PROJECT DESCRIPTION

1 Introduction

With the ever-growing volume of web content, users have become overwhelmed with searching for information and finding and using services. The semantic web and personal software agents purport to offer a solution to this challenge [BLHL01]. But exactly how this solution will play out is still unclear.

It is clear, however, that ordinary users will not have the sophistication to use the semantic web if they have to learn query languages or specialized service protocols. Ordinary users need a simple interface, one with essentially no restrictions. Natural language—more precisely, degenerate natural language with no restriction on proper or complete sentence construction—is likely to be a key way to simplify interaction for ordinary semantic web users.

To illustrate what we mean and to clarify our intentions, consider the following examples.

Example 1. Figure 1 shows two ordinary, human-readable web pages for selling cars. The system we are proposing can annotate these pages automatically with respect to a given ontology about car advertisements and thus can convert them to semantic web pages in machine-readable form. Furthermore, the system we are proposing can answer ordinary, natural-language text queries over semantic web pages. Thus, assuming our annotation system had already created semantic web pages for Figure 1, the query

> Find me a red Nissan for under $5000 – it should be a 1990 or newer and have less than 120K miles on it.

would yield the results in Figure 2. These results are actual answers to the query in a table whose header attributes are the concept names from the given car-ads ontology, restricted to those concepts mentioned in the query. In addition, there is always one additional attribute, *Source*, whose entries are links back into the original document. When a user clicks on *Car13*, for example, the Athens document in Figure 1 appears, except it would be scrolled to the right place and the information requested in the query would be highlighted.

Example 2. The system we are proposing can also provide certain types of services. Using our proposed system, the request

> Please schedule me a visit to see a dermatologist next week; any day would be OK for me, at 4:00. The dermatologist should be within 5 miles from my home and must accept my health plan.

would be answered by the illustrative interaction in Figure 3.[1]

In order to succeed, we must accomplish three objectives. We must (1) be able to automatically annotate web pages with respect to an ontology, (2) provide a way for the system to answer free-form[2] queries, and (3) provide a way for the system to satisfy free-form service requests. We base our approach to all these problems on information-extraction ontologies [ECJ+99], and we explain in the remainder of this proposal the details about how we intend to accomplish these objectives.

[1] Although not yet worked out, we intend to add a generous sprinkling of syntactic sugar to these interactions so that they become much more palatable to the ordinary user. The interaction in Figure 3 is in raw terms, based on our current extraction ontologies.

[2] The modifier *free-form* is our term for "possibly degenerate, natural language."

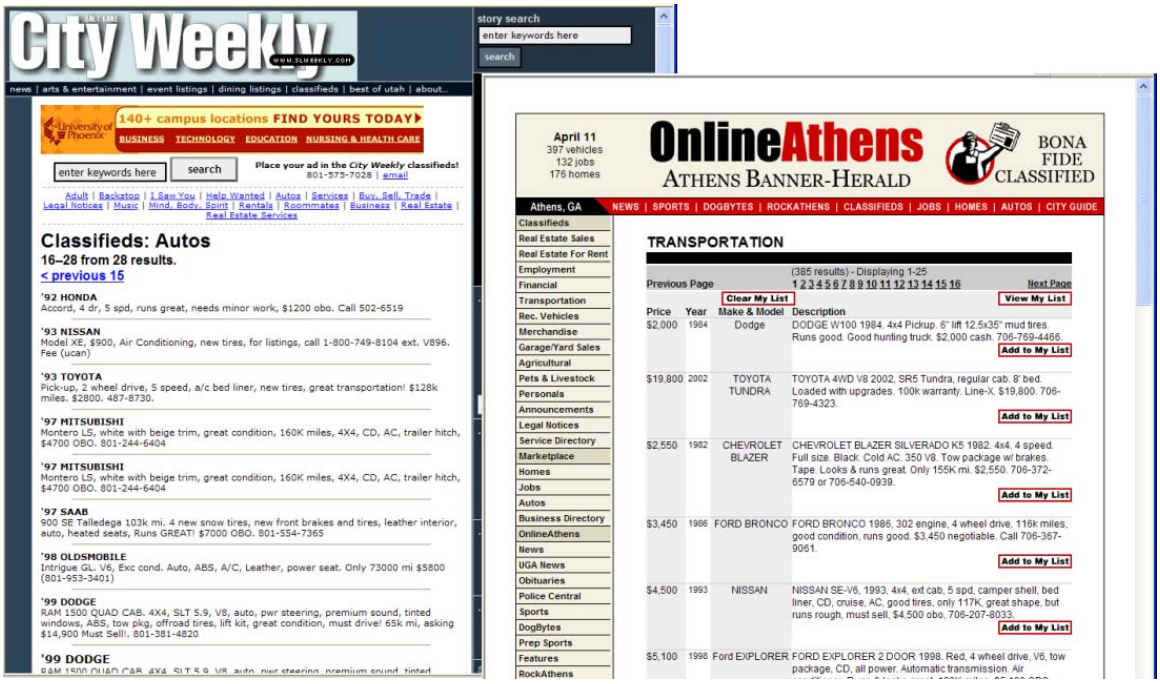

Figure 1: Sample Car Ad Web Pages from Salt Lake City Weekly and Athens Banner-Herald Sites.

For your query:

Find me a red Nissan for under $5000 – it should be a 1990 or newer and have less than 120K miles on it.

the system found the following results:

Color	Make	Price	Year	Mileage	Source
	NISSAN	$900	'93		Car1
....					
	NISSAN	$4,500	1993	117,000	Car13
....					

Figure 2: Query Results.

For your service request:

Please schedule me a visit to see a dermatologist next week; any day would be OK for me, at 4:00. The dermatologist should be within 5 miles from my home and must accept my health plan.

The following constraint(s) are not satisfied:

LessThanOrEqual(DistanceBetween("600 State St., Orem", "300 State St., Provo"), "5")
where DistanceBetween("600 State St., Orem", "300 State St., Provo") = 6

What do you wish to do?

user input> 6 miles is fine with me - schedule the appointment

Appointment is for Person - has Name(Lynn Jones).
Appointment is on Date(5 Jan 05).
Appointment is at Time(4:00).
Appointment is with Dermatologist - has Name(Dr. Carter); accepts Insurance(IHC);
is at Address(600 State St., Orem).

