The OWLlink API

Teaching OWL Components a Common Protocol

Olaf Noppens[1], Marko Luther[2], and Thorsten Liebig[1]

[1] Institute of Artificial Intelligence, Ulm University, Ulm, Germany
firstname.lastname@uni-ulm.de
[2] DOCOMO Communications Laboratory Europe GmbH, Munich, Germany
lastname@docomolab-euro.com

Abstract. We introduce the OWLlink API that implements the OWLlink protocol on top of the Java-based OWL API. Besides providing an API to access remote OWLlink reasoning engines, it turns any OWL API aware reasoner into an OWLlink server. As such the OWLlink API provides the missing piece to replace the outdated DIG protocol by OWLlink in applications such as Protégé.

CEUR-WS.org/Vol-614/owled2010_submission_1.pdf

1 Introduction

Imagine you are editing a fancy ontology at your work place using your favorite ontology editor. For validation, however, your laptop lacks the necessary free memory and computing power. Why not access the reasoning engine installed on your company's ultra fast server over the network? Imagine you have to integrate a set of reasoning engines from various vendors, each running on a different platform, into your company's next-generation product, without affecting its stability. Often the only economic solution is to rely on a common protocol for the connection of external components.

In the past DIG [1] has been the standard protocol for connecting applications to Description Logic reasoners. However, DIG's poor support for OWL as well as other conceptual shortcomings have become a major obstacle these days. Therefore the extensible OWLlink protocol [2] has been defined as its follow up. OWLlink is fully aligned with the latest OWL 2 specification and adds modern features like retraction, introspection of capabilities and configurations as well as a rich set of elementary queries.

This paper introduces the OWLlink API,[3] which provides a programmatic interface for the OWLlink protocol on top of the Java-based OWL API [3]. It enables applications to access remote reasoners (so called OWLlink servers) and enhances existing OWL API reasoners with OWLlink functionality.

2 Architecture

The OWLlink API realizes a client and a server adapter (cf. Figure 1). The client adapter allows applications to access OWL reasoner functionality via

[3] http://owllink-owlapi.sourceforge.net/

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

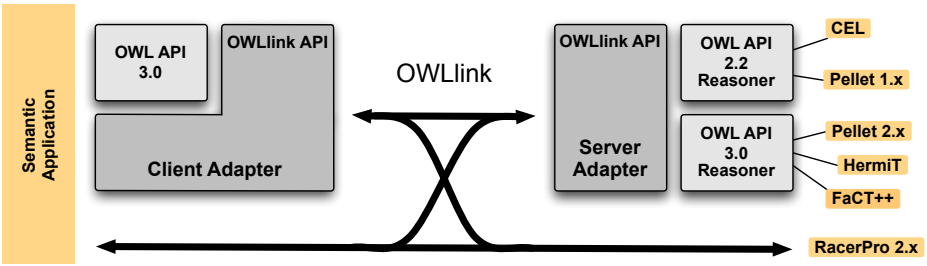

Fig. 1. Architecture

native OWLlink or OWL API 3 Java interfaces. The server adapter enables OWL API 2 or 3 reasoners to speak OWLlink. The OWLlink API implements the HTTP/XML binding of the OWLlink interface and builds on the actual OWL API 3 for the management of OWL 2 ontologies as well as for the parsing and rendering of axioms in OWL/XML syntax.

The client adapter enables Java applications to access remote OWLlink services. It provides an extensible OWLlink specific API that supports all OWLlink core requests and responses. In addition, the client adapter fully integrates with the OWL API structures by implementing its `OWLReasoner` interface. The server adapter offers a framework for developers to enhance their reasoners with OWLlink functionality. It supports OWL API 3 aware reasoners directly and enables OWL API 2 reasoners via a mediator component.

3 Example Use

The `OWLlinkHTTPXMLReasoner` class is an implementation of the OWL API `OWLReasoner` interface, which allows existing applications to access remote OWLlink servers without any further change of their program logic. The following example shows the typical access of a supported reasoner using the OWL API. The only line that is specific for accessing an OWLlink reasoner is line 7 where the actual reasoner object is allocated.

```
1  OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
2  OWLDataFactory dFactory = manager.getOWLDataFactory();
3  OWLOntology ontology = manager.createOntology(IRI.create("http://owled"));
4  OWLClass A = dFactory.getOWLClass(IRI.create("http://owled#A"));
5  OWLClass B = dFactory.getOWLClass(IRI.create("http://owled#B"));
6  manager.addAxiom(ontology,  dFactory.getOWLSubClassOfAxiom(A, B));
7  OWLlinkHTTPXMLReasonerFactory rFactory = new OWLlinkHTTPXMLReasonerFactory();
8  OWLReasoner reasoner = rFactory.createReasoner(ontology);
9  // Now you can use the reasoner as an arbitrary OWLReasoner
10 boolean b = reasoner.isSubClassOf(A, B);
```

