BioHackrXiv

BioHackJP 2023 Report R4: Enabling SPARQL queries over conventional medical records

Eric Prud'hommeaux[1], **Claude Nanjo**[2], **and Jerven Bolleman**[3]

1 Janeiro Digital **2** University of Utah **3** SIB Swiss Institute of Bioinformatics

BioHackathon series:
DBCLS BioHackathon 2023
Kagawa, Japan, 2023
R4

Submitted: 13 Jul 2023

Published by BioHackrXiv.org

Background

HL7 FHIR is a global standard for accessing and sharing electronic medical records in secure environments. In recent years, FHIR has gained broader adoption among health IT and the major Electronic Health Record (EHR) system vendors. FHIR defines a metamodel, a core clinical model, and a resource-oriented RESTful API for querying FHIR repositories. FHIR also defines a path language called FHIR Path that can be used to filter and project the results of FHIR queries. FHIR Data has been available as RDF since release R2, and reached tech maturity level 3 but is not yet normative. FHIR RDF is supported by a number of FHIR platforms and by the HAPI FHIR library.

Motivation

Medical information is often stored within closed proprietary system as semistructured data. Yet, access to medical information in a structured, standard and interoperable format is key to promoting innovation in medicine, science, and health information technology. This is especially apparent in the area of translational medicine where discoveries in biotechnology have the potential to impact medicine in significant ways.

By allowing the retrieval of clinical information using common semantic web technologies such as SPARQL, researchers will be able to use familiar tooling to integrate clinical information with their existing knowledgebases. FHIR offers a method to return clinical data in RDF format. However, it currently does not support SPARQL - a common way to query RDF data. As a result, to support a SPARQL query interface often requires loading all needed data from an existing FHIR Source into a separate SPARQL-capable database. This approach is undesirable for reasons of security and data freshness. SPARQL support allows organizations to provide researchers and health data integrators the capability for analytical queries that are outside of the design sweetspot of FHIR while leveraging their existing health IT infrastructure and security systems.

Goals

Vision

To enable SPARQL support in existing FHIR-enabled clinical systems.

Objectives

- Stand up a SPARQL endpoint that is decoupled from the FHIR endpoint.
- Construct a valid FHIR query from the incoming SPARQL query.
- Execute the FHIR query against a HAPI FHIR endpoint and transform the results into a valid SPARQL result set
- Operate solely against the FHIR search API thus decoupling queries from the persistence layer as various persistence configurations may exist within health systems.

Outcomes

We have implemented a working prototype that allows for SPARQL queries matching the FHIR RDF R5 specification to be answered by a standard SPARQL protocol compliant endpoint. This includes general SPARQL optimisations but also pattern recognition.

The engine chosen for this project is the Apache Jena's ARQ engine. In this engine architecture we only need to replace standard parts of the SPARQL algebra with specific operators that we call HapiOps. HapiOps use the hapi java clients for FHIR calls over REST to talk to a FHIR-enabled data repository. HapiOps are the glue between the standard Jena SPARQL engine evaluation and the FHIR query language. Using the FHIR REST client as the base for our implementation means that this SPARQL engine can use any compliant FHIR data resource.

Given differences in the expressive power of the query languages, the proposed approach only works for a small subset of SPARQL queries at this time. We aimed to use an existing SPARQL engine and its evaluation strategy to ensure correct SPARQL semantics. Doing this will maintain all options for an as efficient as possible retrieval strategy from FHIR-capable datastores, including filter pushdown and constant bindings.

Query manipulation with ArcTrees

The code recognizing which SPARQL fragments can be mapped to a FHIR Path query was derived from the FHIR RDF ShEx data shapes. Originally evaluated in JS this has been translated into Java.

The SPARQL algebra includes triple patterns and path patterns. While technically, intersection of the variables in the subject and object positions express a graph, the data model being queried is a sparsely-interlinked tree. The FHIR queries which allow efficient selection and transformation of remote data are defined as paths from a FHIR Resource's root node. Expressing the SPARQL query as a tree (called an "ArcTree") with some interconnecting variables provides a structure homologous to FHIR Path's tree path expressions. This enables the recognition of all applicable paths, i.e. those that correspond to the SPARQL query and have values that are either constants in the SPARQL query or variables already bound by an earlier join.

Future work

The approach taken during this biohackathon has focused on executing modestly expressive SPARQL queries against a FHIR endpoint in order to demonstrate feasibility and identify potential complexities. However, we recognize that for this approach to be viable, as with any database query, there is a long tail of possible optimizations and refinements. For instance:

- Our approach only works for a small subset of SPARQL queries such as SPARQL queries that clearly specify a single focal resource whose rdf:type is established. We aim to support a broader set of more general queries in the future.
- We have defined a limited number of patterns to map SPARQL query constructs to FHIR. Additional patterns will be added in the future.
- FHIR query results are currently handled in memory and the implementation of result set paging has been deferred.
- The fetching of references has not been implemented at this time (e.g., fetching the subject of a clinical observation such as a Patient resource). While support for fetching referenced resources does not pose a technical challenge for small result sets per se, properly managing this process for paged result sets presents some important challenges.
- SPARQL query processing currently makes use of the Jena library. Yet, accumulating and processing large FHIR result sets (Bundles) in memory can be expensive. To mitigate this

challenge, we could 'push down' SPARQL filters into the where-clauses of the constructed FHIR query. Thus, the conversion of FHIR query result bundles into SPARQL result sets will lead to performance issues for large result sets.

- SPARQL and OWL allow completing FHIR result queries when using coding systems that have logical extensions such as Snomed CT and LOINC. E.g. querying for a more general LOINC code should retrieve all observations coded with a more specific child LOINC code. Efficient implementation of the consequences of OWL derived query rewriting into FHIR Path is an open issue that needs more development efforts.

Given the exploratory nature of this investigation, we anticipate further investigation and significant testing to ensure better coverage over time, both in the expressivity of SPARQL queries against FHIR repositories and in the number of FHIR query patterns that can be supported by such an approach.

Acknowledgements

We would like to thank the fellow participants at BioHackathon 2023 for their collaboration and constructive input, all of which greatly contributed to the success of this effort. We are especially grateful to the organizers at DBCLS for fostering an amazing environment for knowledge sharing and innovation. We would also like to thank the developers of open source platforms such as Jena and HAPI FHIR that have played a key role in the faster adoption of standards and innovation in the healthcare space. Special thanks to our mentors, advisors, and colleagues for their guidance and support.

References

1. HAPI FHIR, https://hapifhir.io.
2. Fast Health Interoperability Resources (FHIR) Standard Specification, Health Level 7, https://fhir.org.
3. Apache Jena, Apache Software Foundation, https://jena.apache.org/