Linked Data Services for Internet of Things

Jaroslav Pullmann, Dr. Yehya Mohamad
User-Centered Ubiquitous Computing department
Fraunhofer Institute for Applied Information Technology FIT
Sankt Augustin, Germany
{jaroslav.pullmann, yehya.mohamad}@fit.fraunhofer.de

***Abstract* — In this paper we present the open source "LinkSmart Resource Framework" allowing developers to incorporate heterogeneous data sources and physical devices through a generic service facade independent of the data model, persistence technology or access protocol. We particularly consider the integration and maintenance of Linked Data, a widely accepted means for expressing highly-structured, machine readable (meta)data graphs. Thanks to its uniform, technology-agnostic view on data, the framework is expected to increase the ease-of-use, maintainability and usability of software relying on it. The development of various technologies to access, interact with and manage data has led to rise of parallel communities often unaware of alternatives beyond their technology boundaries. Systems for object-relational mapping, NoSQL document or graph-based Linked Data storage usually exhibit complex, vendor-specific programming interfaces. Paraphrasing the resource concept underlying the RESTful architecture of the Web, we attempt to identify and generalize the main data management tasks in terms of common resource operations.**

Keywords— Data management; Linked data; Internet of Things; OSGi

I. Introduction

The evolution of the World Wide Web from a human-driven, document-centric web to a network of interconnected devices, services and self-descriptive data has stimulated the development of various technologies to access, interact with and manage this data. Communities emerged to apply, enhance and promote their respective technology stack. They often act in parallel, unaware of alternatives beyond their technology boundaries. As part of the open-source LinkSmart project [1] our work on the "LinkSmart Resource Framework" attempts to identify and generalize the main data management tasks and prevalent concepts regardless of the actual data model, persistence technology etc. With a clear focus on "data services", it converges a variety of technologies ranging from Object-relational mapping, NoSQL document storage up to graph-based Linked Data being handled uniformly by a generic set of services. In the following, we first introduce the rationale, design considerations and general architecture underlying the LinkSmart Resource Framework. We then describe in detail the Metadata extension and conclude with some examples of how this new development is being used in EU funded research projects GreenCom and ALMANAC.

II. LinkSmart Resource Framework

A. Rationale

The Java Data Objects standard [2] specifies an interface to persist Java objects in a technology agnostic way. The related Java Persistence specification [3] concentrates on object-relational mapping only. We consider both specifications technology-driven, overly detailed with regards to common data management tasks, while missing some important high-level functionality. The rationale underlying the LinkSmart Resource Platform is to define a generic, uniform interface for management, retrieval and processing of data while maintaining a technology and implementation agnostic facade. It does not aim to supersede dedicated persistence interfaces and systems, but to proxy and enhance client interactions with them by means of a homogeneous service layer. Similar to "hybrid storage systems" like Virtuoso Universal Server [4] it shares the idea of transcending technology boundaries within a unified data management hub.

Adopting Fielding's concept of Web resource [5] we define *managed resource* as the central entity of our framework. It is a uniquely identified endpoint with an arbitrary number of discrete, digital representations. A resource can then be manipulated and accessed via a uniform service interface by exchanging resource operation requests. One major contribution of our work is in the analysis and provision of a generic taxonomy of operations for resource management, access and retrieval in terms of Java application programming interfaces (APIs). On top of this native, uniform API custom mappings to remote protocols has been defined. Due to their contractual nature, RESTful interactions are often subject to controversial discussions. There is no normative way to map individual constituents of the HTTP interface – request method (verb), request URI, set of HTTP headers, query and form parameters or a message payload – to data operations. Provision of a tool base for defining such a homogeneous remote APIs for RESTful operations on arbitrary resources is a further contribution of our work.

Management of Linked Data [6] was in the past performed in a proprietary, application-dependent way until recently the Linked Data Platform 1.0 Standard [7] sought for alignment with the RESTful principles of the Web as well to standardize

HTTP-based application integration patterns for read-write Linked Data. An original contribution of our work is in the seamless integration of Linked Data management into the Resource Management framework, the provision of a native Java API and an extensible set of services to facilitate integration and processing of Linked Data.

