PelletServer: HTTP & OWL2 Reasoning

Blazej Bulka[1] and Evren Sirin[1]

Clark & Parsia, LLC, Washington, DC, USA
{blazej,evren}@clarkparsia.com

Abstract. Using OWL2 reasoning services with Pellet requires either interacting with Pellet by using the command line or writing code in Java to invoke its interfaces. Both of these solutions can be impractical in some cases. We present a solution for easier, lightweight integration of Pellet, especially in enterprise environments, by providing access to reasoning services using a RESTful HTTP interface. We intend that PelletServer will provide a common interface for different kinds of OWL reasoning services.

1 Introduction

The Semantic Web attracts significant interest in many application areas and its popularity is still growing. While for some users the Semantic Web is just a convenient and flexible knowledge representation paradigm, the actual ability to define semantics of concepts and reason about them is crucial for a large body of users in academic, medical, and commercial environments. The publication of OWL 2 standard reinforces this trend.

Pellet is an OWL 2 reasoner implemented in Java. As such, programming Pellet is accomplished via a Java API: OWLAPI, Jena,[1] or Pellet internal API. Non-Java applications face a much more difficult task when trying to use Pellet services. Frequently, the only possible options are either using the command-line interface or writing code bridging JVM to the environment of another language. Both of these solutions may involve performance overhead and require significant integration effort. Nevertheless, the growing popularity of Semantic Web motivates the integration of existing software with reasoning services. This increases the diversity of various environments where OWL is used, as frequently these new or existing projects are not Java-based.

A further issue is that OWL-based reasoning often requires significant computing resources — both in terms of processing capabilities and memory — which may not always be available on every system. Moreover, since reasoning can take considerable time, it may not always be practical to perform it when a reasoning service is needed or an application is started (e.g., as it occurs with CLI interface for reasoners, or desktop applications with embedded reasoners; e.g., Protégé).

These issues can be alleviated by providing reasoning as a web service. Such a service can run on a powerful, network-based system (or a collection of systems)

[1] `http://owlapi.sf.net` and `http://jena.sf.net`, respectively

and answer requests from less powerful systems. This approach also amortizes reasoning costs across many requests. Finally, it may provide easier and more lightweight integration for applications written in languages other than Java.

In this paper, we present PelletServer, a RESTful [3] interface to Pellet. It exposes Pellet's reasoning services via HTTP protocol operations. It not only allows more efficient use of computing resources, but also allows for easy integration of non-Java and distributed clients. It paves the way for a common API for different reasoning services and reasoning systems behind a common, easy-to-use façade.

2 REST Design Principles

PelletServer is a RESTful web service implemented using JAX-RS API[2] and Apache's CXF framework.[3] Below we explain how REST design principles were incorporated into the design of PelletServer.

2.1 Resources and Operations

RESTful services focus on the concept of resources whose representations are provided to clients upon invocation of a generic operation (usually an HTTP operation). In PelletServer, these resources are (mainly) ontologies. Further, RESTful services may allow other operations on the resources beyond mere retrieval of the resource's representation. In order to perform these additional operations, clients use other HTTP operations (e.g., `PUT`, `POST`, and `DELETE`). This model of interaction in REST was primarily designed for manipulating collections of data, where the primary operations are create, read, update, and delete.

PelletServer's core use cases are typically of a different kind, that is, not primarily the basic CRUD operations. Clients do not remove or update the ontologies remotely, primarily for reasons of administrative and other feasibility. Instead, they ask questions about an ontology or request some reasoning service to be performed on that ontology (e.g., explaining a particular inference). Thus, core HTTP protocol operations are not especially well-suited to the purposes of OWL reasoning, because most OWL reasoning operations don't map very cleanly or intuitively to CRUD operations. At the very least, there is no obvious one-to-one mapping from HTTP operations to OWL reasoning services.

Since PelletServer's URLs have not only to identify the resource but also a reasoning service, there are two possible URLs "design styles": *resource-centric* and *service-centric*.

For example, the resource-centric URL for retrieving an explanation why a concept *concept* is unsatisfiable in the ontology *ont1* is `/ont1/explain/unsat/concept`. An analogical service-centric URL is `/explain/unsat/ont1/concept`. In short, the difference in these two styles comes down to whether the URL primarily identifies an ontology resource or a reasoning service resource.

