself perform reliably and in real-time for end users. We have defined semantic indexing [2], with which we can

[7] These are the common kinds of queries asked by typical web users. To avoid technical terms (e.g. "conjunctive" and "positive literals"), we provided users with illustrative examples of what not to ask (e.g. not "at 10:00 am *or* after 3:00 pm" and not "*not* at 9:00 am").

quickly find applicable ontologies for user queries, and we have considered large-scale caching, following the lead of modern search engines. But more is required to make the WoK work well.

There is a great deal left to do, both in our work and along many other lines of research as well. But we are optimistic that given the results we report in this paper, together with results coming from many other researchers, the Web of Knowledge can indeed be realized.

References

[1] R. Al-Kamha, D. Embley, S. Liddle, Foundational data modeling and schema transformations for xml data engineering., in: Proceedings of the 2nd International United Information Systems Conferences (UNISCON'08), Klagenfurt, Austria, 2008.

[2] M. Al-Muhammed, D. Embley, S. Liddle, Y. Tijerino, Bringing web principles to services: Ontology-based web services, in: Proceedings of the Fourth International Workshop on Semantic Web for Services and Processes (SWSP'07), Salt Lake City, Utah, 2007.

[3] M. Al-Mumammed, D. Embley, Ontology-based constraint recognition for free-form service requests, in: Proceedings of the 23rd International Conference on Data Engineering (ICDE'07), Istanbul, Turkey, 2007.

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

[5] F. Baader, W. Nutt, Basic description logics, in: F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider (eds.), The Description Logic Handbook, chap. 2, Cambridge University Press, Cambridge, UK, 2003, pp. 43–95.

[6] P. Buitelaar, D. Olejnik, 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), Heraklion, Greece, 2004.

[7] H. Chen, S. Tsai, J. Tsai, Mining tables from large scale HTML texts, in: Proceedings of the 18th International Conference on Computational Linguistics (COLING'00), Saarbrücken, Germany, 2000.

[8] P. Cimiano, S. Handschuh, S. Staab, Towards the self-annotating web, in: Proceedings of the 13th International Conference on World Wide Web (WWW'04), New York, New York, 2004.

[9] P. Cimiano, 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), Alicante, Spain, 2005.

[10] W. Cohen, M. Hurst, L. 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), Honolulu, Hawaii, 2002.

[11] V. Crescenzi, G. Mecca, 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), Rome, Italy, 2001.

[12] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, K. Mccurley, S. Rajagopalan, 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) (2003) 115–132.

[13] Y. Ding, D. Embley, S. 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), Beijing, China, 2006.

[14] A. Dingli, F. Ciravegna, 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, Sanibel Island, Florida, 2003.

[15] D. Embley, Programming with data frames for everyday data items, in: National Computer Conference, Anaheim, California, 1980.

[16] D. Embley, D. Campbell, Y. Jiang, S. Liddle, D. Lonsdale, Y.-K. Ng, R. Smith, Conceptual-model-based data extraction from multiple-record Web pages, Data & Knowledge Engineering 31 (3) (1999) 227–251.

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

[18] D. Embley, Y. Jiang, Y.-K. Ng, Record-boundary discovery in Web documents, in: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD'99), Philadelphia, Pennsylvania, 1999.

[19] D. Embley, C. Tao, S. Liddle, Automatically extracting ontologically specified data from HTML tables with unknown structure, in: Proceedings of the 21st International Conference on Conceptual Modeling (ER'02), Tampere, Finland, 2002.

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

[21] W. Gatterbauer, 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), Boston, Massachusetts, 2006.

[22] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, B. Pollak, Towards domain-independent information extraction from web tables, in: Proceedings of the 16th International World Wide Web Conference (WWW'07), Banff, Canada, 2007.

[23] S. Handschuh, S. Staab, F. Ciravegna, S-CREAM — Semi-automatic CREAtion of Metadata, in: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW'02), Siguenza, Spain, 2002.

[24] W. Holzinger, B. Krüpl, M. Herzoge, Using ontologies for extracting product features from web pages, in:

Proceedings of the Fifth International Semantic Web Conference (ISWC'06), Athens, Georgia, 2006.

[25] HOLT, RINENART and WINSTON world atlas, http://go.hrw.com/atlas/norm_htm/world.htm.

[26] P. Kogut, 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, Victoria, British Columbia, 2001.

[27] K. Lerman, L. Getoor, S. Minton, 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), Paris, France, 2004.

[28] S. Lim, 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), Kansas City, Missouri, 1999.

[29] C. Meadow, Text Information Retrieval Systems, Academic Press, San Diego, California, 1992.

[30] W. Mok, D. Embley, Generating compact redundancy-free XML documents from concptual-model hypergraphs, IEEE Transactions on Knowledge and Data Engineering 18 (8) (2006) 1082–1096.

[31] R. Navigli, P. Velardi, A. Cucchiarelli, F. Neri, Quantitative and qualitative evaluation of the OntoLearn ontology learning system, in: Proceedings of the 20th International Conference on Computational Linguistics, Geneva, Switzerland, 2004.

[32] N. Noy, M. Sintek, S. Decker, M. Crubezy, R. Fergerson, M. Musen, Creating semantic web contents with Protègè-2000, IEEE Intelligent Systems 16 (2) (2001) 60–71.

[33] Protoge ontology library, http://protegewiki.stanford.-edu/index.php/Protege_Ontology_Library.

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

[35] A. Pivk, P. Cimiano, Y. Sure, From tables to frames, in: Proceedings of the Third International Semantic Web Conference (ISWC'04), Hiroshima, Japan, 2004.

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

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

[38] P. Spyns, D. Oberle, R. Volz, J. Zheng, M. Jarrar, Y. Sure, R. Studer, R. Meersman, OntoWeb—a semantic web community portal, in: Proceedings of the 5th International Conference on Practical Aspects of Knowledge Management (PAKM'02), Vienna, Austria, 2002.

[39] C. Tao, Ontology generation, information harvesting and semantic annotation for machine-generated web pages, Ph.D. thesis, Brigham Young University, Provo, Utah (December 2008).

[40] C. Tao, D. Embley, Automatic hidden-web table interpretation by sibling page comparison, in: Proceedings of the 26th International Conference on Conceptual Modeling (ER'07), Auckland, New Zealand, 2007.

[41] A. Tengli, Y. Yang, N. Ma, Learning table extraction from examples, in: Proceedings the 20th International Conference on Computational Linguistics (COLING'04), Geneva, Switzerland, 2004.

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

[43] M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, 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), Siguenza, Spain, 2002.

[44] M. Vickers, Ontology-based free-form query processing for the semantic web, Master's thesis, Brigham Young University, Provo, Utah (June 2006).

[45] Y. Wang, 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), Honolulu, Hawaii, 2002.

[46] Y. Wang, J. Völker, P. Haase, Towards semi-automatic ontology building supported by large-scale knowledge acquisition, in: AAAI Fall Symposium On Semantic Web for Collaborative Knowledge Acquisition, vol. FS-06-06, Arlington, Virginia, 2006.

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

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

[49] Y. Zhai, B. Liu, Web data extraction based on partial tree alignment, in: Proceedings of the 14th International Conference on World Wide Web (WWW'05), Chiba, Japan, 2005.