Natural Language Annotations for the Semantic Web

Boris Katz[1], Jimmy Lin[1], and Dennis Quan[2]

[1] MIT Artificial Intelligence Laboratory
200 Technology Square
Cambridge, MA 02139
{boris,jimmylin}@ai.mit.edu

[2] IBM Internet Technology Division
1 Rogers Street
Cambridge, MA 02142
dennisq@us.ibm.com

Abstract. Because the ultimate purpose of the Semantic Web is to help users locate, organize, and process information, we strongly believe that it should be grounded in the information access method humans are most comfortable with—natural language. However, the Resource Description Framework (RDF), the foundation of the Semantic Web, was designed to be easily processed by computers, not humans. To render RDF friendlier to humans, we propose to augment it with natural language annotations, or metadata written in everyday language. We argue that natural language annotations are not only intuitive and effective, but can also accelerate the pace with which the Semantic Web is being adopted. We demonstrate the use of natural language annotations from within Haystack, an end user Semantic Web platform that also serves as a testbed for our ideas. In addition to a prototype Semantic Web question answering system, we describe other opportunities for marrying natural language and Semantic Web technology.

1 Introduction

The vision of the Semantic Web [2] is to convert information on Web sites into a more machine-friendly form, with the goal of making the Web more effective for its users. This vision grew out of the recognition that although a wealth of information readily exists today in electronic form, it cannot be easily processed by computers due to a lack of external semantics.

Fundamentally, we interpret Semantic Web research as an attempt to address the problem of information access: building programs that help users locate, collate, compare, and cross-reference content. As such, we strongly believe that the Semantic Web should be motivated by and grounded in the method of information access most comfortable to users—natural language. We believe that natural language is the best information access mechanism for humans; it is intuitive,

easy to use and rapidly deployable, and requires no specialized training. In our vision, the Semantic Web should be equally accessible by computers using specialized languages and interchange formats, and humans using natural language. The vision of being able to ask a computer "when was the president of Taiwan born?" or "what's the cheapest flight to the Bahamas this month?" and getting back "just the right information" is very appealing.

Because the first step to building the Semantic Web is to transform existing sources (stored as HTML pages, in legacy databases, etc.) into a machine-understandable form (i.e., RDF), it is sometimes at odds with a human-based natural language view of the world. Although the general framework of the Semantic Web includes provisions for natural language technology, the actual deployment of such technology remains largely unexplored.

Exactly what synergistic opportunities exist between natural language technology and the Semantic Web? State of the art natural language systems are capable of providing users intuitive access to a wealth of textual data using ordinary language. However, such systems are often hampered by the knowledge engineering bottleneck; knowledge bases are difficult, and often time consuming, to craft. This is where the Semantic Web comes in: Semantic Web research is concerned with constructing, integrating, packaging, and exporting segments of knowledge to be usable by the entire world. We believe that natural language technology can tap into this knowledge framework, and in return provide natural language information access for the Semantic Web.

To illustrate the potential opportunities that lie on the intersect between natural language and the Semantic Web, we describe a prototype question answering system capable of retrieving relevant information from a repository of RDF triples in response to user queries formulated in natural language. We draw our inspiration from two existing systems: START, the first question answering system available on the World Wide Web, and Haystack, an end user Semantic Web platform that aggregates all of a user's information into a unified repository.

2 The START Natural Language System

The use of metadata is a common technique for rendering information fragments more tenable to processing by computer systems. We believe that using natural language itself as metadata presents several advantages and opportunities: it preserves human readability and encourages non-expert users to engage in metadata creation. To this end, we have developed natural language annotations [7], which are machine-parsable sentences and phrases that describe the content of various information segments. These annotations serve as metadata that describe the kinds of questions a particular piece of knowledge is capable of answering. We have implemented natural language annotation technology in the START Natural Language System[3] [6, 7].

