Undefined 1 (2009) 1–5
IOS Press

Interoperable read-write Linked Data application development with the LDP4j framework

Editor(s): Name Surname, University, Country
Solicited review(s): Name Surname, University, Country
Open review(s): Name Surname, University, Country

Miguel Esteban-Gutiérrez [a,*], Nandana Mihindukulasooriya [a], Raúl García-Castro [a] and Asunción Gómez-Pérez [a]

[a] *Center for Open Middleware*
Ontology Engineering Group, Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid, Spain

Abstract. Enterprises are increasingly using a wide range of heterogeneous information systems for executing and governing their business activities. Even if the adoption of service orientation has improved loose coupling and re-usability, applications are still isolated data silos whose integration requires complex transformations and mediations.

However, by leveraging Linked Data principles those data silos can now be seamlessly integrated, and this opens the door to new data-driven approaches for Enterprise Application Integration (EAI). In this paper we present LDP4j, an open source Java-based framework for the development of interoperable read-write Linked Data applications, based on the W3C Linked Data Platform (LDP) specification.

Keywords: Enterprise Application Integration, Linked Data, Linked Data Platform, LDP4j

1. Introduction

Nowadays every organization uses several information systems to manage their information and integrating those applications is a key requirement for efficiently executing the business processes of these organizations. Enterprise Application Integration (EAI) techniques, which propose solutions to this problem, have evolved over time from ad-hoc one-to-one integrations to approaches such as Service-Oriented Architectures (SOA) and Enterprise Service Buses (ESBs), using either SOAP-based or RESTful web services. Application integration using read-write Linked Data is a novel approach that is getting traction in the industry[1].

Application connection (interfaces) and data integration are the two main problems in application integration. With regards to data integration there are three main challenges: *syntactic heterogeneity*, *structural heterogeneity*, and *semantic heterogeneity* [1]. The usage of standard data exchange formats such as XML or JSON solves the syntactic heterogeneity problem in current EAI techniques and the structural heterogeneity problem is solved via complex schema transformations. However, since the explicit semantics of data is not clearly expressed, traditional approaches struggle with semantic heterogeneity.

*Corresponding author. E-mail: mesteban@fi.upm.es.

[1] https://jazz.net/story/about/about-jazz-platform.jsp

0000-0000/09/$00.00 © 2009 – IOS Press and the authors. All rights reserved

Nevertheless, Semantic Web technologies provide better solutions to the data integration problem. The Linked Data principles help creating a global data space [2] with typed links between data from different sources [3], hence breaking isolated data silos. RDF provides a simple and flexible data model that is well-suited for data integration, and the conceptualization of domain models can be expressed in terms of RDF Schema and OWL ontologies. Machine-readable structured data with explicit formal semantics that are expressed using standards makes merging, integrating, processing, and analyzing data possible without needing out-of-band knowledge or proprietary tools. Links to related entities in data make it possible to start from a piece of data and traverse through different sources with a follow-your-nose approach[2] in order to discover more entities and get context information.

The Linked Data Platform (LDP) specification[3] provides a standard protocol and a set of best practices for the development of read-write Linked Data based on HTTP access to web resources that describe their state using the RDF data model. The standardization of this protocol represents a step forward in the Linked Data community as it lays the ground for the development of interoperable read-write Linked Data applications.

As a consequence of this, LDP provides a base for application integration using read-write Linked Data; however, this approach requires having tools and libraries that provide support for developing read-write Linked Data applications that support the LDP protocol.

At the time of this writing, support for this specification is being included in existing Semantic Web commercial tools as well as in green-field and brown-field open source projects[4]. Unfortunately, in these cases LDP support is provided as a remote data access mechanism, not as an application integration mechanism.

The LDP4j framework is one effort to fill this gap by providing a middleware that facilitates the development of read-write Linked Data applications so that they can benefit from this novel approach of application integration using Linked Data.

This paper presents the LDP4j framework, discusses the lessons learned, challenges, and related future work. The paper is organized as follows. First, Section 2 provides an overview of the LDP specification. Then, Section 3 discusses the requirements beyond LDP for application integration scenarios and Section 4 examines the development considerations that have to be taken into account for the development of read-write Linked Data applications as well as analyzes the technologies that can be used for developing such applications. After that, Section 5 introduces the LDP4j framework, Section 6 provides an evaluation of the framework, and Section 7 presents planned future work. Finally, Section 8 draws some conclusions.

[2] http://patterns.dataincubator.org/book/follow-your-nose.html

[3] http://www.w3.org/TR/ldp/

[4] A list of LDP implementations can be found at https://www.w3.org/wiki/LDP_Implementations.

