The OWLlink Protocol

Infrastructure for Interfacing and Managing OWL 2 Reasoning Systems

Thorsten Liebig[1], Marko Luther[2], and Olaf Noppens[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. A semantic application typically is a heterogenous system of interconnected components, most notably a reasoner. OWLlink is an implementation-neutral protocol for communication between OWL 2 components. It specifies how to manage reasoning engines and their knowledge bases, how to assert axioms, and how to query inference results. A key feature of OWLlink is its extensibility, which allows the addition of required functionality to the protocol. We introduce the OWLlink structural specification and extension mechanism. Furthermore, we present two extensions, one for retrieving previously asserted axioms and one for retracting axioms from a reasoner. Finally, we describe a binding to HTTP/XML and give an overview of existing implementations.

1 Introduction

Strong evidence suggests that OWL 2 will become an important standard for representing ontologies. According to the W3C, its purpose is to enable applications to meaningfully process information by means of reasoning. However, this requires not only a language standard for ontologies but also a standard way to interact with components that supply reasoning or accompanying services. OWLlink[3] adds this missing piece of ontology infrastructure by providing an extensible protocol for communication among OWL 2-aware systems intended to replace the outdated DIG protocol [1] but based on the DIG 2 proposal [2].

OWLlink consists of a core and a set of extensions as well as respective bindings. The OWLlink core specifies how to introspect the capabilities of an OWL reasoning engine and how to set common or specific system options. It also defines primitives for the handling and manipulation of OWL Knowledge Bases (KBs), such as the creation or deletion of KBs or the successive assertion of single axioms or sets of axioms. Furthermore, the core offers a set of basic queries to access the standard inference services offered by OWL reasoning engines.

[3] `http://www.owllink.org`

OWLlink differs from ordinary ontology APIs, such as the Java-based OWL-API [3], in that it is language-neutral and flexible in how to encode as well as transmit API calls and responses. It supports different transport mechanisms ranging from in-memory access over remote interface calls to Web service invocations. This feature plays an important role within distributed and heterogeneous systems typically found in industry. Here, the distributed architecture results from technical conditions or is implied by requirements such as reliability or exchangeability of components. For instance, the developers of IYOUIT, a context-aware mobile service [4], incorporate OWL reasoning technology in several components, but need to guarantee that a reasoner breakdown will not affect the availability of the whole system. This rules out the use of in-memory connections as provided by the OWL-API.

Extensibility is another important characteristic from the perspective of an application developer. Over time requirements may change or new systems with novel services might emerge. OWLlink provides an extension mechanism that allows the incorporation of additional functionalities as desired. Defining an extension is explicitly intended to be an open and community-driven process. Hopefully, further extensions and bindings will evolve that account for domain specific services which could not be anticipated at this time.

This paper extends a previous publication [5] and introduces the key building blocks of OWLlink. Since its version from October 2008 OWLlink has been updated to the latest OWL 2 specification as well as enhanced with new server configurations and requests which are `LoadOntologies`, `Classify`, and `Realize`. In the following we briefly describe the protocol preliminaries, the core structural specification, the extension mechanism, and the existing as well as potential further bindings. We conclude with a summary and a survey of protocol implementations.

2 Preliminaries

OWLlink is a client-server based on OWL 2. Consequently, OWLlink inherits all of its underlying language concepts such as the notion of structural equivalence. However, it does not support any parts of OWL 2 beyond the level of axioms. Furthermore, the OWLlink specification does not address issues such as transactions, authentication, encryption, compression, concurrency, multiple clients and so on. Some of those features might be provided transparently by the access protocol (e. g., HTTP/1.1) underlying a particular binding.

OWLlink is specified in two parts: the first part defines the structural specification of the protocol, and the second part defines a binding of the protocol to a concrete transport mechanism. The structural specification is introduced within the next sections (3 to 7). Like the OWL 2 structural specification, OWLlink uses a subset of UML class diagram notations within its specification and reuses UML classes provided by the OWL 2 specification [6]. The names of abstract classes (that is, the classes that are not intended to be instantiated) are writ-

