Sesame RDF Repository Extensions for Remote Querying

Simon Schenk[1] and Josef Petrák[2]

[1]Department of Computer Science,
University of Koblenz–Landau
sschenk@uni-koblenz.de
[2]Department of Information and Knowledge Engineering,
University of Economics Prague
petrakj@vse.cz

Abstract. The Semantic Web is envisioned to be a networked environment of machine understandable and interliked information. Despite this vision, however, most Semantic Web applications working on datasets from multiple sources today replicate all neccessary to a single repository. This repository is then used for query answering. We present mechanisms to formulate views on RDF datasets and to evaluate queries against distributed datasets, which do not require data replication, but work in a truly distributed fashion.

Keywords: RDF, SPARQL, Distributed Querying, Views, Networked Graphs

1 Introduction

Data reuse and integration is the basic idea behind the Semantic Web effort. However, the extremely large and highly distributed setting of the Web makes reuse and integration a difficult task. Although the basic technologies of the semantic web (URIs to denote things, shared ontologies) are well suited for a distributed environment, data reuse and integration today usually takes place in centralized settings. Data integration often is done by replicating all data to a single repository and connecting data sources using procedural code.

Among others, replication leads to problems with (1) staleness, as data is frequently updated, (2) scaleability, as (combinations of multiple) large datasets can not easily be handled and (3) access rights, as not all data will be available for copying.

The Semantic Web consists of of machine understandable information, opposed to the machine readable, but not machine understandable Web we have today. RDF [2] is the basic data model of the Semantic Web. In RDF information is represented as statements of the form `(subject, predicate, object)`. These statements form a graph, when subjects or objects are used in multiple statements. As most other resources on the Web, all components of a statement are uniquely identified using a URI. On top of this basic model, various inferencing mechanisms, e.g. RDF-Schema [2] reused.

Named graphs are tuples of a name (again a URI), and an RDF graph. They allow to refer to a piece of information (the graph) and, using the name as subject or object, to express information about the information contained in the graph. The most commonly used query language for RDF is SPARQL [1]. SPARQL is based on graph pattern matching. A pattern basically is a graph template formulated by using variables in subject, predicate or object positions. The values obtained during graph pattern matching can be used to again create valid RDF — which may but needs not be different from the input graph. Hence, SPARQL is a powerful mechanism for information extraction and reuse.

2 Proposed extensions

We extend the RDF model and SPARQL query evaluation in two ways:

First, we define *Networked Graphs* [3]. A Networked Graph basically is a named graph, which may be defined as a view on one or multiple other RDF graphs. This allows for the easy reuse of existing data, without a need to replicate it. Networked graphs can be formulated using an RDF and SPARQL based syntax. They can easily be exchanged and are upwards compatible with existing RDF infrastructure.

Second, we extend the SPARQL query model, such that the operators of SPARQL take the repository into account, where the data they operate on is stored. If this is different from the local repository, parts of a query are forwarded to be evaluated at other repositories, so called *SPARQL endpoints*. The communication takes place using the standardised SPARQL protocol. Hence, data at remote repositories can transparently be accessed without the need for data replication.

2.1 Sesame architecture

We implement our extensions based on the Sesame RDF repository framework[3]. The Sesame framework architecture consists of the following components: The *repository* is the high–level object which defines the application interface and offers various methods to query data, upload files, extract or manipulate the data. It abstracts from the internal implementation, which consists of so called SAIL objects. So called SAIL (*Storage And Inference Layer*) components abstract the repository implementation from details of storying, or inferencing. A SAIL implements either actual storage of RDF data, of inferencing capabilities based on some underlying storage. Using a set of SAILs for storage and inferencing, the actual behaviour of a Sesame repository is defined by a stack of SAILs. Sesame is shipped with various SAILs for in-memory or on disk storage and for inferencing using RDFs and custom rules. The extensions described here are implemented as two SAILs, which add additional querying and inferencing capabilities. Hence, they can be combined with features implemented in other SAILs.