2. Read-write Linked Data: The Linked Data Platform

The Linked Data Platform (LDP) is an initiative by the W3C LDP WG to provide a standard protocol and a set of best practices for the development of read-write Linked Data applications [4]. Having a standard allows to have interoperability between applications and better application connectivity. The LDP specification extends the HTTP protocol with a set of new constraints, HTTP headers, and Link headers that are useful for read-write Linked Data applications.

The two main concepts defined in LDP include:

- A Linked Data Platform Resource (LDPR) which is any HTTP resource that conforms to the additional constraints defined in the LDP specification.
- A Linked Data Platform Container (LDPC) which is a specialization of an LDPR that acts as a collection resource that helps organizing LDPRs and creating new LDPRs as its members.

The aforementioned LDPRs can be further specialized based on their characteristics. If an LDPR has an RDF representation, it is defined as an *RDF Source* (LDP-RS) and if not, as a *Non-RDF Source* (LDP-NR). LDPCs serve two main purposes: (a) listing its member resources and (b) providing a factory mechanism for the creation of new member resources. LDP containers can be either *Basic Containers*, *Direct Containers*, or *Indirect Containers* depending on their interaction semantics.

3. Interoperable Read-write Linked Data Applications: Beyond the Linked Data Platform

Applications that support the LDP protocol can expose all or part of their data using one or more vocabu-

laries and can consume Linked Data from other applications by following links and traversing through data. This opens the door to a novel approach for integrating applications [5]. However, one of the lessons learned while developing LDP4j was that as a middleware provider LDP support is not enough to get adopted as a viable approach in industry. In order to integrate heterogeneous applications using read-write Linked Data in a production environment, several quality requirements have to be fulfilled and this section discusses some of these requirements[5].

Most enterprise applications have security requirements that generally include authentication, authorization, accounting, integrity, confidentiality, and non-repudiation. The LDP protocol must not just be integrated with current security solutions for web applications (*e.g.*, WebID [6]) but also explore the specificities that stem from the way in which data are made available.

Furthermore, applications need support for business transactions [7] to ensure consistency. Depending on the level of consistency required, strong ACID properties [8] or other alternatives such as BASE [9] must be guaranteed. Several RESTful transaction models have been proposed for web applications in the last few years with some limitations [10] and a transaction model for LDP should be built using them as the base.

Finally, data validation is a vital step for ensuring the quality of data in applications and expressive schema languages and related tools are essential for effective data validation. In contrast with relational databases and XML, RDF is built upon the Open World Assumption[6] and the Non-unique Name Assumption[7], and this makes data validation challenging, as modeling languages (RDF Schema and OWL) are more more suited for inferring than for validation. Thus, there is a need for new standards and tooling support[8].

[5]For an extended discussion of the requirements refer to [5].

[6]The assumption that the truth value of a statement may be true irrespective of whether or not it is known to be true. It is the opposite of the closed-world assumption, which holds that any statement that is true is also known to be true.

[7]The assumption that distinct ground terms denote different individuals.

[8]http://www.w3.org/2014/data-shapes/charter

4. Developing a Read-write Linked Data Application

This section discusses the development considerations when building a read-write Linked Data application and the current solutions in the context of Java-based applications, including commercial and open source LDP implementations.

4.1. Development considerations

Building an interoperable read-write Linked Data application based on Web standards and good practices requires to consider different aspects and related standards as shown in Figure 1. First, the developers require to have a basic understanding of the LDP specification to map the domain data model and business logic rules to LDP concepts such as the different types of LDP containers and LDP resources. However, if everything has to be built from the scratch the developers will need to have a deep understanding about the communication protocol including details of how the application should respond to each request, which headers should be used, of which additional data have to be amended or modified in each single case. This requires a thorough comprehension of the LDP specification and the other specifications upon which it is built, *i.e.*, RDF 1.1 specifications, IRI related specifications (RFC 3987 [11], RFC5785 [12]), PATCH method (RFC 5789 [13]), Link header (RFC 5988 [14]), additional HTTP status codes (RFC 6585 [15]), Prefer header (RFC 7240 [16]).