B. *Design considerations*

The following high-level considerations lead the design of our framework:

Uniform: The uniformity and simplicity of Web's HTTP interface has significantly contributed to its success and unprecedented growth. Ease-of-use, learnability and maintainability of the platform should be based on the definition of a set of simple, generic service interfaces and concepts.

Protocol-agnostic: The whole range of synchronous service interactions should additionally be modeled as asynchronous message calls. Such the platform will offer for both, tight synchronous and loosely-coupled asynchronous integration scenarios and allow for binding the uniform interface to a variety of communication protocols (HTTP, MQTT etc.)

Service-oriented: Service oriented architectures (SOA) focus in contrast to the object-oriented paradigm (OOP) on functional singletons, services exchanging light-weight, "passive" data objects. The functionality has been engineered out of heavy-weight objects into a manageable set of dedicated services.

Representation-based: In compliance to the service-oriented architecture, data structures exchanged by the system should omit any business logic and be deliberately simple, optimized for serialization in widely adopted textual formats like JSON, XML, Turtle [18] etc.

Resource-driven: Data manipulation and retrieval is implemented by adopting a RESTful paradigm. The internally maintained state (managed resource) is updated by the exchange of resource representations and the application of a uniform command set.

Domain-agnostic: In order to increase reusability and prevent the provision of custom APIs for every particular domain the interfaces should be designed as generic and domain agnostic as possible, while allowing for definition, maintenance and querying of arbitrarily complex data models.

Standards-compliant: The risk of creating a proprietary solution should be minimized by compliance to standards. Related knowledge, support and open source software tools are re-used where possible.

C. *Architecture*

As depicted in Figure 1 the LinkSmart Resource Framework (c) builds upon the OSGi runtime (a), an environment implementing the Open Services Gateway initiative (OSGi) specification for dynamic, service-oriented component systems in Java [8]. The runtime automatically handles the life-cycle, hot deployment and dependency resolution of components.

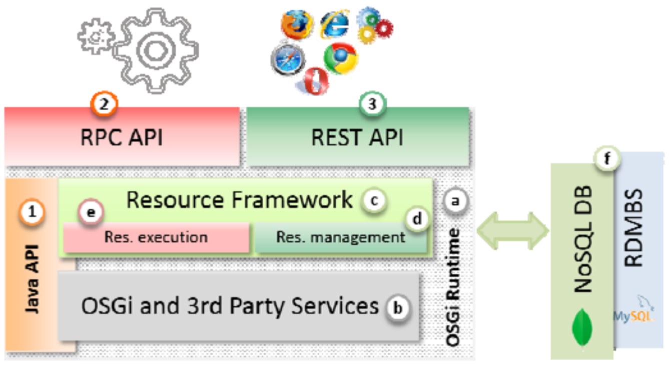

Figure 1. LinkSmart Resource Framework outline

The OSGi *Service Registry* acts as the central integration hub for publication and resolution of services. Every service provided by a component is registered in terms of exposed interfaces and descriptive properties. Clients may filter this public API, obtain and use a service binding to an abstract contract, while the service provider remains transparent. This architecture strongly encourages a separation of interface and implementation bundles and facilitates the creation of extensible loosely-coupled systems.
The LinkSmart Resource Framework relies on a number of standard OSGi and 3rd party services (b), for example the *Service Component Runtime*, *Configuration Admin*, *Event Admin* and *JPA Service*. It further defines a set of services and concepts for technology-agnostic resource management along a data processing pipeline (d). A Java API (1) and remote REST API (3) are defined for this purpose.
It further acknowledges the fact, that some of the managed artifacts may contain interpretable code. State Chart XML [9] for example, is an XML-based formalism to declaratively express control logic. Such documents are per default persisted within a native XML database (f). Given an appropriate execution environment (e), clients might load and instantiate them to become “live” custom services interacted via a native Java or Remote Procedure Call (RPC) interfaces (2). The different request semantics are reflected in the prefix of the remote API address:

Table 1. Remote address mapping

Path	Description
`/resource/...`	Root of RESTful operations, a uniform interface applied to any resource type.
`/service/...`	Root for invocation of RPC-services (via overloaded POST).