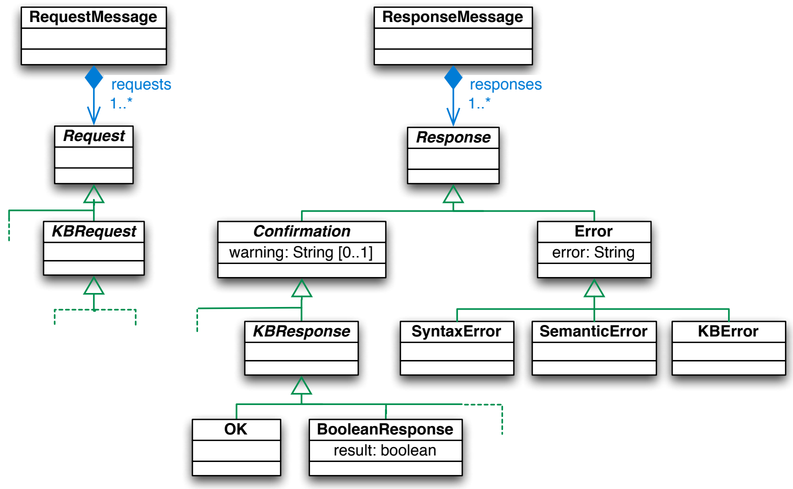

Fig. 1. Basic protocol objects

ten in italic. The names of all OWL 2 UML classes are prefixed with `ox.` to emphasize that they are not defined in the OWLlink specification.

3 Sessions, Messages, and Error Handling

An OWLlink session abstracts the actual bidirectional communication channel between the client and the server. It provides primitives to transport requests and responses. The actual implementation of a session is defined by the transport mechanism used to access an OWLlink server.

OWLlink servers are allowed to service several clients concurrently. However, interaction within one session is not concurrent. A session is assumed to transport requests and responses in sequential order. Each request should be processed by the server such that the results are the same as if the requests were processed sequentially in the order they were dispatched.

The basic interaction pattern is that of request-response. Each request is paired with exactly one response. Depending on the transport mechanism, it might be inefficient to send individual requests to a server separately. Therefore, OWLlink requests are bundled into messages. A `RequestMessage` encapsulates a list of `Request` objects, whereas a `ResponseMessage` encapsulates a list of `Response` objects (cf. Fig. 1). The server must send the responses to the client in exactly the same order in which the requests were received. If a request has been processed successfully, the type of the returned response depends on the request type. For instance, if a request addresses a specific KB, i. e. is a subclass of the abstract `KBRequest` class, the corresponding response has to be a subclasses of the abstract `KBResponse` class. If a request does not produce any specific data, the server should still return a subclass of the general `Confirmation` response,

e. g. an `OK`, to the client. Any confirmation may carry a warning string intended to be meaningful to a human user.

If a request fails, the server should return an `Error` response to the client containing a message specifying the cause for failure. Specific error classes allow the reporting of syntactic violations (`SyntaxError`), semantic problems (`SemanticError`) and issues regarding the management of a Knowledge Base (`KBError`). If a server cannot process a request, it should attempt to recover gracefully, and process other pending requests as if the error did not happen. If, however, this recovery is not possible, the server should send the `Error` response and close the session.

4 Managing Servers and KBs

OWLlink servers have to support the `GetDescription` request, to allow clients to discover their identity and introspect their capabilities. The response to this request is a `Description`, providing information about the server's current state, including: the name of the server, its version, an optional identification message, the protocol version, the currently managed KBs, the supported extensions (see Sect. 7), and a set of configurations.

A `Configuration` is either a `Property` or a `Setting`. While properties are read-only, settings can be adjusted per KB at any time via a `Set` request. The settings given in a `Description` indicate the server's defaults that hold for newly created KBs. The actual settings can be retrieved via `GetSettings`. While OWLlink defines the general format of configurations, it does not provide specific details on available configurations – these will be defined on a per-server basis. However, some configurations (`selectedProfile`, `appliedSemantic`, `supportedDatatypes`, `abbreviatesIRIs`, `ignoresAnnotations`, `ignoresDeclarations` and `uniqueNameAssumption`) have to be supported by any OWLlink server.

OWLlink servers can manage more than one KB simultaneously. A new KB is allocated within the OWLlink server by sending a `CreateKB` request. If the optional argument `kb` is given, the new KB is allocated with the given IRI, otherwise a new (server-generated) IRI is used. On successful creation of the new KB, a KB object containing the IRI that identifies the allocated KB is returned. The optional argument `name` allows to associate a name with a KB, which is then published to other clients (together with its IRI) within the server description.

5 Asserting Axioms

OWLlink relies on the language primitives of OWL 2. With respect to the tell requests – those message parts which add axioms to a KB – this basically means that OWLlink refers to the various axioms about classes, properties or facts defined in Sect. 9 of the OWL 2 specification [6]. A `tell` request contains a set of one or more OWL 2 axioms and will be answered with an `OK` response when successfully processed by the server. Analogous to the definition of an ontology in OWL 2, an OWLlink KB is defined by a set of unordered axioms without duplicates based on the OWL 2 structural equivalence.