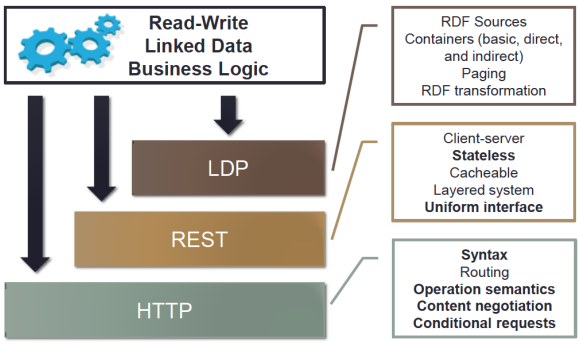

Fig. 1. Developer Considerations

In addition, the developers should have a good knowledge of the REST principles in order to make their application benefit from the good qualities induced by a RESTful design such as scalability, loose

coupling, simplicity, and independent evaluation [17]. The main constraints that a developer has to take into account when building a RESTful LDP application include: separation of concerns using a client-server architecture, increasing reliability and scalability using stateless interactions, improving the efficiency with cacheable responses, reducing complexity using a layered system and increasing simplicity, loose coupling, and independent evaluation with uniform interface. The REST uniform interface constraint is further elaborated with a set of sub-constraints: resource identification (addressability), manipulation of resources through their representations, self-describing messages, and hypermedia as the engine of application state (HATEOAS).

Finally, developers need to know about the HTTP specification. This includes the RFC 2616 [18] and its recent amendments in RFCs 7230-7235 [19,20,21,22, 23,24].

4.2. State of the practice

In this section we analyze the current solutions and libraries that can be used for building a read-write Linked Data application. According to the context of this paper, we limit our scope to Java implementations. For managing RDF there are several libraries that can parse and serialize RDF data, being the most representative ones Apache Jena[9] and OpenRDF Sesame[10]. With regards to LDP, apart from LDP4j, the major open-source products that support the specification are Eclipse Lyo[11], Apache Marmotta[12], and Callimachus[13]. From a commercial standpoint, OpenLink Software's Virtuoso[14] and Base22's Carbon LDP[15] provide LDP support.

On the one hand, Apache Marmotta and Virtuoso provide a domain independent LDP frontend backed by an RDF storage. However, their objective is to be used as LDP-enabled storage services, not to enable exposing the API of an application as read-write Linked Data. In contrast, LDP4j is provided for enabling exposing application APIs as read-write Linked Data application APIs, *i.e.*, instead of storage it provides the means to map the RDF data to the domain data model and drive the business logic of the application using LDP interactions.

On the other hand, Callimachus is focused on the development of portal and web content management systems backed by semantic web infrastructure. On the contrary, LDP4j does not reduce the scope of the target applications that can be developed and exposed as read-write Linked Data applications. Eclipse Lyo's objective is to enable tool integration with OSLC[16]. Thus it is also focused on the development of read-write Linked Data applications, but constrains these applications to OSLC compliant ones. On the contrary, read-write Linked Data applications developed with LDP4j are not domain-constrained.

There are no well known tools that guide developers and facilitate the design and implementation of REST-compliant applications. For instance, there is a lack of frameworks that help application developers to properly use hypermedia controls or manage media types adequately.

In the Java community only the Java Servlet [25] and the Java™API for RESTful Web Services [26] specifications provide a substrate for the development of vendor-independent HTTP-based applications, but both are mainly concerned about the syntax and routing of HTTP operations.

There is a clear gap in tools that are required to build LDP applications that act not only as data storages but as first class applications with their own data model and business rules. The goal of the LDP4j framework is to bridge this gap.

5. The LDP4j Framework

LDP4j[17] is a Java-based framework for the development of interoperable read-write Linked Data applications based on the LDP specification. This framework provides the components required by clients and servers for handling their communication, hiding the complexity of the protocol details from application developers and letting them focus on implementing their application-specific business logic. To achieve this, LDP4j provides the following features:

– **Simplified business object handling**: the framework takes care of lifting from and lowering to an

[9] https://jena.apache.org/
[10] http://rdf4j.org/
[11] http://eclipse.org/lyo/
[12] http://marmotta.apache.org/
[13] http://callimachusproject.org/
[14] http://virtuoso.openlinksw.com/
[15] https://carbonldp.com/
[16] http://www.oasis-oslc.org/
[17] http://www.ldp4j.org/

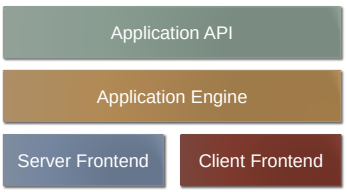

Fig. 2. LDP4j High Level Architecture

intermediate representation that can be automatically unmarshalled from or marshalled to RDF, respectively.

- **LDP support**: the framework takes care of controlling the protocol conversation, managing all the metadata associated to the protocol in a transparent manner.
- **REST aware**: the framework enforces REST best practices and makes them transparent to the user, for instance, publication of RDF URIRefs.
- **HTTP compliant**: the framework takes care of fulfilling the requirements prescribed by HTTP related RFCs (e.g., message Syntax and routing [19], conditional request processing and entity tag handling [21], content negotiation [20]).

In addition, the framework plans extensions to the LDP specification by providing additional features aimed at enhancing the interoperability between LDP-based read-write Linked Data applications (see section 7). Section 6 describes how the framework aligns with the LDP specification as well as how it has been used up to now.