To illustrate how our system works, consider the following paragraph about Joseph Brodsky, which may contain images and other non-textual elements:

[3] `http://www.ai.mit.edu/projects/infolab/`

> "For an all-embracing authorship, imbued with clarity of thought and poetic intensity," Joseph Brodsky was awarded the 1987 Nobel Prize in Literature.

This paragraph may be annotated with the following English sentences and phrases:

> Joseph Brodsky was awarded the Nobel Prize for Literature in 1987.
> 1987 Nobel Prize for Literature

START parses these annotations and stores the parsed structures (called embedded ternary expressions [9, 6]) with pointers back to the original information segments. To answer a question, the user query is compared against the annotations stored in the knowledge base. Because this match occurs at the level of syntactic structures, linguistically sophisticated machinery such as synonymy/hyponymy, ontologies, and structural transformation rules are all brought to bear on the matching process. Linguistic techniques allow the system to achieve capabilities beyond simple keyword matching, for example, handling complex syntactic alternations involving verb arguments. If a match is found between ternary expressions derived from annotations and those derived from the query, the segment corresponding to the annotations is returned to the user as the answer. For example, the annotations above allow START to answer the following questions (see Figure 1):

> What prize did Brodsky receive in 1987?
> Who was awarded the Nobel Prize for Literature in 1987?
> Tell me about the winner of the 1987 Nobel Prize for Literature.
> To whom was the Nobel Prize for Literature given in 1987?

An important feature of natural language annotations is that any information segment can be annotated: not only text, but also images, multimedia, and even procedures!

Since it came online in December, 1993, START has engaged in millions of exchanges with hundreds of thousands of users all over the world, supplying them with useful knowledge. Currently, our system can answer millions of natural language questions about places (e.g., cities, countries, lakes, coordinates, weather, maps, demographics, political and economic systems), movies (e.g., titles, actors, directors), people (e.g., birthdates, biographies), dictionary definitions, and much, much more.

In order to give START uniform access to semistructured resources on the Web, we have created Omnibase [8], a virtual database system that integrates a multitude of Web sources under a single query interface. To actually answer user questions, the gap between natural language questions and structured Omnibase queries must be bridged. Natural language annotations serve as the enabling technology that allows the integration of START and Omnibase. Since annotations can describe arbitrary fragments of knowledge, there is no reason why they cannot be employed to describe Omnibase queries. In fact, annotations can be parameterized, i.e., they can contain symbols representative of an entire

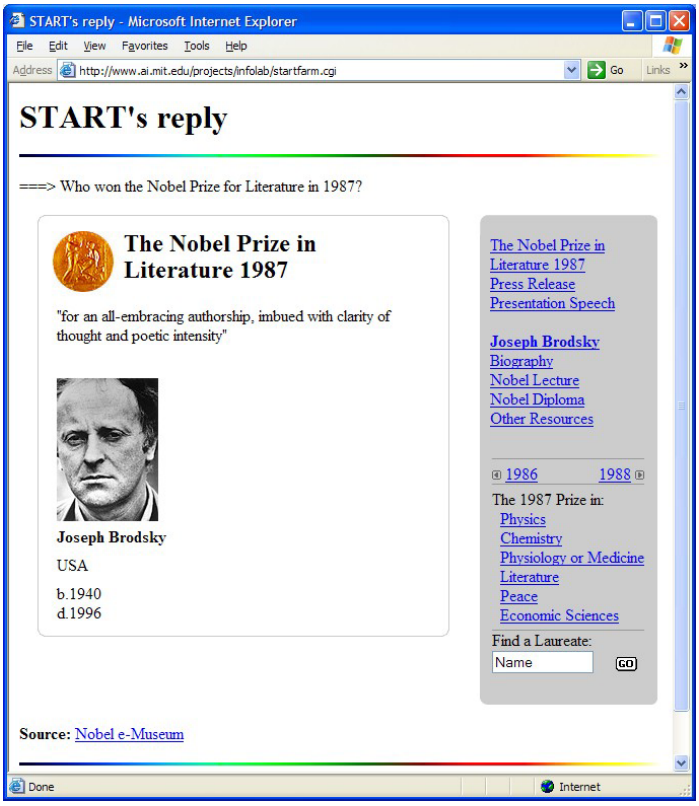

Fig. 1. START answering the question "Who won the Nobel Prize for Literature in 1987?"