6 Asking Basic Queries

The OWLlink core includes a set of general requests for retrieving information about the KB. These so called basic asks cover common queries with respect to the given and inferred axioms of the KB. More complex queries are delegated to query extensions (see the next section). To provide an informal overview, the table in the Appendix lists all of the basic asks. Their semantics and a detailed description of their corresponding responses is given in the OWLlink structural specification.
Within this table "`{O|D}`" abbreviates that this query exists in two flavors, either for Object- or for DataProperties. Furthermore, the "`XXX`" in `Is{O|D}PropertyXXX` is a wild-card for the various characteristics a property can have. The flag *`[dir]`* indicates that the request respects a boolean argument to retrieve only the direct sub- resp. super-classes or properties (default false). The boolean *`[neg]`* argument can be used to retrieve the corresponding negative object and data property assertions of the requests.

Responses of type `SetOfXXXSynset` consist of zero or more synsets of an OWL 2 entity such as NamedIndividuals, Object- or DataProperties, or Classes. Such a synset is a set of one or more elements whose members are all equivalent to each other (i. e., for which mutual equivalence is entailed from the axioms of the KB). In the case this information about equivalence sets is not required (e. g., under the UNA) there are so called flattened variants for those asks which return one single set consisting of named individuals.

7 Extension Mechanism

The OWLlink core is extendable in terms of the supported language fragment, the offered services, as well as provided management tasks. An extension consists of a set of documents specifying the additional messages, a structural specification providing sufficient information about their meaning, and a document per supported binding defining the extra syntax. A server reports the set of extensions supported in the `Description` object by listing their associated IRIs.

To date there are three OWLlink extensions: *Told Data Access*, *Retraction*, and *Ontology Based Data Access (OBDA)*. In the following we briefly describe the first two of them. The third is an extension that provides access to data stored in heterogeneous data sources through a semantic layer in the form of an ontology. The relationship between the data in the sources and the entities (concepts/roles) of the ontology is then expressed through a set of semantic mappings which can be maintained by the OBDA extension via OWLlink.

7.1 Told Data Access

The Told Data Access extension defines a set of queries and their semantics for retrieving previously asserted OWL 2 axioms from a server. Access to told axioms allows distinguishing between explicitly given and inferred axioms. This is of importance for non-standard reasoning services such as black-box debugging or explaining of KBs resp. fractions thereof, and computing the least common subsumer. In addition, it allows a third component to readout assertions for statistical analyses, back up purposes or parallel visualization.

The told data of a KB is defined as the set of successfully received axioms by the hosting OWLlink server since creation of the KB. The extension defines queries to retrieve axioms about OWL 2 entities. There are queries to retrieve class axioms and concept inclusion axioms (GCIs), property characteristics (such as range, domain, etc.) and facts. Additional queries on assertions allow, for example, to retrieve told property fillers or related individuals.

7.2 Retraction

The OWLlink core defines a communication interface which reflects some kind of batch-oriented reasoning procedure that builds up a knowledge base monotonically. After submitting of a set of axioms, a client can pose some queries, add further axioms, query again, etc. Deletion of axioms is only possible by releasing a KB and re-submitting of axioms. The retraction extension allows retraction of previously told KB fragments at the axiom level.

The `Retract` request is the inverse of `Tell` and takes a set of OWL 2 axioms to be removed from the given KB. The removal must be sensitive to the rules of structural equivalence of OWL 2. If all axioms of a retraction request are successfully removed from the KB, the server should respond with an `OK` response.

7.3 Further Extensions

Obviously, the list of possible OWLlink extensions is infinite. However, we want to highlight some extensions that will be defined in the near future. One of the most necessary extensions is an *expressive query language*. To date, there is no commonly agreed conjunctive query language for OWL reasoning systems. The upgrade attempt to SPARQL, namely SPARQL-DL [7], very much relies on the triple structure of RDF and therefore is somewhat orthogonal to the axiomatic representation of OWL. On the other hand, query languages such as nRQL [8] are very expressive but tailored to one specific implementation.

In the case of very large ontologies and frequent updates, it might be advisable to retrieve the answers to even the basic asks not as a whole but as differences from a previous answer. An *incremental answer extension* could for example send only the changes of the class hierarchy since the last hierarchy request.