Figure 3: Service Request Results.

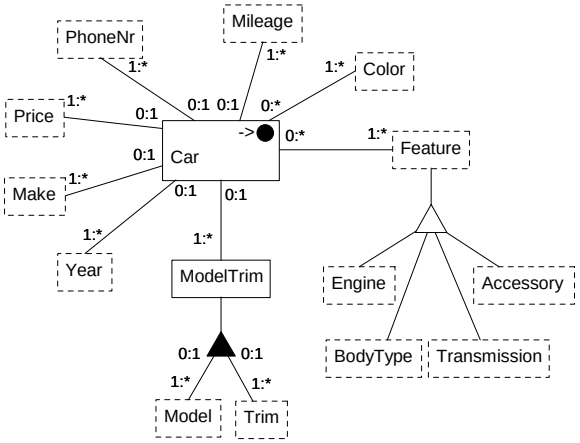

Figure 4: Graphical Component of an Extraction Ontology.

2 Information-Extraction Ontologies

The work proposed here builds naturally on our results from prior NSF support[3] and shows that the work accomplished has established a firm foundation for a potentially significant contribution to the semantic web. We have described information-extraction (IE) ontologies elsewhere [ECJ+99, Emb04], but to make our proposal self-contained and to report on our results of prior NSF support, we briefly reintroduce them here. We have used IE ontologies in a number of applications, including information extraction itself [ECLS98, ECJ+98, ECJ+99, TE02, WE04], high-precision classification [ENX01], schema mapping for ontology alignment [BE03, EJX01, EJX02, EX04, EXD04, XE03a, XE03b, XE], agent communication [TAME04], and record linkage [AKE04]. For this proposal we intend to show how to use IE ontologies for semantic web annotation (Section 3), semantic web queries (Section 4), and semantic web services (Section 5).

An *extraction ontology* specifies named sets of objects, which we call *object sets* or *concepts*, and named sets of relationships among object sets, which we call *relationship sets*. Figure 4 shows a graphical rendition of an extraction ontology for car advertisements. The extraction ontology has two types of concepts: lexical concepts (enclosed in dashed rectangles) and nonlexical concepts (enclosed in solid rectangles). A concept is *lexical* if its instances are indistinguishable from their representations. *Mileage* is an example of a lexical concept because its instances (e.g. "117K" and "5,700") represent themselves. A concept is *nonlexical* if its instances are object identifiers, which represent real-world objects. *Car* is an example of a nonlexical concept because its instances are identifiers such as, say, "Car13", which denotes a particular car in the real world. An extraction ontology also provides for explicit concept instances (represented as large black dots). We designate the main concept in an extraction ontology by marking it with "->●" in the upper right corner, which indicates that the object set *Car* becomes ("->") an object instance ("●") for a single car ad.

Figure 4 also shows relationship sets among concepts, represented by connecting lines, such as the connecting line between *Car* and *Year*. The numbers near the connections between relationship sets and object sets are participation constraints. Participation constraints give the minimum and maximum participation of an object in an object set with respect to the connected relationship set. For example, the *0:1* participation constraint on *Car* in the *Car-Mileage* relationship set denotes that a car need not have a mileage in a car ad, but if it does, it has only one. A white

[3]IIS-0083127: "Target-Based Document-Independent Information Extraction," $428,663, 1 April 2001 to 31 March 2005.

```
Price
   internal representation: Real
   external representation: \$?(\d+ | \d?\d?\d,\d\d\d)
   context keywords: price | asking | obo | neg(\.|otiable) | ...
   ...
   LessThan(p1: Price, p2: Price) returns (Boolean)
   context keywords: less than | < | or less | fewer | ...
   ...
end

Make
   external representation: CarMake.lexicon
   ...
end
```

Figure 5: Sample Data Frames for Car Ads Ontology.

triangle defines a generalization/specialization with the generalization connected to the apex of the triangle and one or more specializations connected to its base. In Figure 4, for example, *Feature* is a generalization of *Engine* and *BodyType*, among others. The white triangle can, of course, appear repeatedly, and thus we can have large *ISA* hierarchies in an extraction ontology. A black triangle defines an aggregation with the aggregation connected to the apex of the triangle and the component parts connected to its base. In Figure 4, for example, *ModelTrim* is an aggregation of the the *Model* and the *Trim*. As is the case for *ISA* hierarchies, large aggregation hierarchies are also possible.

As a key feature of extraction ontologies, each concept has an associated data frame [Emb80]. A *data frame* describes information about a concept—its external and internal representations, its contextual keywords or phrases that may indicate the presence of an instance of the concept, operations that convert between internal and external representations, and other manipulation operations that can apply to instances of the concept along with contextual keywords that indicate the applicability of an operation. Figure 5 shows sample (partial) data frames for the concepts *Price* and *Make* in our ontology for car advertisements. As Figure 5 shows, we use regular expressions to capture external representations. The *Price* data frame, for example, captures instances of this concept such as "$4500" and "17,900". A data frame's context keywords are also regular expressions (often simple lists of keywords). The *Price* data frame in Figure 5, for example, includes context keywords such as "asking" and "negotiable". In the context of one of these keywords, if a number appears, it is likely that this number is a price. The operations of a data frame can manipulate a concept's instances. For example, the *Price* data frame includes the operation *LessThan* that takes two instances of *Price* and returns a *Boolean*. The context keywords of an operation indicate an operation's applicability; context keywords such as "less than" and "<", for example, apply to the *LessThan* operation. Sometimes external representations are best described by lexicons or other reference sets. These lexicons or reference sets are also regular expressions, often simple lists of possible external representations, and can be used in place of or in combination with regular expressions. In Figure 5, *CarMake.lexicon* is a lexicon of car makes, which would include, for example, "Toyota", "Honda", and "Nissan" and potentially also misspellings (e.g. "Volkswagon") and abbreviations (e.g. "Chev" and "Chevy").