class of objects. For example, the annotation "a person wrote the screenplay for `imdb-movie`" can be attached to an Omnibase procedure that retrieves the writers for various movies from the Internet Movie Database (IMDb). The symbol `imdb-movie` serves as a placeholder for any one of the hundreds of thousands of movies about which IMDb contains information; when the annotation matches the user question, the actual movie name is instantiated and passed along to the Omnibase query. After Omnibase fetches the correct answer, START performs additional postprocessing, e.g., natural language generation, to present the answer.

3 Haystack

We are constantly investigating new systems whose integration with START will provide synergy. In this way, Haystack [5] provides a wealth of opportunities from both a research and a practical standpoint. Haystack is a system that aggregates all of a user's information, including e-mail, documents, calendar, and

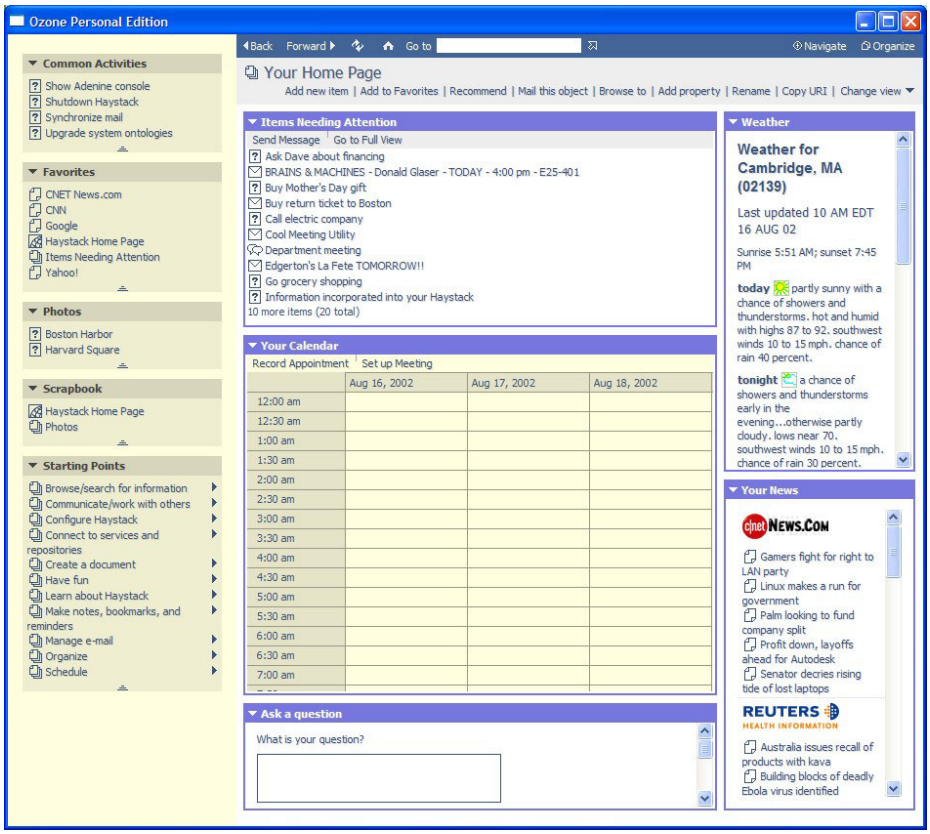

Fig. 2. Screenshot of Haystack

web pages, into a unified repository. This information is described using RDF, which makes it easy for agents to access, filter, and process this information in an automated fashion. As a motivating example, consider a query such as "show me the letter from the woman I met with last Tuesday from Human Resources." Current information technology allows our computers to store all of the information necessary to answer this question. However, it is scattered amongst multiple systems; an agent resolving this query would need to be able to communicate with an e-mail client, a calendar, the filesystem, and a directory server. By reducing the protocol barriers to information—standardizing on RDF as a common model for information—agents are free to mine the semantics of a user's various data sources and not be bogged down by syntactic barriers.

