An RDF NetAPI
A Position Paper

Andy Seaborne
Hewlett-Packard Laboratories
Bristol
UK

andy_seaborne@hp.com

ABSTRACT

This paper reports on some initial work on a NetAPI for accessing and updating RDF data over the web. The NetAPI includes actions for conditional extraction or update of RDF data, actions for model upload and download and also the ability to enquire about the capabilities of a hosting server. An initial experimental system is described which partially implements these ideas within the Jena RDF toolkit.

Categories and Subject Descriptors

[C.2.2] Network Protocols, [D.2.11] Software Architectures

General Terms

Design, Experimentation, Standardization, Languages.

Keywords

RDF, Remote query

1. THE NEED FOR A NETAPI FOR RDF

Part of fulfilling the vision of the Semantic Web is the exchange of RDF data between computer systems. The web enables the reuse of document resources so that people, and now systems, can obtain, combine and process information from other systems without explicit producer-consumer relationships being set up.

One precursor for the Semantic Web to achieve critical mass will be a common framework for accessing RDF data, one sufficiently common that the majority of applications will use it, the majority of publication host systems support it, and only specialized applications will choose to develop their own protocols.

Given a common protocol, the network effect of bringing in more and more semantic applications can start. Application developers can concentrate on the application, and not on the plumbing. Tool builders (server and client side) do not need to support multiple protocols, and can target client-side or server-side environments. Publishers can know that many systems and applications can access their RDF information.

In this paper, we argue for the use of “conditional GET” (query) and “partial PUT” (update) as the central operations for accessing RDF datasources. In the outline framework, we incorporate existing actions of HTTP, such as HTTP GET, as the simplest level acting on complete RDF models. But as the semantic web grows, and as large metadata repositories appear, the paradigm shifts to one that is more like database access than web page download.

We have built a small experimental system that demonstrates part of the NetAPI. We do not propose specific protocols here and this is not the only attempt to create remote interaction with RDF data. We are investigating what it would take to create mass deployment on the current web infrastructure.

2. DESIGN CHALLENGES

One challenge is to balance simplicity of implementation, leading to widespread deployment, with adequate functionality. We want a simple, widely used infrastructure.

In any complex system, there is often a gap between the requirements of the application and the capabilities of the general infrastructure that is properly met by domain-specific sub-systems. These domain specific systems appear as “applications” to the infrastructure and as “infrastructure” to the end applications.

Another challenge in the design of a NetAPI for RDF-based systems is to provide a useful protocol that separates the evolution of client and server software in a community that is pulled from diverse research and industrial domains so the protocol should be simple and domain independent.

We envisage further protocols, both on top of the common access framework proposed here and also more specialized protocols for specific uses where the common access framework is insufficient. A basis for a general system will rarely meet the needs of all possible domains directly.

We also want to utilize as much of the common web infrastructure for the access of RDF data, using existing protocols and existing server systems. This decreases the barriers to deployment.

3. OUTLINE FRAMEWORK

We limit the discussion here to operations on single models. Applications will also want to merge data sources, issue queries across several data sources or (consistently) update multiple

Semantic Web Workshop 2002 Hawaii, USA

sources. This should be considered for possible later addition: the focus here is to identify a simple set of clearly defined operations, which will boost the adoption of the semantic web.

There are 3 categories of operations, which have been identified:

- Operations on the RDF data itself (query, update)
- Transfer of complete sets of RDF data as RDF models.
- Operations relating to the hosting server, such as enquiring about its capabilities and the range of operations on particular RDF stores.

We do not see the basic operations of RDF data as a fetch but one that is conditioned to select a subset of the data available. For this we need selection languages – we do not anticipate that one language will meet a sufficiently wide set of needs at the moment so we make the selection language a parameter of the operation. In future, we hope that a single language emerges with sufficient coverage that both RDF data providers and RDF data consumers can expect to find support for it in all toolkits.

