Metadata-supported Agent Infrastructure

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

1. Introduction

Agents are often useful for allowing users to delegate tedious or complicated tasks to the system. In many systems, multiple agents work in concert to achieve desired objectives. In these environments, good communication and effective sharing of information between agents enable the kind of synergy typically desired of multi-agent systems. Haystack (Huynh *et al.*, 2002), our personal information repository, uses a shared semi-structured metadata store and a system of agents for helping the user manage his or her information. In order for agents in our system to communicate with each other, interface specifications and endpoint information (*e.g.*, IP address, protocol, etc.) need to be made available. Agents can request this information from the shared metadata repository and submit updates when agents are created, moved, or destroyed.

Additionally, most agent systems utilize shared information spaces for exchanging data, such as those based on a blackboard architecture. A common hindrance is the need to settle on a handful of common protocols for exchanging information with one another. Many solutions have required either reliance on language-specific solutions, such as Java RMI, or object brokers such as CORBA. In these systems, the interface and instance information for agents is separated from the data they produce, making it difficult to declaratively predicate agent invocation on the property of a particular datum in the shared information space. Additionally, by representing data according to a class hierarchy, one loses the ability to perform relational queries.

By using a database to store both interface information and shared data, our system gives agents a unified abstraction for accepting information from other agents, producing information for other agents, and locating other agents in the system. As a result, we believe that a centralized repository for storing agent state and interface information can be used to consolidate the functionalities of a component directory and a separate database. Furthermore, our semi-structured data model permits arbitrary flexibility in describing agents, yielding benefits in how agents can be classified and selected for tasks. In this paper, we discuss the use of a metadata repository in facilitating an agent infrastructure.

2. Metadata

Metadata in the Haystack environment is expressed according to the Resource Description Framework (RDF) (RDF, 1998). In essence, RDF is a format for describing semantic networks or directed graphs with labeled edges. Nodes and edges are named with uniform resource identifiers (URIs), making them globally unique and thus useful in a distributed environment. Node URIs are used to represent objects, such as web pages, people, agents, and documents. A directed edge connecting two nodes expresses a relationship, given by the URI of the edge. A standard called RDF Schema (RDF Schema, 1999) specifies a way for schema writers to define meanings for these edge URIs, which are called RDF properties. Because URIs are globally unique (like Java package names, typically URIs are generated to include an Internet domain name), the possibility of namespace conflict is negligible. A URI can be used as a "contract" since its use implies consistency with the semantics provided by the party defining the URI.

3. Communication Protocol

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.

The abstract agent communication protocol adopted by Haystack is extremely general and able to accommodate many popular protocols, including HTTP GET, SOAP, Java interfaces, and Metaglue. Building upon WSDL, Haystack specifies that agents be able to handle method calls, where methods are named either by a URI or a string (in the case of a Java interface). In addition, methods can take any number of ordered and URI-named parameters and similarly return any number of ordered and URI-named return values.

4. RDF Store

Perhaps the most central agent in Haystack is the RDF store. RDF stores act as metadata databases for the other agents in the system. Agents in Haystack typically use an RDF store to persist their internal state. RDF stores expose a standardized interface with methods for adding, querying, and removing metadata, allowing RDF stores to be replaced without affecting client-side access code.

RDF stores can serve many roles previously held by more specialized servers. For example, the contents of network directories, such as LDAP servers and UDDI servers, can be easily described in RDF and stored in RDF stores. Describing this information in RDF also gives the added benefit of being able to store many kinds of user-defined attributes. Additionally, the support for declarative event triggering upon data mutation accommodates the need for agents to be notified when certain forms of data appear.

5. Bootstrapping

When Haystack is started, a component known as the agent host reads an initialization script that tells the system which agents to start. This script also indicates where to find the primary RDF store and how to start it. This primary RDF store is analogous to the root filesystem on a UNIX machine. Agents can only run from within an agent host, and the assignment of agents to the agent hosts is specified in the primary RDF store. If a change is made in this store assigning an agent to another agent host, perhaps on a different machine, then the agent would migrate to the other machine.

6. Currently Implemented Agents

Agents in Haystack serve many purposes. Modern information retrieval algorithms are capable of grouping documents by similarity or other metrics, and previous work has found these automatic classifications to be useful in many situations. Agents are used in Haystack to automatically retrieve and process information from various sources, such as e-mail, calendars, the World Wide Web, etc. By storing information in the RDF store, agents are able to seamlessly build upon the work of other agents. For example, retrieving e-mail typically starts with the POP3 agent contacting a POP3 server and downloading e-mail into the shared repository. Text extraction agents notice the downloaded e-mail and can convert HTML-formatted mail into plaintext. Clustering and classification agents are able to use the plaintext representations to automatically organize the e-mail into different categories within the same repository. Finally, the inbox agent watches over this e-mail and could potentially bring an item to the user's attention if it appears an important message has not yet been viewed by the user.

7. Writing Agents

In a system such as Haystack, a sizeable amount of code is devoted to creation and manipulation of RDF-encoded metadata. We observed early on that the development of a language that facilitated the types of operations we frequently perform with RDF would greatly increase our productivity. As a result, we have created Adenine, which includes native support for RDF data types and facilitates interaction with RDF stores and agents. Some Haystack agents are implemented entirely in Adenine. Specific information on the language, along with more comprehensive motivating remarks, can be found in (Quan *et al.*, 2002).

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.

Quan, D., Huynh, D., Sinha, V., and Karger, D. (2002). Adenine: A Metadata Programming Language. *Student Oxygen Workshop 2002 (SOW2002)*. Gloucester, MA.

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.