In the following sections we will briefly introduce the architecture of LDP4j (Section 5.1), the application model proposed (Section 5.2), and the way in which the framework works (Section 5.3).

5.1. High level architecture

The LDP4j framework is organized into four building blocks, as shown in Figure 2.

The **application API** provides the means for developing read-write Linked Data applications according to the LDP4j application model that will be presented in the following section.

The **application engine** takes care of (a) handling the LDP protocol conversation, (b) carrying out any required content transformation, and (c) managing the application resources and the endpoints used for publishing such resources.

The **server frontend** handles all the server-side HTTP communication, that is, accepts incoming HTTP requests to the endpoints published by the *application engine* and returns the appropriate HTTP responses to these requests according to the dictate of the application engine.

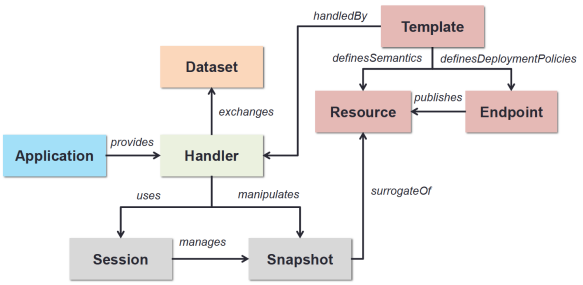

Fig. 3. LDP4j Application Model

Finally, the **client frontend** handles all the client-side HTTP communication, that is, sending HTTP requests to LDP applications and processing the responses returned for these requests.

From all these building blocks, just two are meant to be directly used by read-write Linked Data application developers. Thus, application providers will need to use the *application API* whereas application consumers will have to use the *client frontend*. However, application providers shall also use the *client frontend* if their applications happen to also consume linked data contents from other read-write Linked Data applications.

5.2. Application model

In LDP4j an *application* is defined in terms of **templates** and **handlers**. On the one hand, *templates* are used for defining the semantics of the types of **resources** managed by an application as well as to define the deployment policies of the **endpoints** that the framework will use for publishing these resources. On the other hand, *handlers* are used to implement the business logic required by the templates defined by an application.

Despite the resources and endpoints are managed by the framework, their contents are controlled by the application, which exchanges them with the framework via **datasets**.

In addition, the number of managed resources, their type as well as their relations can be modified using **sessions** that allow a handler to manipulate **snapshots** of these resources. Figure 3 shows how all these concepts are related and Figure 4 shows how are they implemented in the *application API*.

The *templating annotations* allow defining templates for RDF sources and containers (basic, direct,

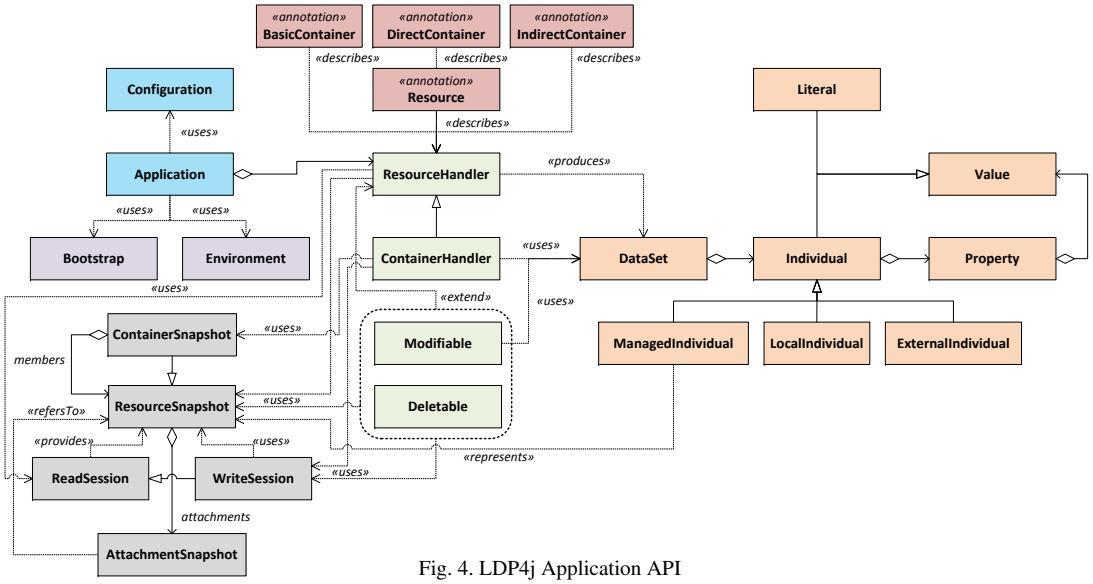

Fig. 4. LDP4j Application API

and indirect). The annotations enable describing templates (identifier, human-name, and description) as well as how are they composed, that is, whether a resource has other resources attached (*i.e.*, subresources) or contains other resources (*i.e.*, containers).