3.1 Server Capabilities

The execution model is one where there are client-initiated actions. Some of the operations have optional parameters so client-side toolkits and applications may need to enquire about the capabilities offered by a host server and about the legal operations and parameters on each of the data sources hosted at that server. This reduces to the server needing to provide information about its capabilities, both overall and with regard to specific RDF data sources.

3.2 Model Operations

Operations that fetch or store whole RDF models are the simplest level of operation for an RDF data source and these naturally map to the HTTP operations GET and PUT. These are already supported by existing web servers, where the data source is an RDF document. Simple extension to retrieving the metadata about a web object, instead of the object itself, based on, say, the MIME type is also natural.

Abstractly, the operations are:

- `GET-MODEL(model, format)`
- `PUT-MODEL(model, format)`

These operations give a sea of RDF models with client systems loading RDF data and performing the extraction and merging of data at the client.

Such whole model operations are not suitable where the quantity of RDF information is large, yet the actual information required by the client is small (for example: data about a publication from a digital library), nor does it provide for the update of a data source, only complete replacement.

For these operations, we need operations acting on client-defined sets of RDF statements, not the whole model (which is a set defined by the server).

3.3 Triple operations

In order to operate on specific data sources we propose operations:

- `QUERY(model, language, condition, format)`
- `UPDATE(model, data-to-delete, data-to-add)`

The language parameter means that we can have a variety of languages for extraction of data from RDF sources: we anticipate that this will usually be a query language and there are advantages in providing a common mechanism where possible. However, we recognize that a single language cannot meet the needs of, say, query and of inferencing rules systems at this stage.

The "format" parameter allows variable in the output format even for the same query language (examples: returning bound variables or the RDF subgraph that matched the query: providing XML or N-triple for update).

The `UPDATE` operation provides a set of changes to move a remote model from one consistent state to another consistent state. This reduces the need for locking of models and for server-side lock management.

4. POSSIBLE PROTOCOL STYLES

Given the three categories of operations, we then have a choice in how these are realized in terms of protocol. The semantic web differs from the current, information, web in that the results of information queries are is no longer primarily render for human consumption but are expected to drive further processing at the requestor (see [8] for a discussion of the design of the current web).

Web services does focus on a web where the interaction across the web is between computer systems, and the transferred formats are in support of computer processing, not necessarily for rendering for end-user viewing.

Each has adopted their own architectural style: using HTTP [6] as the application protocol, with a fixed set of operations and content negotiation; or using SOAP [9], running over HTTP or other transports, where the legal set of operations is open-ended and the further parameters are determined by the operation designer.

Here, we argue that there is a small set of well-defined operations, leading to a single protocol that can used for the vast majority of usages. The operation of `QUERY` maps naturally to HTTP conditional GET but `UPDATE` requires specific use of HTTP POST or adding new operations (c.f. WebDAV [7]).

Either approach is possible: either refining use of HTTP or using a pure message passing model. However, HTTP is a deployed protocol for the specific purpose for the existing web. Reusing infrastructure, rather than modifying, will lead to faster adoption as unmodified web servers can be used as the basis.

5. JOSEKI: AN RDF SERVER

Joseki is an experimental system that implements a protocol for `QUERY` and `UPDATE` on top of HTTP POST. It uses RDF model transfer as the message format; we do not suggest that this represents the best choice for a production protocol – it is an outbreak of "next-bench" syndrome.

The current Jena toolkit [2] already support two forms of remote access: it can load a complete RDF model by using HTTP GET and it can also store the RDF triples in a remote database accessed via JDBC. This experimental system adds the capabilities of `QUERY` and `UPDATE`.

The system is packaged as a standard web application for simple deployment in existing servlet engines, or as a standalone server using an embedded web server [10].

