

# Evaluation Environment for Linked Data Web Services

Sebastian Bader<sup>1</sup>, Alexander Wolf<sup>2</sup>, Felix Keppmann<sup>1</sup>

<sup>1</sup> Karlsruhe Institute of Technology (KIT), 76133 Karlsruhe, Germany

<sup>2</sup> USU Software AG, 76137 Karlsruhe, Germany

sebastian.bader@kit.edu, al.wolf@usu.de,  
felix.leif.keppmann@kit.edu

**Abstract.** In the last decade many approaches and efforts have been made to introduce semantic services into real-life applications. Nevertheless, Linked Data Web Services are still waiting for their major breakthrough. We state that improving the capability to compare and evaluate different approaches leads to a faster propagation in the field. We therefore propose a straightforward test and evaluation environment for distributed Linked Data Web Services. Our implementation allows the objective comparison of approaches for e.g. API descriptions, service discovery, composition or orchestration.

**Keywords:** evaluation environment, linked data platform, web services

## 1 Introduction

The ongoing digitization brings more and more data providers, functionalities and applications to the Web. For example with the Internet of Things the amount of available data sources will further increase massively [2]. In addition, a rising number of services is consumable via Web APIs. But developers have mostly manually configured consuming applications in mind and therefore document, if at all, their API in human understandable but unstructured forms. This is not sufficient for the upcoming volume and variety of new data producers, devices and applications in the Web. But in order to lift the emerging potential for businesses and end users, the various components have to be interconnected and communicate with each other.

Linked Data has the potential to deliver the syntactic and semantic integration by combining well-understood Web technologies (HTTP, URIs etc.) with semantic declarations. Its consistent treatment of data and communication patterns paths the way for a fast integration of loosely-coupled systems. The thereby established distributed architectures can enable a fast generation of powerful applications by the combination and reuse of services and software artifacts.

However, service and data discovery, composition, and orchestration are demanding challenges which already draw major efforts of the Semantic Web community. But in contrast to the huge volume of developed approaches the actual

progress is hard to measure as most research groups evaluate new approaches on own test cases and in non-replicable environments. Consequently, an objective assessment of the current state of the art is hardly possible.

In this work we focus on the creation of distributed service environments in a declarative manner and make the following contributions: 1) we present an approach on building generic but adaptable service stubs based on Linked Data Platform specifications; 2) we provide and describe a prototypical implementation of these services in a reproducible, distributed, and scalable evaluation environment; 3) we show the feasibility of the approach in a preliminary evaluation.

## 2 Related Work

In the recent years Web Services and APIs are more and more shifted from comprehensive protocols like SOAP towards the more light-wight REST APIs. Versteden and Pauwels present `mu.semte.ch` [8], an exemplary platform for mash-ups of RESTful Linked Data-based microservices. They package the elements of their platform in Docker<sup>3</sup> containers to ease their shipment and reuse. But the adaptability for other scenarios is limited as the `mu.semte.ch` acts as an exclusive middleware for any connected application.

Several activities in the last decade promoted the comparison and competition of approaches and implementations against each other. The Semantic Web Challenge<sup>4</sup> pushed recent developments to adding value for end users. The broad set of criteria allowed a wide range of different use cases. In contrast, the Semantic Web Services (SWS) Challenge `citpetrie2008semantic` and Web Services Challenge (WSC) [1] provided testbeds for specified scenarios and aimed to present and enhance the maturity of semantic web service technologies. But even though these efforts drive the comparability of implementations regarding their scope, a generic method to evaluate developments based on Linked Data services is still missing.

Joshi et al. [4] face a similar task for the creation of RDF data sets. The proposed LinkGen system is capable of generating sets of Linked Data for specified vocabularies, configured through a seed parameter. Even malformed, syntactical and invalid statements are included, making the output realistic for controlled evaluations of new approaches. Nevertheless, LinkGen is only applicable for data generation and not services.

## 3 Service Environment

### 3.1 Preliminaries

Our approach for distributed, scalable, and reproducible service environments is build on top of the Representational State Transfer (REST) paradigm for a

---

<sup>3</sup> <http://www.docker.com/>

<sup>4</sup> <http://challenge.semanticweb.org>

resource-oriented viewpoint on distributed applications and for the interaction between their components. In addition, the Linked Data paradigm for state representations, semantics, and interlinking of resources defines a mature data format for the Web.