Figure 2 shows a screenshot of Haystack. In this scene, a user's Haystack "home page" is depicted, in which the user is given a snapshot of her e-mails and to-do items, her calendar, applicable weather and news reports, and access to a question answering service.

In addition to being an end-user application for managing information, Haystack also serves as a powerful platform for experimenting with various information retrieval and user interface research problems. By incorporating natural language search capabilities into Haystack, we are able to both demonstrate the usefulness of natural language search and show its applicability to the Semantic Web in general.

4 Towards Human-friendly RDF

RDF [10, 3] is the lingua franca of the Semantic Web, providing a standardized data model for allowing interchange of metadata across the Internet. In short, it is a portable representation of a semantic network, a labeled directed graph. Nodes in the graph fall into two classes: resources and literals. Resources are concrete objects or abstract concepts such as `http://www.cnn.com/` or a person. Literals are string values used for defining primitive properties of resources, such as names. The basic unit of information in RDF is the statement, consisting of a triple of subject (a resource), predicate (an arc in the graph), and object (another resource or a literal).

In its original form, RDF was meant for consumption by computers, not humans. Our central idea for bridging this gap between the core Semantic Web data model and natural language revolves around the application of the natural language annotations technology employed by START. In essence, we propose to "tag" fragments of RDF with language to facilitate access.

Suppose we want to endow Haystack with the ability to answer the following "family" of questions about various attributes (e.g., state bird, state flower, state motto, population, area, etc.) of states:

> What is the state bird of California?
> Tell me what the state motto of Massachusetts is.
> Do you know Colorado's population?
> What is the capital of Kentucky?

Fortunately, the data necessary to answer such questions can be easily found on the Web.[4] However, in order for this data to be usable by any Semantic Web system, it must be restructured in terms of the RDF model.[5]

Ordinarily, RDF data is written in XML syntax; however, as it was designed for machine-to-machine interchange, this syntax tends to be cumbersome and difficult for humans to read. In order to facilitate frequent manipulation of RDF data, Haystack provides a programming language called Adenine specifically suited for these purposes. Adenine incorporates features of Lisp, Python, and Notation3 [1]. A full specification of Adenine's syntax and semantics is beyond the scope of this paper (for more information, please refer to [5]), but a brief overview is presented here. Adenine's basic data unit is the RDF triple. RDF

[4] `http://www.50states.com/`
[5] That is, until Web sites start exporting the contents of their sites in RDF.

triples are enclosed in curly braces {} and are expressed in subject-predicate-object order. A semicolon denotes that the following predicate-object pair is to assume the last used subject. URIs can be specified either in fully canonical form using angle brackets (e.g., `<http://www.w3.org/>`) or using prefix notation (e.g., `rdf:Property`). RDF literals are written as strings in double quotes. Finally, DAML+OIL[6] lists (i.e., Lisp-style lists) are written with an at sign followed by parentheses `@(...)`.

In this paper we will use Adenine syntax for expressing RDF data. The following Adenine code declares the `:State` class and the `:bird` property as well as some basic information about the state of Alabama.

```
@prefix dc: <http://purl.org/dc/elements/1.1/>
@prefix :   <http://www.50states.com/data#>

add { :State
     rdf:type        rdfs:Class ;
     rdfs:label      "State"
}

add { :bird
     rdf:type        rdf:Property ;
     rdfs:label      "State bird" ;
     rdfs:domain     :State
}

# ... more property declarations

add { :alabama
     rdf:type        :State ;
     dc:title        "Alabama" ;
     :bird           "Yellowhammer" ;
     :flower         "Camellia" ;
     :population     "4447100"

     # ... more information about Alabama and other states
}
```

Adenine also supports imperative functions, called "methods", that can take parameters and return values. A unique feature of Adenine is that methods compile into RDF, i.e., each Adenine instruction is encoded as a node in the RDF graph, and a sequence of instructions is expressed by `adenine:next` arcs between these instruction nodes. As a result, data and procedures can be embedded within the same RDF graph and can thus be distributed together.