The *handler API* allows implementing both RDF sources and containers, where the main difference among the two is that the latter allows creating new resources. In addition, any of these handlers can be extended to support modification and/or deletion using the appropriate interfaces from the *handler extension API*

The *data manipulation API* is organized around the **dataset**, which is a collection of **individuals** that are defined as collections of multivalued **properties**. The framework distinguishes three types of individuals depending on how these individuals are related to the resources managed by the application: (a) *managed individuals* if they are associated to resources of the application and are published via and endpoint (*e.g.*, an RDF resource identified with a URIRef); (b) *local individuals* if they are associated to resources of the application but are not published (*e.g.*, an RDF resource identified with a blank node); and (c) **external individuals** if they are not associated to resources of the application (*e.g.*, URIRefs out-of-the-scope of the application).

Finally the *application API* also includes the *session API* that provides the means for controlling the lifecycle of the resources owned by an application, and the *application configuration API* that provides the means for the configuration and bootstrap of the application.

5.3. Execution pipeline

The process used for handling the input requests is depicted in the BPMN diagram shown in Figure 5. The processing of a request sent by a client involves different parties: the *Server Frontend*, the *Application Engine*, and the *LDP4j application*.

Upon receiving a request, the first task of the *Server Frontend* consists in **dispatching the request**, that is, identifying whether or not the application exposes an endpoint for handling the request. If no endpoint is available (either because it has never existed or because the endpoint has been already deleted) the appropriate HTTP client error status code is returned (*i.e.*, 404 or 410).

If the *Server Frontend* finds an endpoint then it proceeds to **preprocess the request**. This task consists in: (1) verifying that the resource allows the specified operation; (2) handling content negotiation; (3) processing the entity body if present (*i.e.*, unmarshalling the RDF body and lifting it to the intermediate `DataSet` object representation described in the previous section); (4) analyzing the LDP headers included in the request; and (5) verifying any conditional request constraint specified in the request (e.g., `If-Match` and `If-Modified-Since` headers [21]). If the operation is not supported, content negotiation fails, the body cannot be processed, or the conditional request constraints cannot be satisfied, the appropriate HTTP client error status code is returned (*i.e.*, 405, 406, 400/415, or 412).

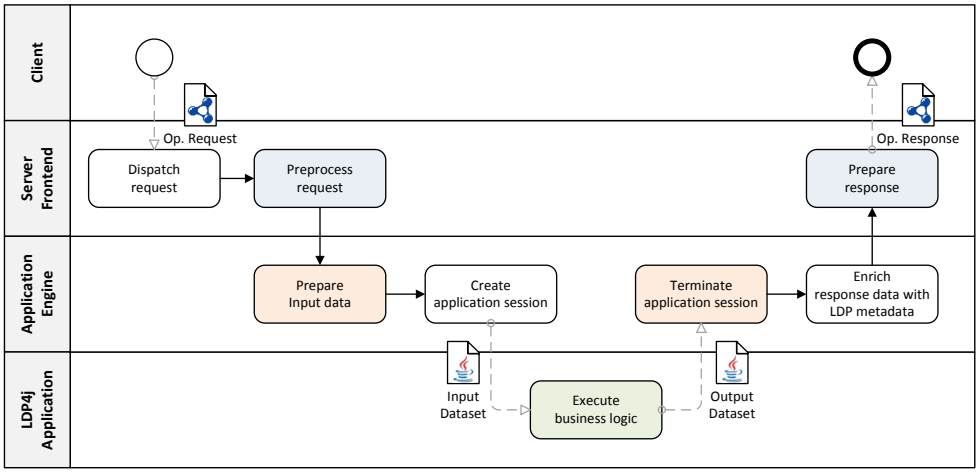

Fig. 5. LDP4j Execution Pipeline

If the *Server Frontend* completes the preprocessing succesfully, the *Application Engine* takes over the processing of the request. Firstly, it takes care of **preparing the input data**, if available. The preparation consists in: (1) checking that the LDP metadata included as part of the input data are consistent with the current status of the resource published via the endpoint to which the request was sent (e.g., in the case of a container, verifying that the container type and associated configuration details have not been changed by the client), and (2) removing any LDP specific details from the lifted `DataSet` intermediate representation, so that the application only has to deal with its business data. If the input request LDP metadata are not consistent the appropriate HTTP client error status code is returned (i.e., 409).