We can apply an extraction ontology to obtain a structured representation of the unstructured information in a document. For example, given the car-ads extraction ontology and the ad

'93 NISSAN Model XE, $900, Air Conditioning, new tires, call 749-8104

we can extract "**'93**" as the *Year*, "**NISSAN**" as the *Make*, "XE" as the *Trim* (even though it is labeled as the model), "$900" as the *Price*, both "Air Conditioning" and "new tires" as *Feature*s

with "Air Conditioning" also being an *Accessory*, and "749-8104" as the *PhoneNr*. As part of the extraction, the conversion routines in the data frames convert these extracted values to a canonical form; "**'93**", for example, becomes the integer *1993*. Although often simple, as illustrated here, there are sometimes subtle problems that can make extraction ontologies fail to extract correctly. Ambiguities can occur; for example, when there are two prices (e.g., list price and selling price). In this case, we use heuristic rules to sort out the ambiguities [ECJ+99]. We assume that we can correctly identify each individual record in a document. This requires work on record separation [EJN99, WE04], and sometimes requires the system to recognize that it must distribute factored information, such as a dealer telephone number that applies to many records, or that it must split records, such as when multiple cars sold by one dealer appear as a single ad, or that it must access off-page information under a link [EX00]. We assume also that the data is accessible, but often it is behind forms, in which case the system must first fill in the form and obtain the information from the hidden web [ELY01, LDEY02, CEL04]. Further, the hidden web and sometimes the visible web present data in a relatively structured form, such as a table. Surprisingly, structure causes more difficulties for extraction ontologies than might be expected, and special techniques are needed to handle structured information [ETL02, ETL].

3 IE-Based Semantic Annotation

Having explained what an extraction ontology is, we now discuss how to apply extraction ontologies for annotating pages for the semantic web. A typical semantic annotation process includes three components. First, an ontology must be created to describe the domain of interest. Second, a data instance recognition process discovers instances of interest in target web documents based on the defined ontology. Third, an annotation generation process creates a semantic meaning disclosure file for each annotated document. Through the semantic meaning disclosure file, any ontology-aware machine agent can understand the target document.

For ontology creation, we assume either that an extraction ontology is hand crafted or semiautomatically generated. Using tools we have developed [LHE03, WLE05] for hand-crafting extraction ontologies of the size needed for car ads, we have found that a few dozen person hours is sufficient for building extraction ontologies that can extract with precision and recall both nearing 90% [ECJ+99]. Others involved in semantic annotation research [DEG+03, HSC02, KPT+04, VVMD+02] also assume that ontologies already exist. Although [SBA+02] reports having tried to automatically generate ontologies for use in semantic annotation, [SR03] reports problems with this approach, particularly the "concept disambiguation problem." Generally, the problem of automatic ontology generation is considered to be difficult and not currently viable [DF02], but research continues, including our own [TEL+], and we certainly welcome a more automated way to generate ontologies.

For data instance recognition, manual recognition and annotation aided by tools is possible (e.g. [Hol96, RMW97, Cri, GSM99, OAM99, HLO99, KKPS01, Cri, Thi]). Although useful for small numbers of pages, it is very unlikely that we can use manual semantic annotation tools to convert the current web to be the semantic web. Automated semantic annotation tools are a necessity.

To develop automated annotation tools, researchers have turned to information-extraction (IE) tools [LRNdST02]. *HTML-aware IE tools* extract data of interest based on pre-defined HTML layout descriptions. [ACMM03] describes an attempt to do semantic annotation with the HTML-aware RoadRunner tool [CMM01], but RoadRunner only finds the location of data of interest, not the semantic meaning of the data with respect to an ontology. This necessitates manual labeling or aligning the data extracted with an ontology, a task which the authors of [ACMM03] point out is not easy. *NLP-based IE tools* analyze text using NLP (natural language processing)

techniques. There are several current efforts to adapt these IE tools for semantic annotation [HSC02, VVMD+02, KPT+04]. Unfortunately, as currently constructed, these tools also suffer from the need to align extracted information with whatever domain ontology they use for semantic annotation. [KPT+04] explains, saying, "The main drawback ... is that none of these approaches expects an input or produces output with respect to ontologies. [Thus,] ... a set of heuristics for post-processing and mapping of the IE results to an ontology ... [is needed]." Furthermore, NLP-based IE tools only work well with sentential text; they do not work well with telegraphic text or with structured or semi-structured data, which constitutes much of the web. *ILP-based tools* learn the formatting features that implicitly delineate the structure of data found in a page. [DEG+03] represents one attempt to use this type of IE tool for semantic annotation. Similar to HTML-aware and NLP-based IE tools, an ontology is not part of the input, and thus ILP-based IE tools have the same postprocessing alignment issues. Furthermore, machine learning processes usually need a large set of training documents, which must be labeled, usually manually. *Ontology-based IE tools* (e.g. [AKM+03, DYKR00, ECJ+99, MNS02]) apply pre-defined data-extraction ontologies to perform data extraction. Since this type of tool extracts information with respect to an ontology, no separate alignment of concepts is necessary. We argue that these types of tools are likely to be the best for developing automated semantic web annotation tools.

Previous to the work proposed here, no attempt has been made to use ontology-based IE tools for semantic annotation. Besides the benefit of eliminating the additional alignment between concepts in domain ontologies and extraction categories in IE engines, ontology-based IE tools also have the benefit of being resilient to the layouts of web pages and immediately work with new web pages in the same domain. These features allow this technique to scale up because extraction ontologies neither need to be rewritten nor regenerated for pages that change and for new pages within the domain that come online.

Turning our attention to the annotation itself and semantic meaning disclosure files, we ask, "How should we record and store annotations for the semantic web?" Although it is likely to be straightforward to adapt our work proposed here to any set of guidelines provided by the semantic web community, we know of no current, agreed-upon guidelines. Generally speaking, we can see two ways to do annotation: *explicit annotation*, which adds special tags that bind tagged instances in a web page to an externally specified ontology, and *implicit annotation*, which adds nothing explicit to the document, but instead extracts instance position information as well as the data instances and stores them in an externally specified ontology. As examples, [HH00] shows how to do explicit annotation (most others have followed this lead), and [HNS+04] shows how to do implicit annotation. We have explored both forms of annotation.