[3] http://www.openrdf.org/

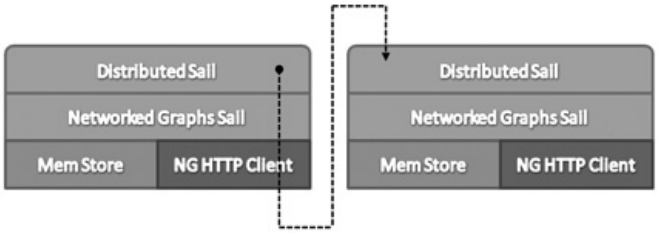

Fig. 1. NG–aware repository architecture and communication flow

In our case, two SAILs are proposed – *Distributed Sail*[4] and *Networked Graphs Sail*[5]. Further, the source finder based on the principle of peer–to–peer is provided to serve as a search engine for the remote SPARQL endpoints for the DistributedSAIL. The SAIL based architecture allow for great flexibility: If only local reasoning with Networked Graphs is required, the Distributed SAIL is not needed. Conversely, Distributed Queries are possible without additional inferencing, if the Networked Graphs SAIL is not used. The full power of both approaches, however, is leveraged when both are used in the same SAIL stack.

2.2 Networked Graphs SAIL

Networked Graphs reasoning is implemented as a SAIL which reuses the query engine of the underlying layer in the stack to evaluate views, but adds some bookkeeping: Making sure all neccessary views are evaluated for a query and detecting cycles. In an uncontrollable environment like the Web, cyclic dependencies among views are hard to detect, but dangerous, as they can lead to infinite forwarding of queries.

Because the SPARQL protocol is based on HTTP and hence stateless, it is not possible to detected, which agent started a request. Therefore, to uniquely identify a query, we send a list randomly generated query IDs with each query. If an endpoint receives a query containing an ID the endpoint has generated itself, a cycle is detected. In the result of the query, the boolean flag "cycle detected" is sent if a cycle query is according to the queries IDs detected, and empty dataset is returned.

We have incorporated the neccessary extensions to the SPARQL protocol definitions, and issue the changed schemas under different XML namespaces[6].

2.3 Distributed SAIL

The distributed SAIL analyses the dataset a query is evaluated upon and locates local graph names and remote graph names available at remote SPARQL endpoints, so we can query multiple sources simultaneously. In detail, the extension parses the query, separates particular graph names, rewrites the queries

[4] http://isweb.uni-koblenz.de/Research/DistributedSPARQL
[5] http://isweb.uni-koblenz.de/Research/NetworkedGraphs
[6] http://isweb.uni-koblenz.de/Research/NetworkedGraphs/2007

and sends them to each local graph and remote endpoints, and merges the result bindings transparently, so as to make the result complete. To make use of our protocol extensions for networked graphs, a modified version of the Sesame HTTP Client is used to send queries and serialize the results.

This solution allows to easily include data from RDF databases like DBPedia[7] without the need to copy them into the local repository. The Networked Graphs SAIL uses the Distributed Sail for actual sending of remote queries.

2.4 Peer-to-peer Source Finder

The Distributed SAIL implementation should include some kind of auto-discovery mechanism for searching for SPARQL endpoints. A component called *source finder* is proposed to fulfil this functionality. A source finder provides a mapping from graph names to endpoint locators, which again are URLs. We use an implementation based on peer-to-peer principles, more specifically on a distributed hash-table as is implemented in P–Grid[8]. Each endpoint collaborating in the peer–to–peer network stores the graph names, which it can answer queries about, in the distributed hashtable. When a query is run, the Disttributed SAIL extracts each graph name from the dataset of the query, the source finder is asked to determine location of any graphs listed in this datased, and modified queries are sent to the endpoint holding the graph.

3 Future work

The presented Sesame extension implementations are intended to offer capability of declarative views to the other RDF graphs and to query multiple endpoints in one time. Currently the basic implementation is working, the extended query and result messages are sent and received. Future work will target on completion of the querying facilities, and optimalization of distributed joins. We plan to extend Networked Graphs towards streaming evaluation of larger datasets.

Acknowledgements The research leading to this paper was supported by the European Commission under contract FP6-027026, Knowledge Space of semantic inference for automatic annotation and retrieval of multimedia content — K–Space.

References

1. E. Prud'hommeaux, A. Seaborne (eds.). SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/, 2007.
2. P. Hayes (ed.). Rdf semantics. http://www.w3.org/TR/rdf-mt/, 2004.
3. S. Schenk and S. Staab. Networked RDF Graphs. Technical report, University Koblenz–Landau, Germany, 2007. http://uni-koblenz.de/∼sschenk/publications/2007/ngtr.pdf .

[7] http://www.dbpedia.org/
[8] http://www.p-grid.org/