Whenever the input data has been prepared, the *Application Engine* then **creates an application session** and then transfers the control to the *LDP4j application*, in particular to the handler in charge of **executing the business logic** of the application, which will use the application session to notify which resources have to be created, modified, or deleted during the processing of the request. Depending on the operation the handler may use an input `DataSet` representation of a resource or may have to return an `DataSet` representation of the resource.

When the handler finishes, the *Application Engine* resumes the process. The first step consists in **terminating the application session**, that is, to handle the resource and endpoint life-cycle changes specified by the *LDP4j application*. The second step consists in **enriching the response data with LDP metadata** (*e.g.*, if the representation of a container is to be retrieved, the *Application Engine* would enrich the `DataSet` returned by the *LDP4j application* with the members and the container type) taking into account the preferences specified by the client in the request (*i.e.*, via the `Prefer` header [16]).

After the application's response data has been enriched, the *Server Frontend* finishes the processing by **preparing the response**, which consists in lowering the enriched `DataSet`, marshalling the contents RDF, and generating the required LDP headers. Finally, once the response has been created it is returned to the client.

6. Evaluation

6.1. W3C LDP Test Suite Results

The LDP4j framework provides middleware for developing read-write Linked Data applications using the W3C LDP protocol; thus the most appropriate evaluation criterion is the compliance of applications that are built using the LDP4j framework. We have evaluated the compliance using the official LDP Test Suite developed by the LDP WG[18].

The Test Suite contains 235 tests that evaluate whether a given implementations adheres to the rules and restrictions defined in the specification. These tests are organized in two dimensions. First they are divided according to the main features of the specification which include: RDF Sources, Non-RDF Sources, Basic Containers, Direct Containers, and Indirect Con-

[18] `http://w3c.github.io/ldp-testsuite/`

tainers. Then the specification requirements are divided into three compliance levels: MUST, SHOULD, and MAY.

Feature	MUST	SHOULD	MAY
RDF Source	24/24 (100%)	7/7 (100%)	1/1 (100%)
Basic Container	37/37 (100%)	15/17 (88%)	3/4 (75%)
Direct Container	42/42 (100%)	17/19 (90%)	3/4 (75%)
Indirect Container	37/39 (95%)	15/17 (88%)	3/4 (75%)

Table 1
Compliance Test Results

The LDP4j framework implements all the features of the specification except those for Non-RDF Resources. The test cases which are not currently passed are because of the functionalities that are not supported at the moment, *i.e.*, put-to-create and PATCH. The official W3C Test Suite results[19] confirm that the LDP4j framework is compliant with the W3C LDP specification and has the majority of its features currently implemented.

6.2. LDP4j in practice

This section presents two practical use cases where the LDP4j framework provided the necessary components to handle the LDP protocol communication allowing application developers to focus on their business logic. The development of these use cases provides evidence of the usefulness of the framework as well as the reduction of the development time.

6.2.1. morph-LDP: Exposing relational data with R2RML and LDP

morph-LDP [27] is a system powered by the LDP4j framework that allows exposing relational data as read-write Linked Data for LDP-aware applications, whilst allowing legacy applications to continue using their relational databases. The W3C R2RML[20] recommendation defines a language to map relational databases (RDBs) and RDF, and morph-LDP combines the W3C LDP and W3C R2RML standards to enable the consumption of the converted RDF data as read-write Linked Data using the LDP protocol.

This system is useful in the contexts where organizations are willing to get the benefits of Semantic Web technologies but are constrained to continue with their existing relational databases that are already being used by the existing applications to avoid drastic changes to existing infrastructure (see Figure 6). For these organizations, transforming data into an RDF format and managing it from a triple store is not a viable option. morph-LDP is built on top of morph-RDB [28], an R2RML implementation for converting relational data to RDF and vice-versa and it uses LDP4j for exposing those RDF as Linked Data that are not only dereferenceable and available through the HTTP GET operation, but can also be updated using write operations such as PUT, POST, and DELETE. The LDP layer extracts the metadata from the HTTP request, handles the LDP protocol and sends the relevant data as an input for morph-RDB.

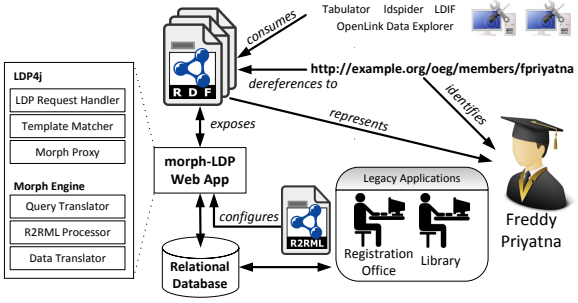

Fig. 6. The morph-LDP use case

6.2.2. LDP adapter for the Bugzilla issue tracker

The LDP protocol provides the basis for a novel paradigm of application integration in which each application exposes their data as a set of Linked Data resources and the application state is driven following the REST design principles. This approach is more suitable when the integration is data-intensive and the traceability links between different applications are important. However, to adapt this approach applications must support the LDP protocol. As we have seen in the early stages of application integration approaches, such as web services, adapters provide a more flexible mechanism to gradually adopting a technology while using the existing tools with minimum changes.

