LinkedDataHub

Declarative, hypermedia-enabled Linked Data application platform

Martynas Jusevičius, martynas@atomgraph.com
Džiugas Tornau, dziugas@atomgraph.com
AtomGraph

1. Introduction

LinkedDataHub[1] is a declarative, low code Linked Data application platform. It aims to dramatically cut application development costs by enabling domain experts with little to none SPARQL knowledge to configure apps in the UI, as opposed to traditional coding for an API (API is also supported).

LinkedDataHub builds on Linked Data Templates (LDT) specification [LDT] which enables read-write RDF Linked Data and supports hypermedia.

We demonstrate how LinkedDataHub uses hypermedia to drive application state in a distributed environment and provides a consistent user experience while doing so.

2. Architecture

LDT specifies the architecture of a read-write Linked Data application backed by a SPARQL service, with operations defined declaratively in an ontology.

LinkedDataHub applies LDT not only in the context of a single application, but also on a higher level of abstraction: applications and services are also defined using RDF and their management is also LDT-based. The applications are divided into 3 types:

- *Context*: top-level application that provides management of applications and services that belong to it
- *End-user application*: second-level application that manages end-user domain dataset
- *Admin application*: attached to every end-user application and manages its administrative dataset which includes user agents, access control, ontology classes and constraints, operations and queries etc.

The pair of end-user and admin applications comprises a *dataspace*. All management actions are accessible via user interface as well as a generic Linked Data API.

LinkedDataHub application can connect to any data source that supports the SPARQL 1.1 Protocol, but it also provides an option to install a default dataset with some built-in containers. Both approaches can be combined using SPARQL federation.

A number of LDT ontologies for different dataset structures are built-in, the default one being one that supports SIOC-based container/item hierarchy.

[1] https://linkeddatahub.com/docs/about

WebID-TLS [WebID] is supported as the primary authentication method. W3C ACL ontology[2] is used to define access control. An *authorization request* mechanism lets authenticated but unauthorized agents request access to application resources, which become authorizations after approval by the application's owner(s).

LinkedDataHub has a built-in read-write Linked Data client with WebID delegation, which enables indirect interactions between applications on the platform and a seamless user experience. For example, a user agent authenticated with `App1` can can navigate to `App2` and modify its contents (given that access has been granted), without ever leaving `App1` and accessing `App2` directly. This also paves way for interesting features such as copying and moving resources between applications.

The XSLT stylesheets that render response data as XHTML web pages belong architecturally to the client component. They are generic enough to render arbitrary RDF, but include support for hypermedia states generated by LinkedDataHub system ontologies, providing a uniform user experience throughout the platform. Hypermedia responses include enough information to render a functional UI, but importing out-of-band metadata from remote Linked Data resources or LDT application ontology results in an improved experience.

Another specification providing similar features as LDT is Linked Data Platform (LDP) [LDP]. LDP functions as a standardized graph store over an RDF dataset, similar to SPARQL Graph Store Protocol, but with different HTTP semantics (which makes it problematic to combine the two). However, LDP is inadequate for a customizable declarative platform such as LinkedDataHub, because its semantics are predefined by the specification and do not allow for application-scoped operation definitions, which LDT does.

3. Hypermedia

Recent research on hypermedia provides lots of prose with vaguely defined terms such as "affordance" and "factor" [Amundsen] as well as vocabularies [Lanthaler] and data examples [Kjernsmo][Taelman], yet offers little semantics.

Verborgh et al. provide a much more rigorous hypermedia definition [Verborgh]. It involves RDF semantics which we argue are orthogonal to the Linked Data/hypermedia semantics.

Linked Data Platform specification unfortunately provides no support for hypermedia.

Since *hypermedia* is the final and most overlooked constraint of the REST architectural style [Fielding], we use the REST version as the canonical definition of the term: it is the "engine of the application state". In other words:

> *[...] the model application is therefore an engine that moves from one state to the next by examining and choosing from among the alternative state transitions in the current set of representations. Not surprisingly, this exactly matches*

[2] https://www.w3.org/wiki/WebAccessControl

the user interface of a hypermedia browser.