Using explicit annotation, we have created an online demo [DEGb] of our semantic annotation tool.[4] Our demo shows that it has extracted specific information from a web site by adding the hover feature of HTML/CSS to each instance so that a user can drag a mouse over instances and see them highlighted. The hover feature is only for the demo. For the annotation itself, we include a four-tuple in a *span* tag for every recognized data instance x. This four-tuple uniquely identifies (1) the ontology used for annotation (in case there are several for the same document), (2) the concept within the ontology to which x belongs, (3) the record number for x so that the system knows which values relate together to form a record, and (4) a value number within the record in case more than one instance of the concept can appear within a record. Thus, for example, we annotate the value *117K* in Figure 1 by 117K. Here *CarAds* is the ontology, *Mileage* is the concept, *13* is the record number, and *0* is the value

[4] As the last of the products produced for our prior NSF grant, this online demo ties our previous work to the work we are proposing here.

```
<?xml version="1.0"?>
<ref:RDF
....
    xmlns:webpage="http://www.deg.byu.edu/demos/..."
....
    <owl:Class rdf:ID="CarAds">
....
        <Mileage rdf:ID="MileageIns13">
        <MileageValue rdf:datatype="xs:string">117K</MileageValue>
        <MileageCanonicalValue rdf:datatype="xs:nonNegativeInteger">
            117000</MileageCanonicalValue>
        <offset rdf:datatype="xs:nonNegativeInteger">37733</offset>
    </Mileage>
....
    <owl:Thing rdf:about="#CarAdsIns13">
        <hasPrice rdf:resource="#PriceIns13" />
        <hasYear rdf:resource="#YearIns13" />
        <hasMake rdf:resource="#MakeIns13" />
....
    </owl:Thing>
....
</ref:RDF>
```

Figure 6: Implicit Annotation for Car Ads Web Page.

number. In addition to the *span* tags surrounding all data instances of interest to an ontology, we also add a *meta* tag in the header that links an ontology name, *CarAds* for our example, with its URL. The URL specifying the OWL [W3C] ontology along with *span* annotations allow the system to create the equivalent of a populated semantic ontology for each annotated page.

For our implicit annotations, we directly augment OWL ontologies with instances. Figure 6 shows a portion of an implicit annotation for the web page in Figure 1. When we do implicit annotation, we also cache a copy of the original web page so that we can guarantee that the instance position information is correct. Figure 6 shows that for our implementation we have cached the web page on our web site *www.deg.byu.edu*. Following the URL in Figure 6, we show the beginning line of our *CarAds* ontology. Following the ontology definition, we add the instances. Figure 6 shows the mileage instance, *117K*, its canonical integer representation, *117000*, and its character offset in the cached web page, *37733*. Observe that we give each instance a unique identifier, *MileageIns13* for the *117K* in our example. We then collect all the instances together as a record. The OWL *Thing* in Figure 6 is the record about *CarAdsIns13*, which includes the unique identifiers of its price, year, make, etc. This semantic meaning disclosure file fully annotates the web page in Figure 1.

4 IE-Based Semantic Web Queries

The authors of [BKF+05b] discuss principles and desirable features for web query languages. In their "Verbalizing" principle, they advocate "some form of controlled natural language processing" and point out that the success of web search engines demonstrates the importance of "a seemingly free-form, 'natural' interface." We agree, and we further advocate an even stronger position. For many ordinary users of the semantic web, we believe that queries should be totally free-form, natural-language text, with no restrictions. The problem becomes how to interpret the query. It is easy to suppose that natural language processing (NLP) can solve this problem, but pure NLP techniques have not proven to be very successful for question/answer systems (as witnessed by the flurry of early NLP work for database querying, which has since died out). Furthermore, free-form natural text may be telegraphic (i.e. short, sometimes non-grammatical, and often with incomplete phrases), on which NLP techniques typically do not work well. [BKF05a] suggests the use of

controlled natural language. Perhaps this may be necessary and potentially could be successful [Kan]. Users, however, must learn and adjust to artificial requirements imposed over the lexis, syntax, and semantics of the language. Many ordinary users may neither have the patience to learn the nuances of the language nor the skill necessary to make the required adjustments.

In light of these requirements (the need for free-form, natural-language-like text) and difficulties (the problems of traditional NLP and controlled natural languages), we propose a different approach. Our approach may be characterized as an *information-extraction, ontology-based, natural-language-like approach*. The essence of the idea is to (1) extract constants, keywords, and keyword phrases in a natural-language or telegraphic query; (2) find the best-match ontology; and (3) embed the query in the ontology yielding (3a) a join over the relationship-set paths connecting identified concepts, (3b) a selection on identified constants modified by identified operators, and (3c) a projection on mentioned concepts. Our expectation for success is based on arguments in [CSvM03] suggesting the possibility of using ontologies to help build better question/answer systems and on our experience long ago with attempting to do NLP over conceptual models [EK85].

Consider our running example, where the user specifies: "Find me a red Nissan for under $5000 – it should be a 1990 or newer and have less than 120K miles on it." The best-matching extraction ontology from our library is the car-ads ontology. When we apply our car-ads extraction ontology to this sentence, we discover that the desired object has restrictions on five concepts: color, make, price, year, and mileage. For string-valued concepts (color and make), we can test equality (either equal or not equal). Since there are no keyword phrases in the query that indicate negation, in this case we search for objects where *Color=red* and *Make=Nissan*. For numeric concepts (price, year, and mileage), we can test ranges. Associated with each operator in a data frame are keywords or phrases that indicate when the operator applies. In this case, "under" indicates a less-than comparison, "or newer" indicates $\geq$, and "less than" indicates $<$. So in our example, we must search for $Price < 5000$, $Year \geq 1990$, and $Mileage < 120000$. Recall, from our discussion in Section 2, that our data frames specify operators that convert a string to a canonical internal representation. Thus, for example, *"120K"* becomes *120000* when we invoke the canonicalization operator. The result of this extraction over the user-specified query is a set of filters of the form $\langle$*concept name*, *operator*, *value*, *optionality*$\rangle$ (*optionality* indicates whether a value may or must appear according to the participation constraints in the extraction ontology). In the search process, we look for objects that satisfy all the filter expressions. The particular concept filters for our example include: $\langle$*Color*, $=$, *red*, *true*$\rangle$, $\langle$*Make*, $=$, *Nissan*, *true*$\rangle$, $\langle$*Price*, $<$, *5000*, *true*$\rangle$, $\langle$*Year*, $\geq$, *1990*, *true*$\rangle$, $\langle$*Mileage*, $<$, *120000*, *true*$\rangle$.