Most current OWL reasoning engines have been designed and optimized for more or less static KBs. However, a recent trend is the application of inference services to applications with frequently changing data like those underlying situation-aware mobile services [9]. The concept of Stream Reasoning [10] couples reasoners with stream management systems to achieve reasoning in near real time. This coupling could be supported by an OWLlink *publish-subscribe extension* that allows the establishment of subscriptions to published queries. As soon as the result set of a query changes, based on the continuous stream of axiom assertions and retractions, the subscriber would be informed.

8 Bindings

An OWLlink binding specifies how request-response pairs are transmitted between the client and server. There are many possible bindings. The following briefly describes the probably most commonly used binding, namely XML over HTTP. Other bindings may utilize SOAP or another particular Remote Message Invocation protocols and encode axioms utilizing the OWL 2 functional-style syntax. Even mapping the OWLlink specification to an API of a programming language for in memory communication is a possible as well as a desired binding.

The HTTP/XML binding of OWLlink uses HTTP for exchanging XML content between a reasoner and a client. An OWLlink session is mapped to an HTTP connection and is typically established upon sending the first request. The XML schema is obtained by a straightforward translation of the objects from the structural specification: the names of XML elements correspond to the names of the corresponding UML classes. It relies on the OWL 2 XML serialization for the primitives of the ontology language. As a result, implementors of the HTTP/XML binding can re-use their implementation of OWL 2 parsers to read the OWL 2 specific contents of the tell and ask primitives.

9 Status and Outlook

We have introduced the extensible OWLlink protocol which facilitates client applications to configure a reasoner, to transmit OWL 2 ontologies or fragments thereof, and to access reasoning services via a set of basic queries. The protocol has eliminated many deficiencies of its predecessor DIG [11], such as limitations in the supported language fragment, and explicitly is defined on a more abstract level as well as language independend than a programming interface like the OWLAPI[4], SPARQL-DL[5], or SPARQL/Update.[6] It enables the communication among OWL components that are implemented on platforms or in implementation languages not covered by existing OWL 2 APIs, like the OCAML-based CB[7] or a Ruby-on-Rails based Web shop.

[4] `http://owlapi.sourceforge.net/`
[5] `http://www.webont.org/owled/2008dc/papers/owled2008dc_paper_8.pdf`
[6] `http://jena.hpl.hp.com/~afs/SPARQL-Update.html`
[7] `http://code.google.com/p/cb-reasoner/`

The OWLlink specification has been aligned with the OWL 2 recommendation. It consists of a core and three extensions as well as two different bindings (HTTP/XML and HTTP/Functional) containing many examples. Furthermore, there is now the first server side OWLlink implementation with the freely available RacerPro 2.0 preview[8] that implements both bindings and the retraction extension. On the client side, the latest version of the Prolog library for OWL 2 Thea[9] now fully implements the OWLlink HTTP/XML binding. An OWLlink connector for the Java-based OWLAPI that supports both, the client and the server side, will follow very soon. It will enable OWLAPI-aware reasoners, such as Pellet, FaCT++ and Cel, and OWLAPI-based clients, like Protege 4, to communicate via OWLlink.

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. Turhan, A.Y., Bechhofer, S., Kaplunova, A., Liebig, T., Luther, M., Möller, R., Noppens, O., Patel-Schneider, P., Suntisrivaraporn, B., Weithöner, T.: DIG2.0 – towards a flexible interface for Description Logic reasoners. In: Proc. of the OWL Experiences and Directions Workshop at the ISWC'06. (2006)
3. Horridge, M., Bechhofer, S., Noppens, O.: The OWL API. In: Proc. of the 3rd OWL Experiences and Directions Workshop at the ESWC'07. (2007)
4. Böhm, S., Koolwaaij, J., Luther, M., Souville, B., Wagner, M., Wibbels, M.: Introducing IYOUIT. In: Proc. of the 7th Int. Semantic Web Conference (ISWC 2008), Karlsruhe, Germany, Springer Verlag (2008)
5. Liebig, T., Luther, M., Noppens, O., Rodriguez, M., Calvanese, D., Wessel, M., Möller, R., Horridge, M., Bechhofer, S., Tsarkov, D., Sirin, E.: OWLlink: DIG for OWL 2. In: Proc. of the Fifth OWL Experiences and Directions Workshop (OWLED 2008), Karlsruhe, Germany (2008)
6. Motik, B., Patel-Schneider, P.F., Parsia, B.: OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. W3C Proposed Recommendation, 22 September 2009, World Wide Web Consortium (2009)
7. Sirin, E., Parsia, B.: SPARQL-DL: SPARQL Query for OWL-DL. In: Proc. of the OWLED 2007 Workshop on OWL: Experiences and Directions. (2007)
8. Wessel, M., Möller, R.: A High Performance Semantic Web Query Answering Engine. In: Proc. of the Int. Workshop on Description Logics (DL2005). (2005)
9. Luther, M., Böhm, S.: Situation-Aware Mobility: An Application for Stream Reasoning. In: Proceedings of the 1st International Workshop on Stream Reasoning. Volume 466 of CEUR Workshop Proceedings., CEUR.org (2009)
10. Della Valle, E., Ceri, S., Barbieri, D., Braga, D., Campi, A.: A First step towards Stream Reasoning. In: Proceedings of the Future Internet Symposium. (2008)
11. Dickinson, I.: Implementation experience with the DIG 1.1 specification. Technical Report HPL-2004-85, Hewlett-Packard (2004)