Adenine uses tabbing to denote block structure as Python does. Function calls and instructions are expressed in prefix notation; for example, `= x 1` assigns the value 1 to the variable x.

[6] `http://www.daml.org/2001/03/daml+oil-index.html`

Given this description of Adenine, we can now express the connection between the RDF schema and the natural language annotations in a *natural language schema*, as follows:

```
@prefix nl: <http://www.ai.mit.edu/projects/infolab/start#>

add { :stateAttribute
     rdf:type        nl:NaturalLanguageSchema ;

     # This annotation handles cases like "[state bird] of [Alabama]"
     # and "[population] of [Maine]".
     nl:annotation   @( :attribute "of" :state ) ;

     # Code to run to resolve state attribute
     nl:code         :stateAttributeCode
}

add { :attribute
     rdf:type               nl:Parameter ;
     nl:domain              rdf:Property ;
     nl:descriptionProperty rdfs:label
}

add { :state
     rdf:type               nl:Parameter ;
     nl:domain              :State ;
     nl:descriptionProperty dc:title
}

# The identifier [state] will be bound to the value of the named
# parameter :state. The identifier [attribute] will be bound to the
# value of the named parameter :attribute.
method :stateAttributeCode :state = state :attribute = attribute
     # Ask the system what the [attribute] property of [state] is
     return (ask %{ attribute state ?x })
```

The definition of `:attribute` restricts the resource representing the attribute to be queried to have type `rdf:Property`; furthermore, the `rdfs:label` property should be used to resolve the actual literal, e.g., "State bird" or "population". Similarly, `:state` restricts the resource to have type `:State` and to have the resolver `dc:title`. In short,

```
@( :attribute "of" :state )
```

is a stand-in for any natural language phrase such as *state bird of Alabama*, *population of Maine*, *area of California*, etc.

Given this natural language schema, Haystack and START can now answer questions about various natural attributes of states. The process of answering a question such as "what is the state bird of Alabama?" is as follows:

1. START parses the question and determines that `:stateAttribute` is the relevant natural language schema to invoke.
2. START extracts the natural language bindings of `:attribute` and `:state`, which are "state bird" and "Alabama", respectively. This is further resolved into the RDF resources `:bird` and `:alabama`.
3. As a response to the question, the method `:stateAttributeCode` is invoked with named parameter `:attribute` bound to `:bird` and named parameter `:state` bound to `:alabama`.
4. The invoked method performs a query into Haystack's RDF store, which returns "Yellowhammer", the state bird of Alabama.

Because the user query is parsed by START, a single natural language annotation is capable of answering a wide variety of questions:

What is the state bird of California?
Tell me what the state motto of Massachusetts is.
Do you know Colorado's population?
What is the capital of Kentucky?

For example, START knows that a possessive relation can also be expressed as an *of* prepositional phrase. In addition, START is capable of normalizing different methods for requesting the same information, e.g., imperative ("Tell me..."), interrogative ("What is...").

As another example, consider the following natural language schema:

```
add { :stateAttribute
     rdf:type        nl:NaturalLanguageSchema ;
     nl:annotation   @( :state " has the largest " :comparisonAttribute ) ;
     nl:code         :maxComparisonAttributeCode
}

method :maxComparisonAttributeCode :comparisonAttribute = attribute
        return (ask %{
                rdf:type ?x :State ,
                adenine:argMax ?x ?y 1 xsd:int %{
                        :attribute ?x ?y
                }
        } @(?x))
```

Instead of a simple request for information, the method invoked by the natural language schema queries the RDF store for the resource of type `:State` that contains the maximal integer value for the property given by `:comparisonAttribute`. As a result, this schema would allow a system to answer the following questions:

Which state has the largest population?
Do you know what state has the largest area?

