A FHIR-to-RDF converter

Gerhard Kober[1] and Adrian Paschke[2]

[1] Tiani Spirit GmbH, Vienna, Austria
gerhard[DT]kober[AT]tiani-spirit.com
[2] Fraunhofer FOKUS and Freie Universitaet Berlin, Berlin, Germany
adrian[DT]paschke[AT]fokus.fraunhofer.de

Abstract. Clinical medical data can be accessed as HL7 Fast-Health-Care-Interoperability-Resources (FHIR). These resources can be used as the basis for further processing in semantic rule engines. To access them using Semantic Web technologies they need to be transformed to RDF so that they can be queried with SPARQL. This paper proposes an approach to convert FHIR resources to RDF. The implementation provides an RDF interface for accessing FHIR stores with SPARQL using the Apache Jena library.

Keywords: FHIR · RDF · Interoperability

1 Introduction

The medical HL7-standard *Fast Healthcare Interoperability Resources* (FHIR) provides the option to represent FHIR resources as RDF graph [1]. This RDF presentation can be used for finding semantic links and dependencies between multiple resources.
Usually FHIR based systems are used for providing interoperability, and do not provide RDF interfaces. Such an interface is needed to perform e.g. SPARQL queries. Another problem is that FHIR resources are often stored highly distributed.
These issues need to be addressed before further processing the results in semantic rule engines. The challenge is to combine the query outcomes and parse the complete RDF model.
The key component contributed in this paper is a FHIR-to-RDF converter, using existing libraries, combining these to fulfill the needed requirements. By now the limitation is that the client needs to know which type of resource is expected in addition to the query URL. Since the query is not limited, all results from a FIHR store are returned.

2 Related Work

A standard used for medical data is FHIR (abbreviation for Fast Healthcare Interoperability Resources). This HL7 standard describes two core-components [2]:

- Resources: resources are collections of information-models, defining data elements, constraints and relations between these business-objects
- APIs: APIs are collections of well-defined Interfaces to describe the interoperability between applications

FHIR is describing the data structure, the data flow between systems, and the links between different resources. For data exchange FHIR supports the following interoperability paradigms: RESTful API, Messaging, Documents and Services [3].
There are more than hundred resources defined - e.g. *Patient*, *Observation*, *Medication* and FHIR allows the extension and creation of own resources. In this paper the term "FHIR store" is mentioned. This describes the storage container for different FHIR resources, providing an interface for REST calls to insert, update and query FHIR resources.

In [5] the conversion from FHIR to RDF was a manual process, but it is foreseen to translate automatically by using RDF shapes. They proved a conversion from FHIR to RDF is doable. For further automatic processing the online conversion from the FHIR resources to RDF format is missing, in a way that it can be further processed, e.g. by asking SPARQL queries over the result set.
Another paper [6] mentions an approach to store data in RDF format. Since FHIR is meant to store the defined resources "as is", to facilitate interoperability on health care information, having a RDF schema creates an additional overhead for practical medical-clinical use. Secondly this would affect and change the originally sent data, which is used for medical documentation. Another issue is that this solution can not be used for data which already exists, but only for newly generated data. In order to find dependencies in between FHIR resources the conversion to RDF is needed.

3 Proof of concept implementation

The system-architecture for the proof of concept implementation consists of three components (see Fig.1):

1. The client, calling the FHIR-to-RDF converter in order to provide the URL to the FHIR store itself, the FHIR version of the FHIR store and the type of FHIR resource, that is expected to be returned for further processing on the client-side.
2. The FHIR-to-RDF converter: This is a central component in our solution, since it is dispatching the HTTP calls to different servers, and transforming the incoming FHIR bundles to RDF.
3. The FHIR store: this is the place where all the FHIR resources (e.g patient, observations, medications) are persisted.