Given a set of concept filters, we can readily search a web page by generating an XQuery (e.g. Figure 7) over an external semantic meaning disclosure file (e.g. Figure 6). Each filter expression generates a *let* clause to look up the corresponding extracted value, a phrase within the *where* clause to test the given condition, and an element in the *return* clause to generate XML that contains the extracted value. In our running example, all the concepts happen to be optional; for required concepts we drop the "**or** empty(...)" phrase. To perform semantic web searches, we apply this query to all documents that are applicable to the given domain, collect the results, and display them to the user in tabular format as Figure 2 shows.

Note that optional elements might not be present in some of the records, and thus—as is the case with the ordinary web—our semantic web queries may return irrelevant results. For example, suppose a car ad does not list the car's color, but otherwise it satisfies the user's constraints. Rather than miss a potential object of interest, we allow optional elements to be missing and we return the partial record with the query results. It would not be hard to allow users to override this behavior and require the presence of all concepts in each of the query results (imagine a checkbox that allows the user to require all filters to match). Another aspect not demonstrated by the running example

```
for $doc in document(URL)/rdf:RDF return
<QueryResult> {
for $Record in $doc/owl:Thing
let $id := substring-after(xs:string($Record/@rdf:about), "CarAdsIns")
let $Color := $doc/carads:Color[@rdf:ID=concat("ColorIns", $id)]/carads:ColorValue/text()
let $Make := $doc/carads:Make[@rdf:ID=concat("MakeIns", $id)]/carads:MakeValue/text()
let $Price := $doc/carads:Price[@rdf:ID=concat("PriceIns", $id)]/carads:PriceValue/text()
let $Year := $doc/carads:Year[@rdf:ID=concat("YearIns", $id)]/carads:YearValue/text()
let $Mileage := $doc/carads:Mileage[@rdf:ID=concat("MileageIns", $id)]/carads:MileageValue/text()
where   ($Color = "red" or empty($Color)) and ($Make = "Nissan" or empty($Make)) and
        ($Price < 5000 or empty($Price)) and ($Year >= 1990 or empty($Year)) and
        ($Mileage < 120000 or empty($Mileage))
return  <Record ID="{$id}"> <Color>{$Color}</Color><Make>{$Make}</Make>
        <Price>{$Price}</Price><Year>{$Year}</Year><Mileage>{$Mileage}</Mileage> </Record>
} </QueryResult>
```

Figure 7: XQuery to Search an Annotated Web Page.

is what happens when a concept is mentioned in the query but there is no operator indicated for comparison. For example, suppose the user added "I'm okay with any body style" to the query. In this case, "body" is a keyword for the *BodyType* concept in our ontology. Since a relevant keyword is present, the system extracts a filter expression $\langle$*BodyType*, $\emptyset$, $\emptyset$, *true*$\rangle$. The generated XQuery then includes a *let* clause and output tags in the *return* clause for *BodyStyle*, similar to those for the other concepts shown in Figure 7. Nothing is added to the *where* clause because no additional selection is appropriate.

5 IE-Based Semantic Web Services

The work we are proposing to do for semantic web services is unique. We know of no research trying to satisfy free-form user service requests by establishing needed relationships within the context of the constraints imposed by an ontology. Thus, rather than competing with the current work on web services, we offer a unique, complementary approach. We are not developing a traditional service-oriented architecture [GBR05] in which business functionality is packaged as reusable services that can be invoked through standard interfaces. Neither are we developing infrastructure competing with W3C's Web Services Description Language (WSDL) [WSD] nor its surrounding constellation of related standards including the XML Protocol (which replaces SOAP) [XML], the OASIS UDDI service registry mechanism [UDD], and other proposals within W3C's Web Services Activity [Web05]. And we are not proposing additional architectural support (e.g. WSMF for addressing decoupling and mediation needs [FvH02]; the Internet Reasoning Service for automatically publishing, locating, composing, and executing heterogeneous services [MDCG03]; BPEL4WS for providing higher levels of web-services "choreography" [ACD+03]; and workflow for semantic web services [MSZ01] that typically follow the DAML-S/OWL-S line of work [FvH02, KKOF02, KB04, OWL04]). Instead, we want to show that for a specific type of web service—namely one in which the service request can be satisfied by establishing a single new relationship among concepts in an ontology—we can leverage the principles of ontology-based data extraction to hide complexity from the end user.

We show by an example why we believe that it is possible to build a system to automate everyday tasks (such as scheduling appointments, buying and selling, and making job assignments) whose invocation results in establishing agreed-upon relationships in an ontology. We assume in our example that a user makes the service request in Example 2 in Section 1. Before a user states a service request, our proposed system knows nothing regarding the domain of the task nor anything about what is to be done within the domain of the task. Therefore, this specification first