Both, the management and execution aspects of resources, as detailed out in the following sections, rely on the concept of a *processing pipeline*. Being themselves services of type `ResourceManagementPipeline` and `ResourceExecutionPipeline` they define the series of steps within the processing of inbound resource requests and the association of handler services to a particular stage.

The appropriate pipeline and handler association is assembled by querying the OSGi service registry via LDAP search filters [8]. They attempt to match the public interfaces and registration properties of the services with properties of the request. Alternative handlers are prioritized according to their `service.ranking` property.

III. *Resource Management Pipeline*

As described in Figure 2 the life-cycle, retrieval and querying of the managed resources is governed by a configurable "data processing pipeline". Clients might exchange textual resource representation via the remote REST interface (a). These are parsed into a lightweight object representation according to the request URL, their media type and further HTTP headers by an appropriate `RepresentationParser` service (1). Equally, on resource retrieval, a `RepresentationSerializer` service is resolved from the OSGi service registry by matching the request properties and the pipeline configuration. While of general purpose, not restricted to remote interactions, these services are compliant with the Entity Provider specification of the Java API for RESTful Web Services (JAX-RS) [11] allowing for a seamless interoperability.

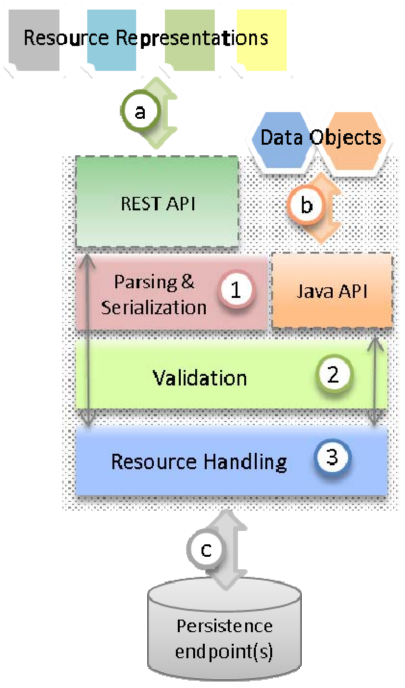

Figure 2. Data Processing Pipeline

Clients may alternatively exchange plain data objects via native Java interfaces (b). In line with our design principles, these objects are considered light-weight binary data representations and a part of the API. They should not contain any application logic beyond accessor and standard methods (`toString`, `equals`, `hashCode`). Regardless of the entry point the client request is unambiguously captured and mapped onto an instance of the `ResourceRequest` class, for example `CreateRequest`, `ListRequest` or `FilterRequest` etc. Within the further step a predefined or the most appropriate `DataValidator` service is selected and applied to check for consistency and validity of the input data (2). Finally a `ResourceHandler` service executes the requested operation, such as creating a new resource instance or synthesizing a representation view of a requested resource. We have translated the concept of inline JAX-RS Resource methods into a layer of fine-granular services implementing a singular resource operation logic. They share the benefits of the standard OSGi ecosystem, are easy to discover and reuse via their explicit public API.

The default processing pipeline might easily be extended by intermediate ETL steps, allowing for data extraction, transformation or merging etc.

A. *Resource Execution Pipeline*

Internally managed programmatic resources might contain interpretable code or queries. Given a secure runtime these resources, once uploaded, could dynamically extend the functional range of the platform. We are currently performing research on requirements and design of such a secure execution sandbox, considering e.g. processual aspects (code review and manual approval by human curator), code inspection and annotation aspects (permission-governed execution). Experimental support is provided for persistent, parametrized data queries that provide a high-level contract and entry point to managed resources. Based on deployed queries clients might create alternative APIs. As described below, our framework uses persistent SPARQL queries [12] for internal management of LinkedData.