The LDP adapter for the Bugzilla issue tracker [29] provides a novel way to use the functionalities of the application using read-write Linked Data in order to make it an LDP application. In the Bugzilla adapter, the Bugzilla native data model is mapped to the ALM iStack ontology [30] . The adapter exposes the Bugzilla data as LDP resources by transforming the data between the ALM iStack ontology and the

[19] http://www.w3.org/2012/ldp/hg/tests/reports/ldp.html

[20] http://www.w3.org/TR/r2rml/

Bugzilla native model so that LDP clients can consume RDF data from Bugzilla as if it was a native LDP application. The Bugzilla LDP adapter, which is a JavaEE web application, consists of three main layers: (a) LDP layer, (b) transformation layer, and (c) application gateway layer as illustrated in Figure 7.

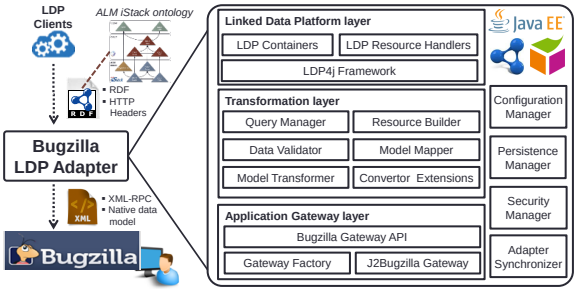

Fig. 7. The Bugzilla LDP adapter

The LDP layer which handles the LDP communications is built using the LDP4j framework. It exposes the Bugzilla data as LDP resources and the concepts such as bugs, products, product versions, and users are mapped to LDP containers which list these entities and allow creating new entities. Each entity is mapped to an LDP resource with its own URI that can be used by clients to access them. This enables the semantic integration of Bugzilla with other LDP-enabled applications and makes possible to have typed links between application data.

7. Future Work

Supporting the whole LDP specification is the first step in the development roadmap of LDP4j, specially providing support for Non-RDF sources and for the `PATCH` operation. However, in order to fully realize the vision of interoperable read-write Linked Data applications it is necessary to make explicit any application-specific domain knowledge required for interacting with the particular application in a sensible way. Further LDP4j work is geared towards the development of a set of extensions for enriching the LDP protocol along this line:

- **Vocabulary support**. Enable the publication and discovery of the vocabularies used by a read-write Linked Data application together with the restrictions that apply. As a side effect, this would relief application developers from the burden of input data validation as this could be transparently done by middleware.
- **Co-reference support**. Provide middleware services for dealing with the *co-reference problem*[21], following the lines already identified in [31].
- **Transaction support**. Provide middleware services for providing transaction support for Linked Data applications using a RESTful transaction model.

Finally, support for other enterprise requirements aimed at facilitating the adoption and uptake of the framework are planned, in particular the provision of Linked Data-specific access control, authorization, and accounting mechanisms.

8. Conclusions

The usage of Linked Data as a means for integrating applications has several advantages over traditional EAI approaches. However, using Linked Data for this purpose requires having mechanisms not just for reading, but for writing data. The LDP specification defines a protocol for read-write Linked Data applications and its standardization represents a big step towards the industrial adoption of Linked Data technologies.

However, the interoperability of applications supporting the LDP protocol is limited to domain-independent concerns (*i.e.*, exchange formats, communication patterns, failure signaling, etc.). Thus, their integration requires out-of-band domain knowledge in order to use them properly while interacting with them via the LDP protocol. As a result, LDP is not enough for Linked Data-based EAI: it is necessary to put domain-knowledge into the game.

In this paper we have presented LDP4j, a framework for the development of read-write domain-aware Linked Data applications. The framework implements and extends the LDP protocol with features for exposing the domain knowledge that dictates how to interact with the application. This framework represents the next big step for realizing the vision of Linked Data-based Enterprise Application Integration.

References

[1] Gagnon, M.: Ontology-based integration of data sources. In: Proceedings of 10th International Conference on 10th International Conference on Information Fusion (FUSION2007). (July 2007) 1–8

[21] When information about a certain entity —which is identified in different manners— is spread across different sources.