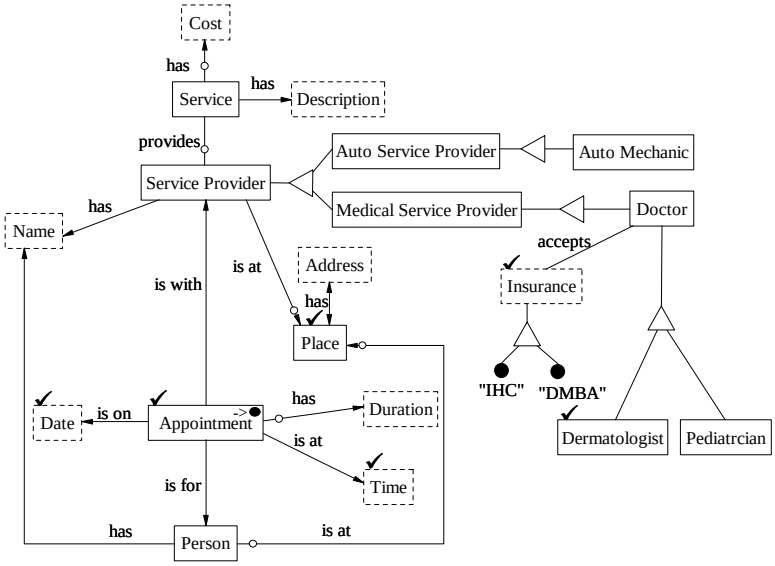

Figure 8: The Output of the Recognition Process.

goes through a domain ontology recognition step and then goes through a (possibly interactive) execution step. In the following paragraphs, we explain these two steps. An important feature of both these steps is that the code to execute them is fully specified once the extraction ontology for the domain is chosen. Code is either automatically specialized from code written for *all* domain ontologies or is automatically generated given the chosen domain ontology.

The domain ontology recognition process selects the (correct) domain ontology for a user's request from among potentially many domain ontologies deployed on the semantic web. The recognition process takes the set of available domain ontologies and a user's request as input, and returns the domain ontology that best matches with the user's request as output. The recognition process works in two steps. First, for each domain ontology the recognition process applies concept recognizers in the data frames to the user's request and marks every concept that matches a substring in the user's request. Second, the process computes a rank value for each domain ontology with respect to the user's request and then selects the domain ontology with the highest rank value.

Referring to our example, given that a domain ontology for appointments exists and given the user's request in Example 2, the recognition process produces the output in Figure 8. We assume here that the concept recognizer in the data frame for *Dermatologist* recognizes the string "dermatologist" in the user's request, and therefore the concept *Dermatologist* is marked ($\surd$). Likewise we assume that recognizers in the *Date*, *Time*, *Appointment*, *Place*, and *Insurance* data frames respectively recognize the phrases, "next week", "4:00", "schedule ... visit", "my home", and "health plan"; and therefore these concepts are marked. The recognized substrings cover a large part of the user's request; for our example, we assume that no other ontology covers the user's request as well and therefore that the system selects the *Appointment* task ontology.[5]

We have defined a process ontology that is responsible for executing tasks (for details see [AM05]). Significantly, and as already mentioned, the process ontology invokes subprocesses that either execute the same way for all domain ontologies or can be automatically generated from any

[5]It is interesting to think about possible errors in the selection process. The user's request should establish enough context for the system to select the right ontology. If the system selects the wrong task ontology, it will respond in terms foreign to what the user expects, just like humans sometimes do. We can correct the system by providing more or better context. As another possibility, the system may have no task ontology for the service being requested; the system should be able to recognize this possibility and respond by saying it cannot provide the service.

$$
\begin{aligned}
&Appointment(x_0)\ is\ with\ Dermatologist(x_1) \wedge Appointment(x_0)\ is\ for\ Person(x_2)\\
&\wedge\ Appointment(x_0)\ is\ on\ Date(x_3) \wedge Appointment(x_0)\ is\ at\ Time(\text{“4:00”})\\
&\wedge\ Dermatologist(x_1)\ has\ Name(x_4) \wedge Dermatologist(x_1)\ is\ at\ Address(x_5)\\
&\wedge\ Dermatologist(x_1)\ accepts\ Insurance(x) \wedge Person(x_2)\ has\ Name(x_7)\\
&\wedge\ Person(x_2)\ is\ at\ Address(x_8)\\
&\wedge\ ...\\
&\wedge\ NextWeek(x_3) \wedge LessThanOrEqual(DistanceBetween(x_5, x_8),\ \text{“5”})\\
&\wedge\ ...
\end{aligned}
$$

Figure 9: Generated Predicate Calculus Statement.

given domain ontology. We now briefly describe these subprocesses.

- *Create Task View.* Task view creation takes a marked domain ontology as input and produces a task view as output. A task view restricts the domain ontology to just those concepts needed to satisfy the user's request. Although not quite so simple because spurious object sets may be marked, the process basically operates by keeping the main concept of the domain ontology (the concept marked with "–>●"), all marked concepts, and all concepts that mandatorily depend on the main concept, while pruning away all other concepts along with all their relationships and any marked concepts considered to be spurious because they conflict with other marked concepts (in the sense that ontology constraints do not allow both). In addition, the process replaces generalization concepts by marked specializations (e.g replaces *Service Provider* by *Dermatologist* in Figure 8), and replaces nonlexical concepts by lexical concepts when there is a one-to-one correspondence (e.g. replaces *Place* by *Address* in Figure 8).

- *Augment Task View with User-Requested Constraints.* Given the task view and the operations in the data frames associated with the concepts of a task view, the system generates any additional constraints imposed on a task beyond those that are already part of the conceptual-modeling constraints of the task view. It then combines them with the constraints in the task view to produce the full set of constraints. Figure 9 shows part of the resulting predicate calculus statement that must be satisfied in order to service the request.[6] We obtain this expression by converting our extraction ontology into predicate calculus as explained in [EKW92] and adding user-request constraints in a four-step process: (1) *Get the Boolean operations implied by the user's request* (e.g. in our example, the process finds the operator *NextWeek(d: Date)* because it is Boolean and one of its context phrases is "next week"). (2) *Get constant values from the user's request that can serve as parameters of the Boolean operations* (e.g. the data frames recognize and extract "5" as a *Distance* and "4:00" as a *Time*, and thus obtain *LessThanOrEqual(d1: Distance, "5")* and *Time("4:00")*.[7] (3) *Get operations that depend on the task view and can provide values for parameters of the Boolean operations* (e.g. by analysis and substitution, we obtain *LessThanOrEqual(DistanceBetween(a1: Address, a2: Address), "5")*). (4) *Get sources, within the task view, for values of Boolean operations* (e.g. because *Insurance* is related only to *Dermatologist*, the process determines that the source of the insurance value comes from the relationship *Dermatologist accepts Insurance*).