To benefit from the additional OWLlink functionality beyond the OWL API `OWLReasoner` interface, such as introspection, Knowledge Base management, parallel handling of Knowledge Bases, and additional queries, one can allocate the `OWLlinkReasoner` interface instead.

```
OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
OWLOntology ontology = manager.createOntology(IRI.create("http://default"));
URL url = new URL("http://localhost:8080");
OWLlinkHTTPXMLReasonerFactory rFactory = new OWLlinkHTTPXMLReasonerFactory();
OWLlinkReasonerConfiguration conf = new OWLlinkReasonerConfiguration(url);
OWLlinkReasoner reasoner = rFactory.createReasoner(ontology, configuration);
// Create a new Knowledge Base...
CreateKB createKB = new CreateKB();
KB kb = reasoner.answer(createKB);
IRI kbIRI = kb.getKB();
// ...and transfer a set of axioms (e.g., from another ontology) to it
Tell tell = new Tell(kbIRI, anotherOntology.getAxioms());
OK ok = reasoner.answer(tell);
GetSubClassHierarchy request = new GetSubClassHierarchy(kbIRI);
ClassHierarchy hierarchy = reasoner.answer(getSubClassHierarchy)
ReleaseKb releaseKB = new ReleaseKb(kbIRI);
reasoner.answer(releaseKB);
```

In this case, the reasoner is also accessible via the standard `OWLReasoner` interface, which performs all method invocations on a knowledge base that can be retrieved via `getDefaultKB()`.

As defined by the OWLlink protocol, requests can be bundled into one request message for efficiency reasons. The OWLlink API supports this by an `answer` method that consumes multiple requests. The resulting `ResponseMessage` contains the responses corresponding to the given requests.

```
ResponseMessage responseMessage = reasoner.answer(query1, query2, query3, ..);
for (Response response : responseMessage)
  // process responses
```

On server side the adapter can turn any reasoner into an OWLlink server as long as the reasoner implements the `OWLReasonerFactory` interface of either OWL API 2 or 3. For most popular reasoning engines (such as HermiT,[4] Pellet[5] and FaCT++[6]) the required `OWLlinkServerFactory` interfaces are already provided. One just needs to make the libraries implementing the reasoner available and start the server from the command line.

For not yet supported reasoners, like CEL,[7] only a factory class with an implementation along the following lines needs to be added.

```
// reasoner factory of your reasoner
OWLReasonerFactory reasonerFactory = ...;
// HTTP port for the server
int port = ...;
OWLlinkServer server = new OWLlinkServer(factory, port);
server.run();
```

4 Discussion

Even for applications running on platforms with in-memory access to reasoners, like Java which features the native implementations Pellet and HermiT as well

[4] http://hermit-reasoner.com/
[5] http://clarkparsia.com/pellet/
[6] http://owl.man.ac.uk/factplusplus/
[7] http://lat.inf.tu-dresden.de/systems/cel/

as CEL and FaCT++ via their foreign function interface, the importance of the standardized remote access to reasoners has been identified, mainly for stability and scalability reasons [4,5,6,7]. While the IYOUIT system [4] already connects to RacerPro[8] using its native OWLlink support, LarKC [6] still makes use of DIG and both HERAKLES [5] and KDI [7] are currently using some proprietary protocol, which in the latter case, is based on the serialization of OWL API structures into remote objects. However, all plan to switch to the common OWLlink protocol.

The OWLlink API client adapter facilitates the use of OWLlink for such Java-based applications in a similar way as the Thea library [8] does for the Prolog platform and is used by the OWLlink plug-in for Protégé[9] that is to replace its outdated DIG handler. Additionally, the OWLlink API server adapter turns existing reasoners supporting the OWL API into OWLlink aware reasoners, ready for remote access. The current OWLlink API implements all of the core protocol as well as the retraction extension. Support for further OWLlink extension will be added in the near future. We are also about to add further protocol bindings besides HTTP/XML, like HTTP/Functional, to ease the integration of further reasoners that come without OWL/XML support on non-Java platforms like the CB reasoner[10] into the Java platform.

Acknowledgements

The authors would like to acknowledge the support of Matthew Horridge who aligned parts of his OWL API implementation to the needs of the OWLlink API and Alan Ruttenberg who tested early versions of the OWLlink API.

References

1. Bechhofer, S., Möller, R., Crowther, P.: The DIG Description Logic interface. In: Proc. of the Int. Workshop on Description Logics (DL'03). (2003)
2. Liebig, T., Luther, M., Noppens, O.: The OWLlink Protocol. [9]
3. Horridge, M., Bechhofer, S.: The OWL API. [9]
4. Böhm, S., Koolwaaij, J., Luther, M., et al.: Introducing IYOUIT. In: Proc. of ISWC'08. Volume 5318 of LNCS., Springer Verlag (2008) 804–817
5. Bock, J., Tserendorj, T., Xu, Y., Wissmann, J., Grimm, S.: A reasoning broker framework for OWL. [9]
6. Huang, Z.: Initial evaluation and revision of plug-ins deployed in use-cases. EU IST FP7 Project Deliverable 4.7.1, The Large Knowledge Collider (LarKC) (2009)
7. Koutsomitropoulos, D., Solomou, G., Pomonis, T., et al.: Developing distributed reasoning-based applications for the Semantic Web. In: Proc. of the IEEE Int. Symposium on Mining and Web (MAW'10). (2010) To Appear.
8. Vassiliadis, V., Wielemaker, J., Mungall, C.: Processing OWL 2 ontologies using Thea: an application of logic programming. [9]
9. Hoekstra, R., Patel-Schneider, P.F., eds.: Proc. of the OWLED'09 Workshop. Volume 529 of CEUR Workshop Proceedings., CEUR-WS.org (2009)

[8] `http://www.racer-systems.com/products/download/preview20.phtml`
[9] `http://owllink-owlapi.sourceforge.net/download.html`
[10] `http://code.google.com/p/cb-reasoner/`