IV. METADATA FRAMEWORK

The LinkSmart Metadata Framework (a) as depicted in Figure 3 extends the Resource Framework by capabilities of managing and querying "semantic resources". These are graph-based descriptions of identifiable entities, their attributes and relationships compatible with the RDF data model [13].

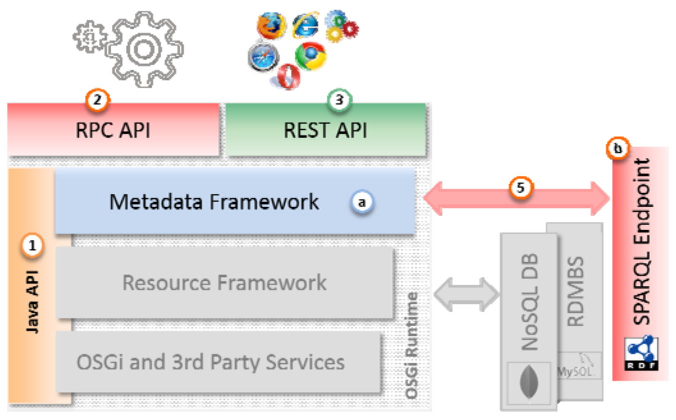

Figure 3. LinkSmart Metadata Framework outline

The framework acts as a transparent proxy to conventional SPARQL endpoints (b) compatible with the SPARQL 1.1 protocol (5) [11]. It considerably augments the typical repository functions and interface coverage by providing native Java interfaces (1) and remote HTTP APIs for execution of SPARQL statements (2) and RESTful data manipulation (3). The underlying data model is defined by the RDF standard [13] – a labeled, directed multi-graph consisting of series of ternary statements (*triples*):

```
<subject><predicate><object>
```

Subject nodes and predicate edges are identified by Unicode-aware Internationalized Resource Identifiers (IRIs) [15], object values are either IRIs or literals. A variety of concrete syntaxes exist to generate an RDF representation, among others the normative RDF 1.1 XML Syntax [15], and the popular JSON-based Serialization for Linked Data (JSON-LD) [17]. A shortened example using the human-readable RDF 1.1 Turtle notation to describe an energy consumption sensor in the GreenCom project is given below:

```
res:2F15BC001D024544 a gc:EnergyConsumptionSensor ;
 rdf:value "8665"^^xsd:double ;
 rdfs:label "Computer + Monitor" ;
 rdfs:comment "Consumption of the PC equipment(Wh)";
 gc:installation  res:85f4d922d87646934.
```

RDF is well suited for tasks of knowledge representation and sharing. Usage of resolvable HTTP URLs for nodes enabled the rise of the Linked Data paradigm, where descriptions of a particular semantic resource might be retrieved and aggregated from distributed repositories such as Linking Open Data cloud[1]. The W3C Linked Data Platform [19] specification recently standardized the HTTP API for management of read-write Linked Data. Nevertheless implementers of this standard are challenged by some peculiarities of the RDF data model.

[1] http://lod-cloud.net/

A. Validation

Unlike other technologies (SQL DDL, XML Schema) RDF is missing a standard schema language for constraint-checking and validation. As appointed in [20] the related RDF Schema [21] and Web Ontology Language [22] standards specify declarative constraints on classes and properties, that may help to infer implicit knowledge, but they are inappropriate to express imperative validity criteria. Due to the open-world assumption that underlies RDF contradicting statements will rarely result in an inconsistent model and thus lead to an error.

There are two levels of RDF validation:

- Validation of the textual representation: considering for example the default RDF/XML syntax, one may test whether the document is well-formed and valid with regards to the specification criteria [16] and an optional, custom XML schema.
- Validation of the abstract model with regards to schema-derived or custom constraints: SPARQL ASK queries are well suited to express the validity criteria by a graph pattern within the WHERE clause to determine the data conformance by programmatic means. The query returns true, for a valid (matching) graph input, false otherwise.