We have built a prototype implementing the natural language schemata described above. The system is currently limited in the types of questions that it can answer and the domain; in fact, START can easily handle the types of questions discussed above. However, we believe that the system is a proof of concept that demonstrates a viable method of marrying natural language with the Semantic Web. Naturally, more development of our system is required to validate our approaches.

In our vision of the Semantic Web, natural language schemata, such as the ones presented above, would co-exist alongside RDF metadata. These schemata could be distributed (e.g., embedded directly into web pages) or centralized; either way, a software agent would compile these schemata into a question answering system capable of providing natural language information access to users.

5 Further Integration

In addition to our working prototype of natural language schemata, we have further explored other methods of integrating natural language technology with the Semantic Web. Specifically, we propose two additional opportunities for the integration of natural language technology with the Semantic Web.

5.1 Adding Language to RDF Properties

We have noticed a striking similarity, both in form and in spirit, between RDF triples and START's ternary expression representation of natural language [9, 6]. To support a seamless integration of the two technologies, we propose to hook natural language annotations directly into `rdf:Property` definitions.

To illustrate our proposed approach, consider this fragment of an ontology modeling an address book entry in Haystack:

```
add { :Person
      rdf:type          rdfs:Class
}

add { :homeAddress
      rdf:type          rdf:Property ;
      rdfs:domain       :Person ;
      rdfs:range        xsd:string ;

      nl:annotation     @( nl:subject " lives at " nl:object ) ;
      nl:annotation     @( nl:subject "'s home address is "
                           nl:object ) ;
      nl:annotation     @( nl:subject "'s apartment" ) ;

      nl:generation     @( nl:subject "'s home address is "
                           nl:object )
}
```

The `:homeAddress` is a property specifying a user's home address. Our annotation expresses this connection concretely in natural language, via the `nl:annotation` property. For example, the phrase "`nl:subject` lives at `nl:object`" is linked to every RDF statement involving the `:homeAddress` property, where `nl:subject` is shorthand for indicating the subject (domain) of the relation, and `nl:object` is shorthand for the object (range) of the relation. From this, a natural language-aware software agent could answer the following English questions:

Where does John live?
What's David's home address?
Tell me where Bob's apartment is.

In addition, the `nl:generation` property specifies a natural language rendition of the knowledge, allowing software agents to present meaningful, natural sounding responses to users:

Question: Where does Jimmy live?
Jimmy's home address is 200 Technology Square.

By "hooking" natural language annotations directly into RDF property definitions, we can not only ensure that our triples "make sense" to a user, but also provide natural language question answering capabilities simultaneously with minimal cost to the knowledge engineer.

5.2 Natural Language Plans

Consider the question "how do I get from Dave's apartment to John's apartment?" A person faced with this question would first lookup the address of Dave's apartment, i.e., from the user's personal address book, and then find the address of John's apartment using the same method. Given the two address, the user would probably then use a mapping service, e.g., MapQuest, to obtain directions from one address to the other. People generally have no difficulty describing in natural language a "plan" for answering questions that require multiple operations from different sources. Could humans "teach" such plans to a computer directly? Currently, the answer is no, because existing mechanisms of knowledge acquisition require familiarity with precise ontologies, something that cannot be realistically expected for all users. Despite having plenty of common sense, most users cannot become effective knowledge engineers. We propose to utilize natural language annotations to address this difficulty in imparting knowledge to computers.

