Linked Data Notifications (LDN) and Linked Data Fragments (LDF)

This is a report of the work done on the research and integration of Linked Data Notifications (LDN) and Linked Data Fragments (LDF) in VitroLib.

Linked Data Fragments

Overview

LDF (http://linkeddatafragments.org/) is the concept of having a uniform view on Linked Data (LD) interfaces that can be used towards reliable Web querying. To that end, a client asks a server about the kinds of Linked Data fragments that are available and then dynamically adapts its query plan for the server to execute. A LD fragment is defined by three characteristics: data (what triples does it contain?), metadata (what do we know abut it?), controls (how to access more data?) Triple Pattern Fragments (TPF) (http://www.hydra-cg.com/spec/latest/triple-pattern-fragments/) is one possible way to define a fragment that enable clients to create low-cost queries that can be executed on live data.

Technical Outputs

Testing the Feasibility of Using LDF alongside VitroLib

(See also: https://docs.google.com/document/d/1LHlZuuypc_JNYO3vMzJaiiUquuuYg2xwXUCjRgJAig8/edit)

The FAST (Faceted Application of Subject Terminology) Linked Data (https://www.oclc.org/research/themes/data-science/fast/download.html) based on the Library of Congress Subject Headings (LCSH) was used to test an LDF instance. In order to keep the codebases (VitroLib and LDF Server /Client) separate, as well as for potential reuse of the LDF server outside of the VitroLib interface, the LDF server instance ran independently from VitroLib.

OCLC-FAST (https://github.com/ld4l-labs/oclc-fast) includes a script to fetch each of the FAST facets. It unpacks each archive which contains the data in N-Triples format. It then concatenates all facet files to a single N-Triples file, and converts (compresses) to HDT format - a compact binary format for RDF that can be used by an LDF server - resulting in 24 million triples. There is also a LDF Server configuration for the OCLC-FAST HDT.

An **LDF Server** (https://github.com/LinkedDataFragments/Server.js) was instantiated on the machine at stanley.lib.harvard.edu, and responds to query patterns as described in its TPF metadata for the OCLC-FAST dataset. Tests on the Server were done with LDF Client (https://github.com /LinkedDataFragments/Client.js) using the command-line interface. The response from the server is in RDF using content-negotiation.

Two possible architectures were considered for VitroLib and its UI to use the LDF Server:

1. The front-end UI can talk directly to an LDF Server, and in that case, it acts like an LDF Client. This is typically accomplished through JavaScript. If the LDF Server is at a different Origin, then for it to work properly, it needs to be CORS enabled, and possibly have Access-Control-Allow-Origin set properly. If same origin, it is no problem. The advantage of this is that, the front-end UI is capable of performing its needs (queries/processing), and it can potentially reach to other LDF Servers as it sees fit. This can be asynchronous, and the TPF controls can be more closely aligned with (and possibly more responsive) to what the user does. Possible caching is done in browser’s local storage or given HTTP headers. Data synchronisation and updating VitroLib’s backend data may be more challenging in this setup since the client may have access to data that the VitroLib backend doesn’ t.

blocked URL

2. The front-end UI talks directly with VitroLib server, and in that case, VitroLib’s server-side code acts as an LDF Client - at least partly. VitroLib may or may not do additional data collection from other sources and by other means, eg. RDF store, federated SPARQL queries, through direct local or remote file access. The advantage of this is that, the front-end UI isn't required to do anything beyond passing the input from the interface to VitroLib. It can then process the response that the VitroLib server gives back update the UI. The server could possibly cache the responses it gets from LDF server(s). Synchronisation may be simpler in this setup since the data (at LDF server) is closer to VitroLib backend (including more assurance with access controls).

blocked URL

The VitroLib backend would need to preprocess client's input in either of these options.

At this juncture - once the proof of concept was in place - the group decided that pursuing the full implementation would require further intervention with the broader VitroLib community and LD4L partners. Cornell's work on Questioning Authority was favoured over the LDF integration given that it was further in the implementation process as per querying the data on VitroLib.