[8] `http://www.racer-systems.com/products/racerpro/preview/`

[9] `http://www.semanticweb.gr/thea/`

Appendix: List of Basic Asks and Responses

	Ask		KBResponse
KB Entities	GetAllClasses		SetOfClasses
	GetAllObjectProperties		SetOfObjectProperties
	GetAllDataProperties		SetOfDataProperties
	GetAllAnnotationProperties		SetOfAnnotationProperties
	GetAllIndividuals		SetOfIndividuals
	GetAllDatatypes		SetOfDatatypes
Status	IsKBSatisfiable		BooleanResponse
	IsKBDeclaredConsistent		BooleanResponse
	GetKBLanguage		StringResponse
Schema	IsClassSatisfiable		BooleanResponse
	IsClassSubsumedBy		BooleanResponse
	AreClassesDisjoint		BooleanResponse
	AreClassesEquivalent		BooleanResponse
	GetSubClasses	*[dir]*	SetOfClassSynsets
	GetSuperClasses	*[dir]*	SetOfClassSynsets
	GetDisjointClasses		SetOfClassSynsets
	GetEquivalentClasses		SetOfClasses
	GetSubClassHierarchy		ClassHierarchy
	Are{O|D}PropertiesEquivalent		BooleanResponse
	Is{O|D}PropertySatisfiable		BooleanResponse
	Are{O|D}PropertiesDisjoint		BooleanResponse
	Is{O|D}PropertySubsumedBy		BooleanResponse
	Is{O|D}PropertyXXX		BooleanResponse
	GetSub{O|D}Properties	*[dir]*	SetOf{O|D}PropertySynsets
	GetSuper{O|D}Properties	*[dir]*	SetOf{O|D}PropertySynsets
	GetDisjoint{O|D}Properties		SetOf{O|D}PropertySynsets
	GetEquivalent{O|D}Properties		SetOf{O|D}Properties
	GetSub{O|D}PropertyHierarchy		{O|D}PropertyHierarchy
Facts	AreIndividualsEquivalent		BooleanResponse
	AreIndividualsDisjoint		BooleanResponse
	IsInstanceOf		BooleanResponse
	GetTypes	*[dir]*	SetOfClassSynsets
	GetFlattenedTypes	*[dir]*	SetOfClasses
	GetDisjointIndividuals		SetOfIndividualSynsets
	GetEquivalentIndividuals		SetOfIndividuals
	GetFlattenedDisjointIndividuals		SetOfIndividuals
	Get{O|D}PropertiesOfSource	*[neg]*	SetOf{O|D}PropertySynsets
	GetObjectPropertiesOfTarget	*[neg]*	SetOfObjectPropertySynsets
	GetDataPropertiesOfLiteral	*[neg]*	SetOfDataPropertySynsets
	Get{O|D}PropertiesBetween	*[neg]*	SetOf{O|D}PropertySynsets
	GetInstances	*[dir]*	SetOfIndividualSynsets
	GetObjectPropertyTargets	*[neg]*	SetOfIndividualSynsets
	GetDataPropertyTargets	*[neg]*	SetOfLiterals
	Get{O|D}PropertySources	*[neg]*	SetOfIndividualSynsets
	GetFlattenedInstances	*[neg]*	SetOfIndividuals
	GetFlattenedObjectPropertyTargets	*[neg]*	SetOfIndividuals
	GetFlattened{O|D}PropertySources	*[neg]*	SetOfIndividuals
	AreIndividualsRelated	*[neg]*	BooleanResponse
	IsIndividualRelatedWithLiteral	*[neg]*	BooleanResponse