We propose to capture human-like question answering knowledge in "natural language plans," which can dramatically simplify the task of knowledge engineering:

```
add { :directionsPlan
     rdf:type          nl:NaturalLanguagePlan ;
```

```
    nl:annotation     @( "directions from " :location1
                         " to " :location2 ) ;
    nl:annotation     @( "getting from " :location1
                         " to " :location2 ) ;

    nl:plan           @(
         ${   nl:annotation  @( "What is the address of "
                                :location1 "?" ) ;
              nl:result      :address1
         }
         ${   nl:annotation  @( "What is the address of "
                                :location2 "?" ) ;
              nl:result      :address2
         }
         ${   nl:annotation  @( "How do I get from "
                                :address1 " to " :address2 "?" ) ;
              nl:result      :directions
         }
    ) ;
    nl:action         :displayDirections
}

method :displayDirections :directions = directions
    # Some code to display this information
    print directions
```

Instead of directly manipulating RDF, which would require knowledge of domain-specific ontologies, we could use natural language itself to describe the process of answering a question. The answer plan (`nl:plan`) reflects the user's thought process expressed in natural language: first find the respective addresses of the desired locations, and then obtain directions. In the fragment above, the `${}` operator denotes an RDF anonymous node whose properties are given by the predicate-object pairs within the curly braces {}.

This method of specifying schemata essentially serves to capture the intuitive thought patterns of a human, and allows ordinary users to "teach" a computer knowledge using natural language.

5.3 Hiding the details

Ultimately, the actual details of natural language annotations should be hidden from the user behind GUI authoring tools, so that she need not come into direct contact with XML or RDF. Haystack's user interface was specifically designed with these needs in mind. Additionally, an authoring tool could pre-parse the natural language annotations and store those representations (essentially triples themselves) alongside the annotations.[7] With both natural language and parsed

[7] Without an authoring tool, such a scheme would not be feasible because we cannot expect humans to manually generate parse structures. Note also that keeping natural

representations at their disposal, software agents would have even greater flexibility in manipulating metadata.

6 Deploying the Semantic Web

We believe that natural language annotations are not only an intuitive and helpful extension to the Semantic Web, but will also assist in the deployment and adoption of the Semantic Web itself. The primary barrier to the success of the Semantic Web is a classic chicken-and-egg problem: people will not spend extra time marking up their data unless they perceive a value for their efforts, and metadata will not be useful until a "critical mass" has been achieved. Although researchers have been focusing on ontology editors to reduce barriers to entry, such initiatives may not be sufficient to overcome the hurdles. As James Hendler [4] remarks, lowering markup cost is not enough; for many users, the benefits of the Semantic Web should come for free.

Haystack takes a relatively unique approach to bringing the Semantic Web to the average computer user: Semantic markup should be a by-product of normal computer use. The act of organizing documents, entering contact information, or replying to e-mails in Haystack all result in the creation of semantic markup. In addition to the relatively structured information that can be entered with graphical user interfaces, natural language descriptions can be used as an alternative input modality for entering information. This markup then serves as a rich source of semistructured information that can then be queried by advanced natural language question answering systems such as the one described here.

By providing "natural" means for creating and accessing information on the Semantic Web, we can dramatically lower the barrier of entry to the Semantic Web. Natural language support gives users a whole new way of interacting with any information system, and from a knowledge engineering point of view, natural language technology divorces the majority of users from the need to understand formal ontologies and precisely defined vocabularies.

Furthermore, just as RDF schemata can be shared on the Semantic Web, natural language schemata, also being expressed in RDF, can be shared in the same fashion. Clients such as Haystack will be able to interact with global ontology directories and download both the RDF schema and the natural language schema for any data types encountered by the user. On the flip side, new data types invented by the user can be uploaded to these directories. Our technology of information access schemata provides a system for creating these annotations suitable for different levels of user experience. Novices to the Semantic Web merely have to tag resources with a short natural language description in order to access to those resources later on using natural language. For more advanced users, the ability to access RDF directly and manipulate it using Adenine allows finer-tuned control, greater flexibility, and more concise descriptions.

---

language annotations makes it possible for them to be re-analyzed later as more powerful parsers become available.