As we can see, the essential concept in REST hypermedia is the application *state*:

> *[...] a given representation may indicate the current state of the requested resource, the desired state for the requested resource*
>
> *REST concentrates all of the control state into the representations received in response to interactions*
>
> *The next control state of an application resides in the representation of the first requested resource*

Given that Linked Data representations are RDF graphs, it is pretty clear from the above descriptions that application states need to be part of that graph.

LDT addresses this in a straightforward way: for each interaction, the evaluation method augments the RDF description of the requested Linked Data resource with an RDF state of that interaction. The state is generated by treating request URI query parameters as arguments for LDT template parameters defined in the application ontology, and is expressed as an RDF graph using the LDT vocabulary. For example, given `<container/?offset=20>` request URI, this could be the state graph (base URI and prefixes omitted):

```
<container/?offset=20> c:stateOf
<container/> ;
    dh:next <container/?offset=40> ;
    ldt:arg [
        a aplt:Offset ;
        spl:predicate dh:offset ;
        ldt:paramName "offset" ;
        rdf:value 20
    ] .
```

All hypermedia states relating to the current interaction become RDF resources in the response graph, explicitly connected using properties. The client agent can simply follow them without building request URIs or using any out-of-band information.

Clients can recognize state arguments by their parameter types, and choose to support parameters from system ontologies that are imported by LinkedDataHub applications. For example, support for `aplt:Offset` (as well as `aplt:Limit` etc.) enables container pagination. Depending on the interaction, additional states can be added, such as those pointing to previous/next page, as in the example above, or constructor states that are used to create new resources.

In the cloud-based version of LinkedDataHub, applications reside on the same physical host. However, since they are accessible under distinct base URIs and hypermedia is used as the interaction protocol, the communication between the applications is no different than in a distributed setting.

4. Conclusions

The LDT specification advances web applications from API descriptions to *API definitions*. In addition to that, LDT provides a foundation for application-defined hypermedia protocols, with application states as RDF resources that are globally identifiable and have machine-readable representations, making them available for user agents. These unique properties of the

RDF data model are the key in enabling the LDT design as well as Linked Data in general.

LDT-based client-server architecture is generic and flexible enough to implement arbitrary web applications that have a uniform API and can be used both in a centralized and in a distributed setting. In future work, we will continue to formalize this architecture by extending the LDT specification.

As we demonstrate, hypermedia applications provide a user experience on the level of current mainstream web applications, and enable features that can go far beyond what is possible with other technologies.

LinkedDataHub platform makes hypermedia accessible for web developers and Linked Data accessible for non-programmers and domain experts.

Bibliography

[LDT] Jusevičius, M., 2016. Linked Data Templates. *XML LONDON 2016.*

[WebID] Story, H., WebID-TLS, WebID Authentication over TSL, W3C Editor's Draft, 08 July 2013.

[LDP] Speicher, S., Arwe, J. and Malhotra, A., 2015. Linked data platform 1.0. *W3C Recommendation, February, 26.*

[Amundsen] Amundsen, M., 2011. Hypermedia types. In *REST: From Research to Practice (pp. 93-116)*. Springer, New York, NY.

[Lanthaler] Lanthaler, M. and Gütl, C., 2013. Hydra: A Vocabulary for Hypermedia-Driven Web APIs. *LDOW, 996*.

[Kjernsmo] Kjernsmo, K., 2012, May. The necessity of hypermedia RDF and an approach to achieve it. In *Proceedings of the First Linked APIs workshop at the Ninth Extended Semantic Web Conference.*

[Taelman] Taelman, R. and Verborgh, R., 2017, December. Declaratively Describing Responses of Hypermedia-Driven Web APIs. In *Proceedings of the Knowledge Capture Conference* (p. 34). ACM.

[Verborgh] Verborgh, R., Arndt, D., Van Hoecke, S., De Roo, J., Mels, G., Steiner, T. and Gabarro, J., 2017. The pragmatic proof: Hypermedia API composition and execution. *Theory and Practice of Logic Programming*, *17*(1), pp.1-48.

[Fielding] Fielding, R.T., 2000. REST: architectural styles and the design of network-based software architectures. *Doctoral dissertation, University of California*.