Ryman proposed in [23] a constraints vocabulary to specify the "shape" of RDF resources, i.e. an enumeration of triples the resource is expected to contain and the integrity constraints those triples should satisfy. This work has recently been submitted for standardization as W3C Shapes Constraint Language (SHACL) [24]. We intend to provide a support for SHACL once the specification reached a stable state.

B. Contextualization

The plain RDF data model is missing the concept of a context, a means to link individual triple statements to their originating graph and provenance. Further, since RDF is based on a vocabulary of global identifiers distinguished contexts are needed to delimit scopes in which the asserted triples apply. Programmatic support for context maintenance was introduced by SPARQL named graphs[2] . A set of statements might be asserted to or retrieved from the default graph or a particular named graph within the underlying RDF Dataset. The *Linked Data Platform Container* is the entity for contextualized data management in LDP. It is a surface abstraction that needs to be aligned with the notion of context in RDF Datasets [25] and the various RDF Dataset languages TriG [26] and N-Quads [27]. Potential nesting of LDP Containers (hierarchical arrangement of graphs) needs to be explicitly modelled, for example via a containment relationship.

[2] http://www.w3.org/TR/sparql11-query#namedGraphs

C. Semantic resource handling

In contrast to file-based, document-oriented or relational database systems which share an implicit concept of entity boundaries (like a file, XML root element, JSON root object or SQL table row etc.) there is no simple mean, in graph-based RDF models, to define the boundaries of a particular sub-graph (i.e. a semantic resource). The available options are:

1. Usage of one named graph per resource to provide a context to and consolidate all statements about a single entity. The drawback of this approach is a missing support within the RDF abstract model itself (only available at serialization and manipulation level), high fragmentation of the RDF data set and a difficulty to link and query.

2. Usage of intermediate blank nodes[3] to express boundaries, context, provenance etc. Drawback: this solution would require a proprietary data schema and framework implementing such a blank node traversal and would be incompatible with most of the existing vocabularies.

3. Usage of SPARQL 1.1. Query and Update languages to purposefully construct and manipulate entity sub-graphs. Drawback: development of a management framework and an initial configuration effort to set-up the required queries and updates. Such a programmatic handling of graph fragments may optionally build on top of an additional named graph organization 1).

The latter solution was selected for implementation in our framework, since it employs a standard tool chain and is not limited to a particular data schema. There are generic `ResourceHandler` services provided to handle different types of operations. The default `ReadRequestHandler` for example extracts a flat (one-level) graph representation of an entity:

```
CONSTRUCT { ?resource ?predicate ?object }
WHERE { ?resource ?predicate ?object }
```

The default `ListRequestHandler` retrieves only the annotation properties of resources within the underlying graph:

```
PREFIX rdf: <http://www.w3.org/.../22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/.../rdf-schema#>
CONSTRUCT {
  ?resource rdf:type ?type .
  ?resource rdf:value ?value .
  ?resource rdfs:label ?label .
  ?resource rdfs:comment ?comment .
} WHERE {
  OPTIONAL { ?resource rdf:type ?type }
  OPTIONAL { ?resource rdf:value ?value }
  OPTIONAL { ?resource rdfs:label ?label }
  OPTIONAL { ?resource rdfs:comment ?comment }
}
```

User defined SPARQL resources might be uploaded to the platform and made available for custom resource handling thanks to the above mentioned Execution Pipeline.

V. Linked Data Service Examples

The development of the LinkSmart Metadata Framework was driven and supported by the FP7 EU research projects GreenCom and ALMANAC.

A. GreenCom Installation Management

The SmartGrid project GreenCom[4] aims at supporting demand response operations to stabilize the low-power power grid challenged by consumption peaks and fluctuating renewable energy sources. Wireless sensor and actuation networks for focused monitoring of energy load, environmental conditions and appliance control were deployed in more than 25 houses. Installation details – sensor equipment, its capabilities and associated appliances etc. – are collaboratively maintained within highly structured wiki pages.