[2] Heath, T., Bizer, C.: Linked Data: Evolving the Web into a Global Data Space. Synthesis lectures on the semantic web: theory and technology **1**(1) (February 2011) 1–136

[3] Bizer, C., Heath, T., Berners-Lee, T.: Linked Data - The Story So Far. Semantic Services, Interoperability and Web Applications: Emerging Concepts (June 2011) 205–227

[4] Speicher, S., Arwe, J., Malhotra, A.: Linked Data Platform 1.0 (December 2014) W3C Proposed Recommendation, `http://www.w3.org/TR/ldp/`.

[5] Mihindukulasooriya, N., García-Castro, R., Esteban Gutiérrez, M.: Linked Data Platform as a novel approach for Enterprise Application Integration. In: Proceedings of the 4th International Workshop on Consuming Linked Data (COLD2013), Sydney, Australia (October 2013)

[6] Story, H., Harbulot, B., Jacobi, I., Jones, M.: FOAF+SSL: RESTful Authentication for the Social Web. In: Proceedings of the First Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2009). (June 2009)

[7] Papazoglou, M.P.: Web Services and Business Transactions. World Wide Web: Internet and Web Information Systems **6**(1) (March 2003) 49–91

[8] Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. first edn. Morgan Kaufmann, California, USA (September 1992)

[9] Pritchett, D.: BASE: An ACID Alternative. Queue **6**(3) (May 2008) 48–55

[10] Mihindukulasooriya, N., Esteban Gutiérrez, M., García-Castro, R.: Seven challenges for RESTful transaction models. In: Proceedings of the companion publication of the 23rd international conference on World wide web, Seoul, South Korea (April 2014) 949–952

[11] Duerst, M., Suignard, M.: Internationalized Resource Identifiers (IRIs). RFC 3987 (Proposed Standard) (January 2005)

[12] Nottingham, M., Hammer-Lahav, E.: Defining Well-Known Uniform Resource Identifiers (URIs). RFC 5785 (Proposed Standard) (April 2010)

[13] Dusseault, L., Snell, J.: PATCH Method for HTTP. RFC 5789 (Proposed Standard) (March 2010)

[14] Nottingham, M.: Web Linking. RFC 5988 (Proposed Standard) (October 2010)

[15] Nottingham, M., Fielding, R.: Additional HTTP Status Codes. RFC 6585 (Proposed Standard) (April 2012)

[16] Snell, J.: Prefer Header for HTTP. RFC 7240 (Proposed Standard) (June 2014)

[17] Wilde, E., Pautasso, C.: REST: From Research to Practice. Springer (August 2011)

[18] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard) (June 1999) Obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, 7235, updated by RFCs 2817, 5785, 6266, 6585.

[19] Fielding, R., Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230 (Proposed Standard) (June 2014)

[20] Fielding, R., Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231 (Proposed Standard) (June 2014)

[21] Fielding, R., Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests. RFC 7232 (Proposed Standard) (June 2014)

[22] Fielding, R., Lafon, Y., Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Range Requests. RFC 7233 (Proposed Standard) (June 2014)

[23] Fielding, R., Nottingham, M., Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Caching. RFC 7234 (Proposed Standard) (June 2014)

[24] Fielding, R., Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Authentication. RFC 7235 (Proposed Standard) (June 2014)

[25] Mordani, R.: Java^{TM}servlet specification. JSR 315 (Maintenance Release) (February 2011)

[26] Pericas-Geertsen, S., Potociar, M.: JAX-RS: Java^{TM}api for restful web services. JSR 339 (Final Release) (May 2013)

[27] Mihindukulasooriya, N., Priyatna, F., Corcho, O., Garcıa-Castro, R., Esteban Gutiérrez, M.: morph-LDP: An R2RML-based Linked Data Platform implementation. In: Demo at the 11th Extended Semantic Web Conference (ESWC2014), Crete, Greece (May 2014)

[28] Priyatna, F., Corcho, O., Sequeda, J.F.: Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph. In: Proceedings of the 23rd International World Wide Web Conference, Seoul, South Korea (April 2014)

[29] Mihindukulasooriya, N., Esteban Gutiérrez, M., García-Castro, R.: A Linked Data Platform adapter for the Bugzilla issue tracker. In: Demo at the 13th International Semantic Web Conference (ISWC2014), Riva del Garda, Italy (October 2014)

[30] Esteban Gutiérrez, M., García-Castro, R., Mihindukulasooriya, N.: Implementation and Design of the ALM iStack Proof-of-concept. Final Project Report, Center for Open Middleware (July 2014)

[31] Esteban Gutiérrez, M., García-Castro, R., Mihindukulasooriya, N.: A Coreference Service for Enterprise Application Integration using Linked Data. In: Proceedings of the 7th International Workshop on Applications of Semantic Technologies (AST2013), Koblenz, Germany (September 2013)