[2] `http://jcp.org/en/jsr/detail?id=311`
[3] `http://cxf.apache.org`

The difference seems to us to be of a largely aesthetic nature; after all, one URL style can be automatically converted into the other. There seems to be consensus in the REST community to prefer the resource-centric design of URLs, as more closely capturing the ideas of REST. Thus, PelletServer supports both patterns of URLs to provide users with maximum flexibility.

2.2 Service Discovery

Another important REST design principle is "hypertext as the engine of application state" — a principle we take to be far more crucial than URL design style. The ability of PelletServer clients to dynamically discover the URLs of resources, either ontology or reasoning service, is an important design feature. This feature differentiates RESTful architectures from remote procedure call (RPC) architectural style in which the client and server are tightly coupled with respect to the services they invoke (or provide) and the way these services are invoked (or provided). Since PelletServer is to be the primary point-of-entry to a variety of OWL reasoner products and reasoning services, it is crucial that PelletServer provides loosely coupled service advertisement and, hence, discovery of all reasoning services and capabilities provided by a given instance of PelletServer.

In order to support this functionality, PelletServer provides the description of all resources and services available on the server. That description is available in both JSON and RDF format (depending on HTTP content negotiation), and provides information about the URL required to invoke each service, as well as its parameters (using URI Template syntax), and supported representations for results. (The server-wide information can be obtained by requesting the root document on the server.) PelletServer can also provide information about a single resource (knowledge base), available by using the resource URL.

3 Functionality of PelletServer

We now turn to an overview of the most important capabilities of PelletServer.

Consistency, classification and realization These are the most basic services provided by PelletServer. They are invoked by de-referencing URLs for an ontology named *ont1* thusly: `/ont1/consistency`, `/ont1/classify`, and `/ont1/realize`. The first returns boolean, whether a particular ontology is consistent, while the latter two return the class hierarchy (classification), along with information about types of individuals in this hierarchy (realization). For classification and realization, the results can be returned as either a human-readable text format or an RDF/XML serialization of the class tree or realization hierarchy, respectively.

Query This feature essentially makes PelletServer a SPARQL endpoint that accepts and executes queries and returns query results. PelletServer accepts both SPARQL-DL queries, as well as the recently introduced Terp syntax [5].

Explanation PelletServer allows to explain any inference reached by the reasoner. For example, de-referencing the following URL will generate an explanation for an unsatisfiable class *concept* in the ontology: `/ont1/explain/unsat/concept`.

4 Related work

The idea of transforming a reasoner into a language-neutral, web-accessible service has been proposed a few times. One of the first attempts was DIG [1], and later OWLLink [4] was introduced as an intended successor of DIG. Both DIG and OWLLink described a protocol for transmitting and managing ontology data, and provided access to reasoning services. However, OWLLink is not a RESTful service and does not provide a SPARQL-compliant end-point.

Another implementation of reasoner as a service was described by Bock *et al.*[2]. They presented a client-server framework that provided a common interface for multiple reasoners. The common reasoner interface was based on OWLAPI, and the communication between the client and the server was performed over RMI, and therefore was tightly bound to Java.

One of the most important features of PelletServer is the possibility to discover the services and their capabilities available on a given server instance. Similar work has been done for SPARQL 1.1 in Service Description API [6], which allows to describe capabilities of a SPARQL endpoint.

5 Future work

One of our main goals for the future is to provide support for SPARQL update, and combine it with Pellet Integrity Constraints Validator. We also plan to add additional reasoning services like clustering and machine learning services.

References

1. S. Bechhofer, R. Moeller, and P. Crowther. The DIG Description Logic Interface. In *Proceedings of DL2003 Workshop*, 2003.
2. J. Bock, T. Tserendorj, Y. Xu, J. Wissmann, and S. Grimm. A Reasoning Broker Framework for OWL. In *Proceedings of OWLED*, 2009.
3. R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.
4. T. Liebig, M. Luther, and O. Noppens. The OWLLink Protocol. Infrastructure for Interfacing and Managing OWL2 Reasoning Systems. In *Proceedings of OWLED*, 2009.
5. E. Sirin, B. Bulka, and M. Smith. Terp: Syntax for OWL-friendly SPARQL Queries. In *Proceedings of OWLED*, 2010.
6. G. T. Williams. SPARQL 1.1 Service Description (Working Draft). *http://www.w3.org/TR/sparql11-service-description/*, 2010.