Editable webpages represent a natural user interface to manage and perceive textual information. On the other hand, the cloud-based warehouse used for data persistence and evaluation unavoidably duplicates some of the meta-data. The RDF data model is well suited to capture this kind of structured, potentially incomplete data. Its built-in elimination of duplicate values and powerful query capabilities of SPARQL motivated the decision to (semi)automatically map the human-centric textual information onto an integrated RDF data set as depicted in Figure 4. Likewise recent observation values received from the meters are published and seamlessly integrated into the same data set.

An RDF representation of the GreenCom domain not only helped us to homogenize the meta-data and increase its quality. It enabled queries on the structure and characteristics of the considered micro-grids. Given query, for example, selects identifiers of all houses attached to given power line (radial) and their corresponding heat pump consumption sensor:

```
PREFIX rdf:<http://www.w3.org/.../22-rdf-syntax-ns#>
# Ontology definitions, concepts
PREFIX def: <urn:gc:def:>
# Domain resources, individuals
PREFIX res: <urn:gc:res:>
SELECT ?h ?hpcs
WHERE {
  ?h rdf:type def:House; def:location res:idRadial1.
  # Identify heat pump somewhere in respective house
  ?hp rdf:type def:HeatPump;
      def:location* ?h;
      def:property ?hpc .
  # Select sensor observing its consumption
  ?hpc rdf:type def:PowerConsumption .
  ?hpcs def:observes ?hpc .
}
```

[3] http://www.w3.org/TR/rdf11-concepts/#section-blank-nodes

[4] http://www.greencom-project.eu/

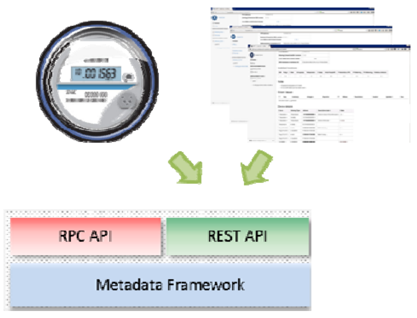

Figure 4. GreenCom metadata integration.

B. *ALMANAC Iot Resource Metadata*

The ALMANAC project[5] focuses on the development of an innovative Smart City Platform (SCP) to collect, aggregate, and analyze sensor data transmitted across dynamically federated networks. A service layer abstracting access to and control of heterogeneous devices along with a Service Development Kit is intended to stimulate proliferation of innovative Smart City Applications. ALMANAC's Semantic Representation Layer component largely builds upon the Metadata Framework and provides RDF descriptions of Smart City resources. Next to usage of established vocabularies new models are being developed to e.g. capture common network capabilities for automated reliability assessment and routing, to semantically annotate data streams and observations. Our demonstrator showcasing demand-driven garbage collection maintains the descriptions of approximately 60.000 waste bins, distributed within the city of Turin. The so called "Driver App" computes an optimal collection route and visualizes garbage containers, that were reported full by evaluating the static resource annotations (bin type, location) and transient measurement values (fill level).

VI. Conclusion

In this paper we presented our initial work on a generic framework for handling of data resources transcending the traditional technology boundaries. Thanks to its protocol-agnostic, uniform interface and a highly extensible service-oriented architecture it is expected to ease the integration of heterogeneous data sources and devices within the IoT context. Our future work will focus on further usability evaluation of this approach with developers, alignment with relevant standards and public release of the code as part of the LinkSmart Middleware [1].

VII. Acknowledgment

The presented research work received funding from the European Community's Seventh Framework Programme (FP7 / 2007-2013) projects GreenCom and ALMANAC under grant agreement n.609081.