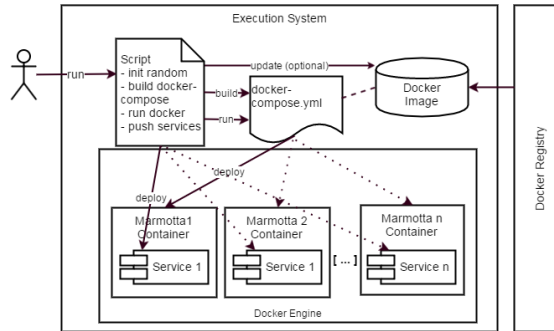
This combination of Linked Data and REST has recently been standardized by the Linked Data Platform (LDP) specification [7] of the World Wide Web Consortium (W3C). It specifies the semantics and interactions for the concept of a resource, in particular of the LDP Resource, with its specializations of the LDP RDF Source which provide state representations that adhere to the RDF data model. As subtype of the LDP RDF Source, the LDP Container with its specializations of the LDP Basic Container enables different kind of one-many parent-child relations between other LDP Container and LDP Resources. Following the LDP specifications all interactions towards container and resources are restricted to basic HTTP methods (GET, PUT, POST, DELETE) and therefore guarantee simple and transparent data manipulation. Yet, only data handling is specified. Web Services of any kind are not regarded by these standards.

### 3.2 Architecture

The main goal of our approach is to support reproducibility in building service environments, while, as sub-goals, support scalability in terms of number of service hosts and adaptability in terms of service functionality. We strictly comply to LDP specifications in order to benefit from their mature interaction handling and data presentation.

The **Configuration Layer** consists of a limited number of configuration arguments, the so-called seed, that guarantees – for equal argument values – the same distribution, scale, and functionality in order to reach reproducibility. At the second layer, the **Declaration Layer**, we generate – in a reproducible way – custom or use given deployment declarations based on the provided configuration arguments of the seed layer. This declarations include 1) a composition declaration, i.e., the specific number of host with their specific configuration, 2) declarations for the host systems, i.e., specify how each host system should be build, and 3) declarations of different service functionalities that should be provided by hosts. The **Deployment Layer** of our architecture is the actual deployment of the service environment, i.e., the deployment of services at real virtual and/or physical hosts with a certain functionality as declared by the second layer.

We enable a declarative adaptation of service functionality by utilizing Notation3 (N3) rule programs, proposed by the Smart Components approach in [5]. This rule programs are evaluated by an interpreter against RDF graphs and enable deduction of new knowledge, data transformation, and via build-in functions HTTP interaction as well as mathematical operations. In our approach we use this rule programs to describe the function between RDF input and RDF output of services. As the program’s data format is N3 and therefore a superset of RDF, they can not be described as LDP RDF Sources but only as LDP



**Fig. 1.** Architecture of evaluation environment

Non-RDF Sources, stored as binary files. The integration of this custom functionality and the execution of the programs are established via HTTP POST on certain 'Start' LDP Resources, which is explicitly marked as optional by the LDP specification and without further implications for LDP Resources. Only for LDP Containers the LDP specification provides specific behaviour for HTTP POST, in particular the creation of child-resources. Relying on LDP Resources, the proposed behaviour on incoming POSTs is therefore in accordance with the LDP specifications but also enables the integration of Web Services.

## 4 Implementation

In the first phase of the deployment process, empty service stubs are deployed as part of an extended Apache Marmotta<sup>5</sup> Linked Data Platform Web server<sup>6</sup>. At this point, the started LDP server contains some basic configurations and is already accessible but has no included data or further functionalities beyond the LDP interaction models. In order to add a Web Service, at least four special LDP Resources have to be created. We will describe the procedure regarding a simple service which applies basic mathematical operations on input values and returns the result.

### 4.1 Deployment

The building blocks of our testing and evaluation environment are depicted in Fig. 1. The main control component is a pythonscript which coordinates the complete setup. Mandatory input parameters of the script are the number of desired instances, a seed number and the host name of the executing machine. Started on a system that fulfills the requirements<sup>7</sup>, the script performs the following tasks in order:

<sup>5</sup> <http://marmotta.apache.org/>

<sup>6</sup> The source code is available through GitHub: <http://github.com/aifb/s2apite>

<sup>7</sup> System requirements: python 3.5+; docker 1.13.0+; docker-compose 1.11.1+