5.1 Protocols and Query Language

The protocol consists of a number of layers (from lowest to highest in the stack):

1. A model exchange layer
2. A request-response layer
3. A query and update layer

Each operation is encoded as a single RDF model to give SOAP-like operation, except we have a predetermined, small set of options, with higher levels on the stack attaching their protocol information via properties to the message URI. Both body and header information are contained in the same RDF model. The request-response layer simply matches responses to the request that caused them.

5.2 Conditional Selection Language

The conditional selection language is RDQL, an implementation of SquishQL[1] for the Jena toolkit[2]. SquishQL is an SQL-like query language that matches a graph pattern to a data source; filter functions can restrict the values of variables. RDQL returns both bound variables and the triples that caused the binding. SquishQL has been variously implemented [3, 4, 5].

The query request contains an RDQL query, complete with URI to specify the data source. The response contains the variable bindings in an RDF data structure, and the triples that cause the binding are associated with the particular binding using reification. The return format is N-Triple because there are usually shared bNodes.

5.3 Client Interface

The client-side implements remote query and remote update operations for the Jena toolkit. The same Java interface is provided for remote query as for query of local models and the URI of the data source in included in the query itself as usual. The client creates a remote query engine that takes the URL as the location of the remote server.

The client processes the results through an iterator in the same programming paradigm as for local queries. The only difference is the creation of the query execution object; a remote query engine takes a location URL.

5.4 Server

The server implementation is a conventional servlet in a servlet-container web server, with HTTP POST, and the reply to the POST, used as the transport for model exchange. No server enquiry is implemented although it is planned to do so.

The server is configured with a set of RDF models, each with its own URI, which is independent of the host server, the URL of the servlet and the location within the host of the data for the model. This separation of the name of the data source (the model URI) from the location of the action (the URL where the conditional GET is performed) allows systems administrators control over the location of models on the host systems and hides information such as filenames and JDBC connection URLs. RDF data sources can be relocated on the server without change to the client code.

5.5 Review

This experimental system is not finished. The use of the query paradigm for processing information is well suited to the network environment where operations are coarser grained and higher latency than API calls directly on a model implementation. The application writer sees the conventional query result processing paradigm that masks the network details. Streaming the return of the results is possible but not with the use of RDF as the message exchange layer.

One difference is that the client application can not fully mix the query calls with Jena API calls as is possible when the local model is available. This is because only a partial copy of the RDF data is available at the client as a model.

6. NEXT STEPS

We plan to use the experimental RDF server in the construction of RDF driven applications, to test the protocol structure, the client programming paradigm and the server-side issues in managing RDF data. We also plan to implement `QUERY` on HTTP GET to enable the HTTP caching already deployed in today's web infrastructure.

7. REFERENCES

1. L. Miller, A. Seaborne, A. Reggiori, "Three Implementations of SquishQL, a Simple RDF Query Language", submitted to ISWC2002.
2. The Jena toolkit: HPLabs Semantic Web activity, http://hpl.hp.com/semweb/
3. A. Seaborne, RDQL – RDF Data Query Language. RDQL grammar: http://www.hpl.hp.com/semweb/rdql-grammar.html
4. L. Miller, "Inkling: RDF query using SquishQL", web page: http://swordfish.rdfweb.org/rdfquery/
5. A. Reggiori, D. W. van Gulik, RDFStore, http://rdfstore.sourceforge.net
6. RFC2616, "Hypertext Transfer Protocol -- HTTP/1.1"
7. E.J.Whitehead Jr., "Lessons from WebDAV for the Next Generation Web Infrastructure" http://www.ics.uci.edu/~ejw/http-future/whitehead/http_pos_paper.html
8. R.T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures". Doctoral dissertation, University of California, Irvine, 2000. http://www1.ics.uci.edu/~fielding/pubs/dissertation/top.htm
9. M. Gudgin, M. Hadley, J. Moreau, H.F. Nielsen (editors), "SOAP Version 1.2" (working draft), http://www.w3.org/TR/2001/WD-soap12-part1-20011217/
10. Jetty: http://jetty.mortbay.org/