Linked Data Notifications

Overview

The LDN protocol is a W3C Recommendation that "describes how servers (receivers) can have messages pushed to them by applications (senders), as well as how other applications (consumers) may retrieve those messages." LDN allows any Web resource (target) to advertise an endpoint (Inbox) to receive messages. The messages are expressed using the RDF language, and can contain any data.

Technical Outputs

Use Cases for Linked Data Notifications in the Library Context

(See also: https://docs.google.com/document/d/124FAYW73AXGVxFL7I1nUtBeLObSge_6o8he2jHdGvUo)

We have first identified and analysed a few familiar use cases in the library context where brief communication takes places between data producers, providers, and consumers. The use cases identified the actors, eg. cataloguer, data provider, and their interactions to create and reuse Linked Data artefacts. Broadly speaking, the use cases described the following interactions: error reporting, incoming and outgoing links, description changes, external description enhancements, and the generation of new information. We have also documented the assumptions and requirements for each use case.

Through this process, we have also identified some features that may be applicable to all actors (senders, receivers, and consumers) based on the following:

* The discovery and reuse of machine-readable constraints on data shapes.

* Authenticated Inboxes to check the credentials of actors.

* Access-control and/or user sender/consumer verification for read-write operations.

* Maintaining a subscriber list or having fixed well-known trusted endpoints

In order to have an initial check on the feasibility of using LDN, we have focused on one use case:

UC3: As a data provider, when I update a graph of a resource known to be linked to by a third party, notify them of changes.

UC3 Requirement: there is a subscription list (where a third-party is listed) for a given resource, or the "known to be linked" is referring to a statement in which the third-party's resource occurs as the subject resource.

For example: 2 collections hold an item of the same Work; one institution adds a subject to their description of the Work; notify the other collection of additional subject assertion.

Implementation

For this kind of interaction to take place, we have decided to use the tooling called maytkso (https://github.com/csarven/mayktso/) to fulfill the role of an LDN Receiver from a potential list of conforming LDN implementations (https://linkedresearch.org/ldn/tests/summary). The choice for the tooling was based on the initial need to experiment with the use case, as well as having a member of this project being the author of the tooling.

Receiver

VitroLib is a Vitro-based editor for library cataloging. Two LDN Receiver instances were setup on a server at stanley.lib.harvard.edu to be available for two VitroLib instances (https://github.com/ld4l-labs/vitrolib): http://stanley.lib.harvard.edu:8080/vitrolib1 and http://stanley.lib.harvard.edu:8080/vitrolib2 . The LDN Receivers were only accessible on the local network of the server. The server was only VPN accessible.

Consumer and Sender

To bypass authentication and authorization layers for the purpose of the tests, notifications were made on a test Inbox located at https://linkedresearch.org/inbox/ld4l/ . This also helped to demonstrate that the implementation can work regardless of the location of the Inbox, ie. not strictly administered by VitroLib itself.

Both LDN Sender and Consumers were implemented in VitroLib (https://github.com/ld4l-labs/vitrolib/tree/feature/listenersExperiment). The LDN Sender in VitroLib is able to discover the Inbox location (above) from a target resource, and send notifications (in JSON-LD) to it. Similarly, the LDN Consumer is able to discover the Inbox, read and parse the notifications in that Inbox. The notifications are subject-centric in that, they contain a small payload were it described an activity about a resource that a VitroLib instance kept in its store. Hence, as per the use case, a VitroLib instance sends a notification to the Inbox of a subscriber's Inbox ie. those interested in being notified about any changes (eg. create, update) to a particular resource. The notifications did not indicate the actual data about the change, only metadata (which resource, when, kind of activity/change). The consuming applications (eg. as in VitroLib) are then able to retrieve these notifications and process them to their needs.

At this time, the Consumer and Sender implementations skip the discovery step of the LDN protocol, where the target, a resource that advertises its Inbox. While the initial experiment used a fixed Inbox URL - as opposed to using any Inbox URL - the plan is to use any Inbox URL once the same discovery mechanism is implemented in VitroLib and usable by both Consumer and Sender components.

GitHub repositories

- https://github.com/ld4l-labs/LDN
- https://github.com/ld4l-labs/oclc-fast