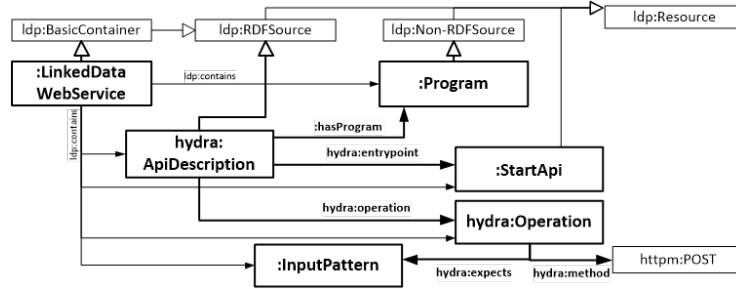


Fig. 2. Basic resources for each service

- (1) Pulling a specific docker image<sup>8</sup> from the public docker repository. This image is used for all containers running in the evaluation environment. It is based on tomcat7:alpine<sup>9</sup> and contains the enhanced version of Apache Marmotta.
- (2) Based on the input parameter, the script creates a docker-compose.yml deployment descriptor. Each docker container gets port 8080 (tomcat/marmotta) mapped to an individual port of the host machine. The port forwarding uses ports in range [9000:9000+n] by default but can be set to any range at starting time. With this information the docker containers are deployed.
- (3) Pseudo-random variants of the service are selected controlled through the seed parameter. For each instance, a new variant of the service program is created (like the one in Fig. 3). As the same seed will always generate the same services, this procedure assures the reproducibility of every scenario.
- (4) The generated programs are pushed to the waiting containers by utilizing the LDP REST-API provided by Marmotta. First, an instance of *:LinkedDataWebService* is required. A *:LinkedDataWebService* is a subclass of the *ldp:BasicContainer* and serves as the Web Service root. It contains all descriptions and links to the other Web resources. Second, the service gets its execution instructions through a *:Program* resource. For now, only declarations in Notation3 are supported. For further details on the rule-based syntax see [3].

Next, a start resource gets posted to the server in the form of a new *ldp:Resource*. This resource must include an RDF statement declaring it an instance of the *:StartAPI* class and conform to the *LDP Resource interaction model*. As the server treats members of this class as triggers for services, in contrast to *ldp:Containers* or *ldp:RDFSources*, this information changes the default handling of resources, in particular on incoming POST requests.

The last deployment step is the creation of a description resource. For this, and most other semantic relations between the LDP resources, the outlined concept relies on the HYDRA [6] vocabulary wherever possible (Fig. 2). For the execution of a Web Service in this context, a *:LinkedDataWebService* container, a *:Program* and a *:StartApi* resource are mandatory with optional *ApiDescriptions*, *Operation* and *InputPattern* declarations.

<sup>8</sup> <https://hub.docker.com/r/aifb/s2apite/>

<sup>9</sup> [https://hub.docker.com/\\_/tomcat/](https://hub.docker.com/_/tomcat/)

The script can also be executed against a docker swarm. In this case the containers are automatically distributed on physical machines which allows to scale the evaluation environment without any further adjustments. By using this approach, the only artifact that is required to setup the evaluation environment is the script itself. Using a certain seed makes environments reproducible.

## 4.2 Service generator

The main purpose of the evaluation environment is to provide an infrastructure for generating reproducible, easy to deploy networks of Web-based and semantically enriched services. Those networks can then be used to test new aggregation, composition or orchestration methods. After the execution of steps (1) and (2) the container with LDP Web APIs are up and running but lack any functionality.

A simple example of a service can be the adding of two numeric values. In our surrounding, such a service is defined by a N3 program, declaring the transformation from input to return values. This program consists of one or many rules (Fig. 3) which here specify the transformation of the variables through a "sum" operator and creates a new triple containing the result.

For test scenarios usually a multitude of different services is required that can then be orchestrated or aggregated. For a reasonable scenario the services should of course differ in functionality and interfaces. To allow the evaluation environment to scale, the service implementations must be generic and parameterizable. But if the parametrization is truly random, an once created scenario can not be repeated. Therefore, we base the selection on a seed value to make the environment with all its services reproducible.

Our evaluation environment setup script creates services that offer for now the functionality of basic arithmetic operations (addition, subtraction, division and multiplication). The services also vary in the number of input parameters. At the moment, the script generates services that apply operations on 2-26 operands. Which operation is selected and how many operands are used is based on a pseudo random generator initialized with the seed.

## 4.3 Execution Engine

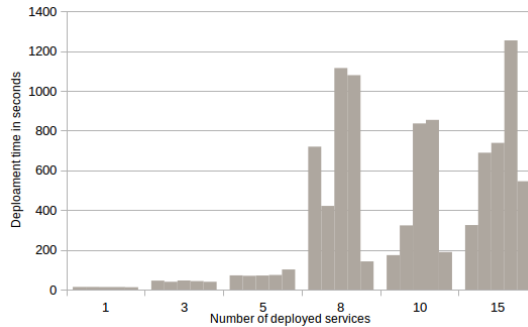
As outlined, the service stubs are filled after they have been deployed. Even more, programs can be modified and even replaced while the instance is active. The Linked Data-Fu engine [3] provides the necessary capabilities. Originally designed for Linked Data Stream integration it is also capable to perform HTTP

```

@prefix ex: <http://example.org/> .
@prefix math: <http://www.w3.org/2000/10/swap/math#> .
    { ?anything    ex:summand1    ?a ;
      ?anything    ex:summand2    ?b .
      (?a ?b)      math:sum        ?sum . }
=> { ex:result    ex:value        ?sum . } .