[6]Note that, with respect to description logics [BN03], this expression can be converted into the language *ALCN* and its satisfiability is thus decidable and its complexity is *ExpTime-complete*. With regard to complexity, we show next that for our special case, we can reduce the execution to a straightforward select-project-join query over a standard relational database to obtain the result we need. We conjecture, and will prove as part of our proposed work, that all predicated-calculus expressions generated from any domain ontology by the process we have described here will be decidable and, for the results we need, will reduce to straightforward relational-database queries.

[7]Note that *Time(x)* is a one-place predicate. Every object set in a domain ontology is a one-place predicate, and every n-ary relationship set is an n-place predicate.

$$\{< x_1, x_3, x_4, x_5, x > |$$
$$\quad Appointment\ is\ with\ Dermatologist(x_1)\ and\ is\ on\ Date(x_3)\ and\ is\ at\ Time(\text{“}4{:}00\text{”})$$
$$\quad \wedge\ Dermatologist(x_1)\ has\ Name(x_4) \wedge Dermatologist(x_1)\ is\ at\ Address(x_5)$$
$$\quad \wedge\ Dermatologist(x_1)\ accepts\ Insurance(x)\}$$

Figure 10: Generated Predicate Calculus Query.

- *Obtain Information from the System.* Our objective, as the process continues, is to provide values for free variables such that there is one and only one appointment (i.e. one and only one value for the nonlexical argument x_0 in Figure 9). Given the predicate calculus statement in Figure 9, the system can generate a query for the system's appointment databases.[8] Assuming that the appointment database has a view definition that corresponds with its ontology, the generation of a relational calculus query is straightforward. The process simply cuts the predicate calculus statement down to include only those relationship sets that appear in the database's ontological view, and in one case, namely for the object set of interest, it combines the relationship sets from the database's ontological view that are connected to the object set of interest. For our example, the generated predicate calculus query is in Figure 10.

- *Obtain Information from the User.* Execution of the generated query returns a set of partially filled-in interpretations. For our example the partially filled-in interpretations are the set of possible appointments. At this point in our example, the remaining free variables are x_0, which represents the appointment we are trying to establish; x_2, which is the person for whom the appointment is being made; x_6, which is the insurance in the request; x_7, which is the name of the person for whom the appointment is being made; and x_8, which is the address of the person for whom the appointment is being made. We can establish the variable x_0, which represents the appointment, if we can obtain the remaining variables, which, of course, are exactly the ones we need to obtain from the user. To obtain a reasonable dialog with the user we base our interaction on verbalizations for conceptual models [Hal04, LEW00] and ask for the information we need.

- *Satisfy Constraints and Negotiate.* At this point in the process, the system has one free variable, namely the nonlexical object we are trying to establish (the *Appointment* for our example). When exactly one interpretation satisfies all the constraints, we are ready to establish the object and finalize the process. When no interpretation satisfies all the constraints, the system can try to negotiate with the user by relaxing one or more constraints (e.g. suppose the distance is within 6 miles instead of 5 miles). When a few interpretations satisfy the constraints, the system can simply present the possibilities and let the user choose from among them. When many interpretations satisfy the constraints the system should try to rank them and present the top-k possibilities to the user.

- *Finalize by Generating Domain-Dependent Code.* At this point in the process, the system either has a single solution or has agreed with the user that there is no solution. Hence, it is straightforward to know what to do. What is interesting here is that although this code cannot be written in advance because it does depend on the domain ontology, it can be generated on-the-fly by code that can be written in advance.

[8] We assume that the system has databases that store real-world instances of concepts of all domain ontologies known to the system.

6 Experimental Evaluation

The success of our system will depend largely on its ability to successfully perform and integrate five types of processing to correctly retrieve all and only relevant information from web pages for a given query and to correctly service given user requests: (1) appropriately and completely annotate web pages with respect to ontological specifications; (2) correctly extract useful semantic content from user queries and requests; (3) correctly select the most relevant ontologies from user queries and requests; (4) correctly clarify/negotiate intent/meaning when resolving ambiguous free-form queries and unsatisfiable or multi-satisfiable requests; and (5) correctly produce answers to user queries and satisfy user service requests.

Evaluating item (1) involves metrics for scoring how good the match is between data-rich semantic content on a web page and our store of information-extraction ontologies. This evaluation will be similar to the evaluation work we have done using precision/recall measures for record boundary detection [EJN99, WE04], rating ontology matching success [ENX01], and scoring the correctness and completeness of ontology concept alignment [EJX02, XE].

Evaluating item (2) is an open-ended problem, though significant steps have been taken in the last few years towards formal assessment of semantic content extraction. For example, SENSEVAL-3 [SEN] and CoNLL [CON] have both recently involved semantic role labeling tasks, and a substantial literature on this topic has developed [PHW$^+$03, Lit04, HPW$^+$04]. We are familiar with the relevant formats, tools, and literature [Tus04] and will be able to show incremental and overall performance of the system in terms of precision/recall measures. Our system's performance in parsing user queries will thus follow accepted state-of-the-art conventions.

Evaluating the success of item (3) reduces to ascertaining the correctness, consistency, and completeness of our extraction ontologies with respect to conceptual information contained in user queries or requests. This is similar to step (1) in that we calculate conceptual coverage, but here we are examining service requests and user queries rather than web pages.

Evaluation of item (4) involves, at a minimum, evaluating the performance of word sense disambiguation (WSD); pushed further, it must also measure performance in disambiguating potentially problematic syntactic and semantic constructions. SENSEVAL [SEN] and several other machine learning conferences have addressed the issue of evaluating WSD performance using traditional precision/recall measures, and we will use the emerging annotation conventions, scoring tools, and evaluation literature in extending our work in this area. In particular, our processing of user queries and service requests will build on our prior work in resolving word sense [LR01], syntactic [ML04], and semantic [RL01] ambiguities using symbolic [Lon00], exemplar-based [Jon96, SLP02], and hybrid [LM04] architectures.[9] We have also worked with dialogue move engines [Ree02] and corpora,[10] particularly those related to user queries and content negotiation. We also intend to show how interaction is useful and effective by evaluating and comparing how well the system performs by sidestepping interactions and using default assumptions versus when it interacts with the user for meaning clarification/negotiation.