The proposed system architecture is meant to be a centralized one, because of the following advantages:

- the result of a specific query is always complete (for further processing)

– the approach (implementation) is applicable to every FHIR store, even if it does not provide an RDF interface
– less re-querying is needed: doing a re-query might be necessary, if the client needs more information than provided (this cannot happen if the complete information is available on the client-side)

A disadvantage of the centralized approach might be the processing of big data and the need of data replication in the execution environment if the data cannot be managed in memory.

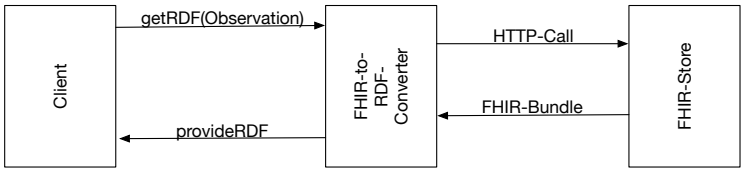

Fig. 1. Workflow to provide RDF to a client-system

The concrete Implementation The implementation is done using Java, with the support of the HAPI-FHIR library and the Apache-Jena library. For the proof-of-concept-implementation the client is a method of the FHIR-to-RDF converter. The client passes the URL, the FHIR version and the FHIR resource-type of the FHIR store to the connector method of the FHIR-to-RDF converter.

The connector method is able to call an FHIR endpoint using HTTP, in order to do queries for a specific sort of FHIR resource. The HAPI-FHIR library provides a RestfulGenericClient in order to connect to different Servers. This library also takes care on different versions of the FHIR standard. There are options for *Dstu2*, *Dstu3*, *R4*.
Next is the search process in the FHIR store: The HAPI-FHIR library also provides methods for searching resources. In this implementation the FHIR resource-types *Observation* and *Patient* are taken into account.
The result of such an invocation is a FHIR bundle. This bundle contains all FHIR resources for a specific type, returned from the server. When we have this result set we need to create out of this a RDF Model. There is also an RDF-Parser included in the HAPI-FHIR library. Before translating the FHIR resource to RDF a valid FHIR resource (containing all the information) is needed. Shown in listening 1.1 there is a reduced sample of such a FHIR resource.

```
{"resourceType": "Observation", "category": [..Vital Signs..],
 "subject": { "reference": "Patient/PatientID1" },
 "effectiveDateTime": "2019-03-31T08:00:00.000",
 "component": [{
  "code": "coding": [... Systolic blood pressure ...],
    "valueQuantity": { "value": 100, "unit": "mmHg",
      "system": "http://unitsofmeasure.org", "code": "mm[Hg
          ]"}},{
```

```
 "code": "coding": [... Diastolic blood pressure ...],
    "valueQuantity": { "value": 80, "unit": "mmHg",
      "system": "http://unitsofmeasure.org", "code": "mm[Hg]"}
  }],"id": "3162fab0-9569-48a1-bec3-1ebaeef1165f"}
```

Listing 1.1. Reduced Observation-Resource

This reduced sample contains the resource type, to which patient this observation belongs to, and the observation itself (in our example the systolic and diastolic blood-pressure). From the FHIR bundle returned by the HAPI-FHIR library (i.e. the Java-object) the resource items are iteratively extracted in order to have a list of resources.
The *convertFhirToRdf*-method takes as input the resource list created before and returns an Apache Jena RDF model. The Apache Jena library is able to create a new model, which is used as a skeleton for the new parsed FHIR resources. Adding resources is done by iterating over the complete list, writing it to a stream, which itself is used as input parameter for the Apache Jena model read method. Listening 1.2 shows the transformation from the FHIR bundle to the RDF model.

```
public Model convertFhirToRdf(List<Resource> res) throws
    IOException {
 RdfParser parser = new RdfParser();
 ByteArrayOutputStream bOut = new ByteArrayOutputStream();
 model = ModelFactory.createDefaultModel();
 for (Resource elem : res) {
  bOut.write(parser.composeString(elem).toString().getBytes());
 }
 model.read(new ByteArrayInputStream(bOutput.toByteArray()),null
     ,"TTL");
 return model;
}
```

Listing 1.2. convertFhirToRdf-Method

4 Evaluation

Performance is a critical point in this solution. There are two steps which are time consuming: The HTTP call to the FHIR store including the response, and the conversion from FHIR to RDF. The first step is depending on the network connection and the overall performance of the FHIR store. In a test-setup with a public FHIR Server ("http://test.fhir.org") this connection and retrieval of the result took in average 6.5 seconds.