By facilitating the creation, display, retrieval, and sharing of natural language schemata, we are enabling users to interact with information on the Semantic Web in an intuitive fashion. We believe that these benefits go a long way in advancing the concept of the Semantic Web.

7 Patterns of Information Requests

Now that we have addressed the question, "are natural language annotations a good idea?" let us turn to the question, "are they enough?" Specifically, can information access schemata achieve broad enough knowledge coverage to be useful? We believe the answer is yes.

Natural language annotations can serve as more than metadata; they can capture generalized patterns of information access. As shown in the previous sections, our annotations can be parameterized to encompass entire classes of questions. For example, our prototype can answer questions about half a dozen attributes of any state, which translates into hundreds of possible questions. A schema about the CIA Factbook in which the properties and countries are parameterized can answer tens of thousands of potential questions. The cost of writing schemata is not proportional to the number of class instances but rather to the complexity of the class itself. A single schema to the Internet Movie Database, for example, could grant the user natural language access to over three hundred thousand titles! Furthermore, because a natural language engine analyzes the questions, simple grammatical alternations would be handled automatically without requiring additional annotations.

It is our empirical experience that people ask the same types of questions frequently [11, 8]. Thus, information access schemata are an effective way of achieving broad knowledge coverage at reasonable costs.

8 The Future

Much like the development of the Semantic Web itself, early efforts to integrate natural language technology with the Semantic Web will no doubt be slow and incremental. However, we believe that our prototype system demonstrates a step in the right direction, and that our proposals sketch out a path for future developments. By weaving natural language annotations into the basic fabric of the Semantic Web, we can begin to create an enormous network of knowledge easily accessible by both machines and humans alike. Furthermore, we believe that natural language querying capabilities will be a key component of any future Semantic Web system.

9 Acknowledgements

Special thanks to David Karger for his insightful comments on drafts of this paper. Thanks to David Huynh and Vineet Sinha for numerous conversations about

the Semantic Web, as well as reading earlier drafts of this paper. This research is funded by DARPA under contract number F30602-00-1-0545, the MIT-NTT collaboration, a Packard Foundation fellowship, IBM, and MIT Project Oxygen.

References

1. Tim Berners-Lee. Primer: Getting into RDF and Semantic Web using N3, 2000.
2. Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.
3. Dan Brickley and R.V. Guha. RDF vocabulary description language 1.0: RDF Schema. W3C Working Draft, World Wide Web Consortium, April 2002.
4. James Hendler. Agents and the Semantic Web. *IEEE Intelligent Systems*, 16(2):30–37, 2001.
5. David Huynh, David Karger, and Dennis Quan. Haystack: A platform for creating, organizing and visualizing information using RDF. In *Proceedings of the Eleventh World Wide Web Conference Semantic Web Workshop*, 2002.
6. Boris Katz. Using English for indexing and retrieving. In *Proceedings of the 1st RIAO Conference on User-Oriented Content-Based Text and Image Handling (RIAO '88)*, 1988.
7. Boris Katz. Annotating the World Wide Web using natural language. In *Proceedings of the 5th RIAO Conference on Computer Assisted Information Searching on the Internet (RIAO '97)*, 1997.
8. Boris Katz, Sue Felshin, Deniz Yuret, Ali Ibrahim, Jimmy Lin, Gregory Marton, Alton Jerome McFarland, and Baris Temelkuran. Omnibase: Uniform access to heterogeneous data for question answering. In *Proceedings of the 7th International Workshop on Applications of Natural Language to Information Systems (NLDB 2002)*, 2002.
9. Boris Katz and Patrick H. Winston. Parsing and generating English using commutative transformations. AI Memo 677, MIT Artificial Intelligence Laboratory, 1982.
10. Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) model and syntax specification. W3C Recommendation, World Wide Web Consortium, February 1999.
11. Jimmy J. Lin. The Web as a resource for question answering: Perspectives and challenges. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC-2002)*, 2002.