Item (5) is the aggregate of these four subprocesses. Thus, we will also evaluate in a holistic way the entire process by performing standard information retrieval query tests based on precision/recall measures. This could include direct comparison of results from traditional web search (similar to what has been done in standardized tasks such as MUC [MUC] and TREC [TRE]) versus semantic

[9] For resolving syntactic and semantic disambiguation, no such standardized evaluation tasks have yet emerged in the NLP field, though one co-PI has worked in interactive disambiguation since 1980, mostly in the context of machine translation systems [Lon85, Kan].

[10] BYU is a member of the Linguistic Data Consortium (www.ldc.upenn.edu) and has possession of, and extensive experience with, a wide variety of linguistic corpora encoded via various annotation schemes.

web-annotated pages on some predefined corpus. We will also use to the extent possible standard semantic web corpora such as ISI's RISE [RIS] corpus. We will also evaluate in a holistic way the entire process for servicing user requests by adapting traditional precision/recall measures to this new service-request context.

7 Research Plan

Year 1. Construct system infrastructure and gather/build data sets and corpora to be used in evaluation: (1) Improve and extend annotation system—build a data-frame library and a tool to semiautomatically assemble needed data frames for new extraction ontologies; add many more extraction ontologies for various applications; analyze and improve execution speed. (2) Develop the basic free-form query and service-request systems. (3) Adapt our existing dialogue engine [LBB+01] with expectation-driven specifications to be provided by our ontologies—this will allow for ontology-directed, targeted user interactions for meaning and negotiation. (4) Integrate the different parsing systems (data frames, shallow parsing, and syn/sem interpretation) and provide control for choosing/merging results from the various engines—this will involve adapting the parsing engine to accept free-form user queries. (5) Analyze existing web query collections[11] [SMSH00, SWJS01] in view of developing a web query parsing test corpus. (6) Begin development of a hand-annotated corpus of query examples with the relevant extracted information and service requests with relevant service results.

Year 2. (1) Pipeline the components developed in Year 1 together into a fully functional prototype. (2) Based on our first year's experience, perform system research and development to incrementally improve the system by gradually eliminating weak points. (3) Prepare data and procedures for evaluation experiments.

Year 3. (1) Conduct the evaluation experiments on the new data during the first half of the year. (2) During the last half of the year, prepare the results for publication in archival technical journals. (In every year, we also plan to disseminate intermediate results in appropriate conferences and workshops.)

Our fully developed infrastructure (including our data frame library, ontology editors, free-form query system, free-form service request system, and interaction and negotiation systems), plus our hand-annotated corpus of query examples and service-request examples along with our experimental results and all the raw web pages used in the evaluation will be made available to other researchers through our web sites.

8 Expected Significance

The intellectual merit and broader impact of the proposed work have the potential to make a significant difference in the success of the semantic web.

8.1 Intellectual Merit

Our proposed system makes three major contributions: (1) a mechanism for automatically generating semantic annotations for ordinary web pages based on an extraction ontology, (2) a means for querying these annotated web pages in a simple, free-form, natural-language manner, and (3) a means for satisfying free-form service requests. These contributions can help pave the way for transforming the existing web into the semantic web.

[11] We have access to queries from such sites as www.metaspy.com and the Excite/AskJeeves corpora [SG01].

Our system strongly relies on (1) *conceptual modeling*, which forms the basis for both domain ontologies and process ontologies, (2) *extraction ontologies*, which allows the system to obtain the information it needs to match user requests with an appropriate domain ontology, and (3) *domain-independent process ontologies* that can be automatically specialized for any given domain, which makes our approach work across domains without need for manual configuration. The system has a rigorous theoretical foundation and relies on existing standards where appropriate (e.g., XML, OWL, XQuery).

8.2 Broader Impact

We have developed this project with an eye toward broadening its impact on society. We are collaborating with students of a diverse background, publishing and presenting recent papers, actively mentoring students, incorporating our research into graduate classes, and pursuing a research agenda with significant potential to benefit society in general.

Multi-disciplinary Faculty. Our team of PI's/Co-PI's has complementary expertise in the underlying theory and technology supporting this project. We represent three different colleges and departments (Computer Science, eBusiness, and Linguistics), and we have collaborated in various combinations for many years. Our papers, presentations, and other artifacts appear on our web site [DEGa]. We continually publish our work in conferences, workshops, and scientific journals.

Diverse Student Collaboration. In our research group we collaborate with a diverse collection of students, currently including 4 PhD students (2 female), 4 masters students (1 female), and 1 undergraduate. Our students come from China (3), Syria (2), and the United States (4). The bulk of our request is to support this student team. To broaden our impact on students, we teach several graduate courses (in three different colleges) on web-based data extraction and integration (Embley), information architecture and web development (Liddle), and natural-language processing (Lonsdale).

Benefit to Society. The internet and the web have become an integral and essential part of modern life. Likewise, the vision for the semantic web is compelling and highly ambitious and really could benefit humankind in a significant way—if it could be developed. But there are significant challenges to overcome before we can deliver the promise of this vision. Ultimately, the semantic web will be populated directly with annotated information created purposefully by publishers. But we cannot rely solely on directly annotated sources because (1) the ordinary web is too large to engineer into an annotated, semantic web, and (2) even if we could annotate the entire web, some publishers will always create non-semantic pages because of the engineering effort required to create properly annotated pages.

Our proposal provides a practical way forward. With our method, ordinary web pages can become a part of the semantic web without the need for highly engineered, largely manual annotation processes. Search-engine providers can augment their lookup services with semantic-web queries, providing more-accurate, more-relevant answers to users. Finally, ordinary users will be able to invoke services without the need for custom-programmed, complex software agents.