Since we used a public test server, the network-delay is pretty high. For simulation of potentially high amounts of FHIR observations, a single result of the FHIR store was copied multiple times and then transformed to RDF. As shown in figure 2, the conversion for 5000 observations to RDF takes about 2.5

seconds - for an amount of 25000 observations 11.4 seconds are needed on a local machine.

When debugging the duration, it was found that a over-all conversion for 76 observations takes about 500ms. This is because of the initialization time of the *Apache Jena class* "RDF model" which takes approximately 300ms and the *HAPI-Class "RdfParser"* which takes about 100ms. The performance is not getting worse if there are more than 10 times more observations in the result, which are needed to be converted, because the initialization of the *Apache Jena class* and *HAPI-RdfParser* is just done once in the routine. As seen in figure 2 there is a linear increase of the needed time for conversion.

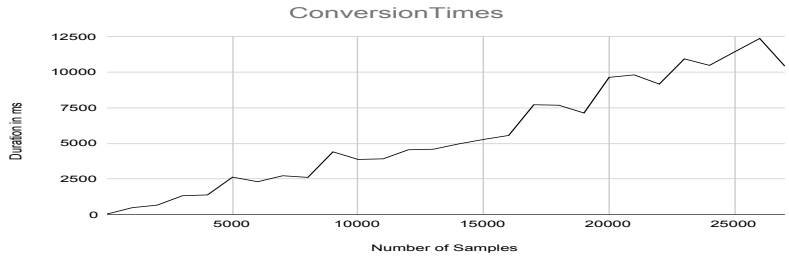

Fig. 2. Duration conversion FHIR Observation to RDF

5 Discussion

The proposed solution has some issues, taking into account that performance might be a bottleneck. The queries on the FHIR store should be specialized to goal in order to get exactly these results back, which are actually needed for further processing. A "query-all" approach is too expensive for a high amount of results. Furthermore, the translation to the Apache-Jena model might be ineffective for a high load of data.

6 Conclusion

This paper has proposed and implemented a FHIR-to-RDF converter, which can be used to convert existing FHIR resources from a FHIR store into RDF. This is done in three steps: first the FHIR resources is fetched via REST, second the FHIR bundle is parsed to an input stream and third an Apache Jena model is provided by reading the input stream into the newly created model. Our converter provides the basis for SPARQL queries on FHIR resources without the need of a FHIR store with an RDF interface. The centralized approach of our conversion tool provides the advantage that not all information is needed in one place.

References

1. FHIR RDF - Representation, https://www.hl7.org/fhir/rdf.html. Last accessed 1 Apr 2019
2. FHIR Overview - Architects, https://www.hl7.org/fhir/overview-arch.html, Last accessed 1 Apr 2019
3. FHIR Overview - Foundation, https://www.hl7.org/fhir/foundation-module.html, Last accessed 1 Apr 2019
4. Bender, Duane and Sartipi, Kamran: HL7 FHIR: An agile and RESTful approach to healthcare information exchange. In: Proceedings of CBMS 2013 - 26th IEEE International Symposium on Computer-Based Medical Systems, pp . 326-331 (2013) https://doi.org/10.1109/CBMS.2013.6627810
5. Martinez-Costa, Schulz: HL7 FHIR: Ontological Reinterpretation of Medication Resources. In: Informatics for Health: Connected Citizen-Led Wellness and Population Health (2017) https://doi.org/doi:10.3233/978-1-61499-753-5-451
6. Sigwele et al: Building a Semantic RESTFul API for Achieving Interoperability between a Pharmacist and a Doctor using JENA and FUSEKI. In: International Conference on Engineering, Technologies, and Applied Sciences (ICETsAS) (2018)