Basic Concepts for Managing Semi-structured Information in Haystack

Dennis Quan DQUAN@MIT.EDU
David F. Huynh DFHUYNH@AI.MIT.EDU
Vineet Sinha VINEET@AI.MIT.EDU
David Karger KARGER@THEORY.LCS.MIT.EDU
Marina Zhurakhinskaya MARINAZ@MIT.EDU

MIT Artificial Intelligence Laboratory, 200 Technology Square, Cambridge, MA 02139 USA

1. Introduction

The Haystack platform (Huynh *et al.*, 2002) is designed to store and manage personal information such as e-mail, documents, contacts, meetings, etc. These types of data are semi-structured in nature and thus are better expressed as semantic networks rather than as tables. We believe that any system whose primary data model is semi-structured has at least four intrinsic needs. First, as the semantic data networks inevitably grow, there is a need to organize them conveniently and flexibly and to navigate them systematically. Second, since there is an enormous amount of structured and unstructured data in existence, the system must also allow data exchange with the outside world to both extend its knowledgebase and to facilitate collaboration with users of other systems. Third, as a consequence of collaboration, managing the cooperation of services among various systems becomes important. Finally, cooperation requires trust management for different sources of information. In Haystack, we provide built-in solutions for all of these four needs for organization, external data integration, service cooperation, and trust management. This paper discusses the various techniques that we use to fulfill these needs.

2. Resource Description Framework

Haystack makes use of the Resource Description Framework (RDF) (RDF, 1998) as its primary data model. RDF is one of the core technologies developed for the Semantic Web project, designed specifically for modeling semi-structured data. Here the basic unit of data is the RDF statement, which takes the form of a triple of subject, predicate, and object. The predicate is like an arrow pointing from the subject to the object, both of which are like graph nodes.

In addition, the predicates can themselves be treated as graph nodes. If a predicate is thought of as an attribute name, then one can form statements about that attribute itself. For example, *age* can be said to be *numeric*. For this reason we do not distinguish RDF entities based on their functions as subject, predicate, or object, but rather, we distinguish them based on their functions as monikers or content holders. An *RDF resource*, or resource for short, is an abstract concept or concrete entity named by a Unique Resource Identifier (URI). An *RDF literal* is a string or XML fragment that holds some content.

It is often necessary to define the kinds of predicates expected on certain resources. For example, if a resource represents a book, one would expect a predicate named "publisher" to exist. The RDF Schema standard (RDF Schema, 1998) specifies the notions of an RDF class and an RDF property, whereby properties can be associated with classes. Moreover, RDF defines a built-in *type* predicate allowing resources to be declared to be of one or more types. With this notion, classes and properties are simply resources with type *Class* and *Property*, respectively, and properties are expressed by using the URI of the property as a predicate.

3. Organization and Navigation

In order to help the user systematically navigate his or her personal information repository, there needs to be some organization of the data by either the user or the system. In most existing software, the sole facility for organization is a hierarchy of containers (*e.g.*, e-mail folders, file directories). Often, each item to be organized can only fall into one container, when in reality, it might belong in several. Furthermore, once filed away, an item remains out of sight and out of mind; it can no longer serve as a reminder of a pending task associated with it. We resolve these problems by introducing an *inCategory* predicate that links any data item to zero or more *category* resources. In essence, the *inCategory* predicate performs semantic tagging without constraining the storage location of the data item. Each data item can be easily classified into several categories. This categorization mechanism is useful in many scenarios, ranging from orga-

nizing e-mail messages to classifying capabilities of software components and agents.

When the purpose of organization is not categorization but rather set inclusion, we make use of *collections*. A collection is a mathematical set of objects. To make a collection, one uses the *hasMember* predicate to link the collection resource to its elements. As with categories, collections work on the principle of membership, not containment: an item can belong in several collections at once. Collections can be used to manage groups of items, *e.g.*, the set of people in a room, the set of documents and notes relevant to a meeting, or the set of agents' notifications needing the user's attention.

4. External Data Integration

It is important for us to address how Haystack interacts with unstructured data in the existing world. Today, URLs are used to represent documents, images, web pages, and other content accessible on a file system or over the World Wide Web. With the advent of technologies such as XML Namespaces and RDF, a larger class of identifiers called URIs subsumed URLs. Initially, RDF provided a means for annotating web content. Web pages, identified by URLs, could be referred to in RDF statements in the subject field, and this connected the metadata given in RDF to the content retrievable by the URLs. This is a powerful notion because it makes use of the existing storage infrastructure.

However, with more and more content being described in RDF, the question naturally arises: why not store content in RDF? We argue this is not the best solution for two reasons. First, storing content in RDF would be incompatible with existing infrastructure. Second, leveraging existing infrastructure (*e.g.*, HTTP) is more efficient than using RDF encoding to retrieve files. Hence, we do not require that existing unstructured content be stored as RDF but instead store some of the user's unstructured data using existing technologies such as HTTP 1.1 and standard file I/O.

5. Service Cooperation

Applying RDF to describing agent interface, protocol, and endpoint metadata can be done by leveraging existing standards. A specification called Web Services Description Language (WSDL) (Christensen *et al.*, 2001) already provides an XML-based format for describing this metadata. When the XML tags in WSDL are expressed as RDF properties, the querying of connectivity metadata becomes simply a special case of a more general RDF query mechanism exposed by an RDF metadata store. Additionally, by defining custom RDF properties developers can annotate agent descriptions with arbitrary attributes, such as reliability, physical location, or even degrees of user preference.

6. Trust Management

In a system where a myriad of collaborators and computer agents work together to process information, multiple agents often generate conflicting information. Consider an example where one agent determines the due date of a document by using natural language processing and another agent does the same by using the first date it finds. It is important that these agents' results be tagged with authorship metadata so the user can make an informed choice of which result to choose in case of a conflict. We can do this by giving RDF statements identifiers ("reification") and making statements about statements using their identifiers.

Once we have a framework for recording statement metadata, we can examine issues of retraction, denial, and expiration of assertions, *i.e.*, statements asserted by specific parties. Consider an example where an agent is responsible for generating the title property for web pages. Those web pages whose contents are updated daily have titles that change constantly. One approach for handling constant mutations in the information store is to allow agents to replace an outdated statement with an up-to-date version. However, it would be powerful to allow users to make queries of the form "Find web pages that had the title 'Car Maintenance Tips' at some point in time." By allowing agents to retract rather than delete their statements, queries can still be made to retrieve obsolete information because this information is not deleted. Additionally, this system permits users to override a statement made by an agent by denying the statement, yet retains the denied assertion for future reference.

References

Christensen, E. *et al.* (2001). *Web Services Description Language (WSDL) 1.1*. http://www.w3.org/TR/wsdl.

Huynh, D., Karger, D., and Quan, D. (2002). Haystack: A Platform for Creating, Organizing and Visualizing Information Using RDF. *Semantic Web Workshop, The Eleventh World Wide Web Conference 2002 (WWW2002)*. Honolulu, HI. http://haystack.lcs.mit.edu/papers/sww02.pdf.

Resource Description Framework (RDF) Model and Syntax Specification. (1999). http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

Resource Description Framework (RDF) Schema Specification. (1998). http://www.w3.org/TR/1998/WD-rdf-schema/.

Acknowledgements

This work was supported by the MIT-NTT collaboration, the MIT Oxygen project, a Packard Foundation fellowship, and IBM.