```

**Fig. 3.** Adding two variables declared with Notation3 rules



**Fig. 4.** Deployment time of Docker container in seconds. Each configuration was conducted five times with one container hosting only one Web Service.

operations in order to interact with RESTful Web resources. As all involved services actually are of the proposed type, a service can select, invoke and change other services and even itself.

Whenever a HTTP POST request is sent to a LDP Resource of the `:StartApi` class, the corresponding functionality is triggered. The server combines an optionally received RDF graph from the request body with the specific program file and starts the engine. The computed RDF triples are then collected and written to the respond message. Therefore, only synchronous requests are possible.

## 5 Evaluation

We claim that the proposed environment can generate a Web like surrounding with minimal effort. In order to give a first impression we therefore measure the necessary computational overhead to establish a certain number of containers. The tests are conducted on an average office laptop<sup>10</sup>.

The results (Fig. 4) show a significant increase of computation time for more than five instances. It is basically caused by the filled main memory of the evaluation machine. In the current state, 1 GB to 1.2 GB of main memory per container are necessary for acceptable responds times. Whenever the docker engine provides less memory, the deployment time for the corresponding instance increases drastically.

We assume that the starting process of the Linked Data server is the main bottleneck. This is supported by recommended settings of Marmotta of about 1 GB. But even if spare memory is available, Docker will start to provide it to the container after an unpredictable amount of time. That is why the reduction of the required memory and the optimization of its distribution is one of our next steps with the highest priority.

<sup>10</sup> i7-4600U with 2.10 GHz & 4 cores, 12GB RAM memory, 64-bit Ubuntu 16.04

## 6 Conclusion

We have outlined an adjustable testing and evaluation infrastructure for LDP-based Web Services. Conforming to LDP specifications, we implemented a functionality to quickly generate large-scale scenarios for distributed applications with a reasonable overhead. The generic Web Service stubs allow functionality adjustments at runtime and therefore support an environment with dynamic changes under laboratory conditions. On top, the combination of a Linked Data server with RESTful Web Service capabilities disperses the separation between data and services.

We illustrated one scenario with an exemplary generator for a set of facile service instances. This specific evaluation design is solely defined by the generator script and the configuration parameters (number of instances and the seed value). It can be deployed anywhere under similar conditions.

Our next steps involve the enhancement of the containers to more elaborate use cases. Asynchronous calls, streaming capabilities and semantic reasoning are not yet possible. In addition, we will examine how more complex services with appropriate semantic descriptions can be generated and the Shapes Constraint Language<sup>11</sup> (SHACL) will be evaluated in order to automatically declare and validate RDF data in order to exactly define input and output data.

**Acknowledgement** The research and development project that forms the basis for this report is funded under project No. 01MD16015 within the scope of the Smart Services World technology program.

## References

1. Bansal, A., Bansal, S., Blake, M.B., Bleul, S., Weise, T.: Overview of the web services challenge (WSC): discovery and composition of semantic web services. *Semantic Web Services: Advancement Through Evaluation* (2012)
2. EMC Digital Universe: The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things. <http://www.emc.com/leadership/digital-universe/> (2014), accessed 30. August 2017
3. Harth, A., Knoblock, C.A., Stadtmüller, S., Studer, R., Szekely, P.: On-the-fly integration of static and dynamic linked data. In: 4th International Conference on Consuming Linked Data. pp. 1–12 (2013)
4. Joshi, A.K., Hitzler, P., Dong, G.: LinkGen: Multipurpose Linked Data Generator. In: *The Semantic Web – ISWC 2016*. pp. 113 – 121. Springer (2016)
5. Keppmann, F.L., Maleshkova, M., Harth, A.: Semantic Technologies for Realising Decentralised Applications for the Web of Things. In: ICECCS. pp. 71–80 (2016)
6. Lanthaler, M.: Hydra Core Vocabulary. <http://www.hydra-cg.com/spec/latest/core/> (2017), unofficial Draft, accessed 07. July 2017
7. Menday, R., Mihindukulasooriya, N.: Linked data platform 1.0 primer, <http://www.w3.org/TR/ldp-primer/>
8. Verstedden, A., Pauwels, E.: State-of-the-art Web Applications using Microservices and Linked Data, <http://ceur-ws.org/Vol-1629/paper4.pdf>, 24.02.2017

---

<sup>11</sup> <https://www.w3.org/TR/shacl/>