[5] www.almanac-project.eu/

VIII. References

[1] LinkSmart, "LinkSmart Middleware," https://www.linksmart.eu, 2015, accessed: October 2015.
[2] JSR 243, "Java Data Objects 3.1", http://svn.apache.org/viewvc/db/jdo-/trunk/specification/OOO/JDO-3.1.pdf?view=co, 2015, accessed: October 2015.
[3] JSR 338, "Java Persistence 2.1", https://www.jcp.org/en/jsr/detail?id=338, 2013, accessed: October 2015.
[4] OpenLink Software, "Virtuoso Universal Server", http://virtuoso.openlinksw.com/, 2015, accessed: October 2015.
[5] Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.
[6] Linked Data, "W3C LinkedData Overview", http://www.w3.org/standards/semanticweb/data, 2015, accessed: October 2015.
[7] LDP, "Linked Data Platform 1.0", http://www.w3.org/TR/ldp/, 2015, accessed: October 2015.
[8] OSGi Alliance, "OSGi Core Release 6 Specification", 2014, https://osgi.org/download/r6/osgi.core-6.0.0.pdf, accessed: October 2015.
[9] SCXML, "State Chart XML (SCXML): State Machine Notation for Control Abstraction", http://www.w3.org/TR/scxml/, 2015, accessed: October 2015.
[10] IETF, "Lightweight Directory Access Protocol (LDAP): String Representation of Search Filters", http://tools.ietf.org/search/rfc4515, 2006, accessed: October 2015.
[11] JSR 339, "JAX-RS 2.0: The Java API for RESTful Web Services", https://jcp.org/en/jsr/detail?id=339, 2014, accessed: October 2015.
[12] SPARQL, "SPARQL 1.1 Query Language", http://www.w3.org/-TR/sparql11-query/, 2013, accessed: October 2015
[13] RDF Model, "RDF 1.1 Concepts and Abstract Syntax", http://www.-w3.org/TR/rdf11-concepts/, 2014, accessed: October 2015.
[14] SPARQL Protocol, "SPARQL 1.1 Protocol", http://www.w3.org/TR-/sparql11-protocol/, 2013, accessed: October 2015.
[15] IRI, "Internationalized Resource Identifiers (IRIs)", https://tools.ietf.org/html/rfc3987, 2005, accessed: October 2015.
[16] RDF XML, "RDF 1.1 XML Syntax", http://www.w3.org/TR/rdf-syntax-grammar/, 2014, accessed: October 2015.
[17] JSON-LD, "JSON-LD 1.0. A JSON-based Serialization for Linked Data", http://www.w3.org/TR/json-ld/, 2014, accessed: October 2015.
[18] Turtle, "RDF 1.1 Turtle. Terse RDF Triple Language", http://www.w-3.org/TR/turtle/, 2014, accessed: October 2015.
[19] LDP, "Linked Data Platform 1.0", http://www.w3.org/TR/ldp/, 2015, accessed: October 2015.
[20] Ryman, Arthur. *Linked Data Interfaces. Define REST API contracts for RDF resource representations*. http://www.ibm.com/developerworks/rational/library/linked-data-oslc-resource-shapes/, 2013, accessed: October 2015.
[21] RDF Schema, "RDF Schema 1.1", http://www.w3.org/TR/rdf-schema/, 2014, accessed: October 2015.
[22] OWL, "OWL 2. Web Ontology Language Document Overview (Second Edition)", http://www.w3.org/TR/owl2-overview/, 2012, accessed: October 2015.
[23] Resource Shape, "Resource Shape 2.0. W3C Member Submission.", http://www.w3.org/Submission/shapes/, 2014, accessed: October 2015.
[24] SHACL, "Shapes Constraint Language" (SHACL), http://www.w3.org/-TR/shacl/, 2015, accessed: October 2015.
[25] Dataset Semantics, "RDF 1.1: On Semantics of RDF Datasets", http://www.w3.org/TR/rdf11-datasets/, 2014, accessed: October 2015.
[26] TriG, "RDF 1.1 TriG. RDF Dataset Language", http://www.w3.org/TR/-trig/, 2014, accessed: October 2015.
[27] N-Quads, "RDF 1.1 N-Quads. A line-based syntax for RDF datasets", http://www.w3.org/TR/n-quads/, 2014, accessed: October 2015.