Exploring JSON-LD as an Executable Definition of FHIR RDF to Enable Semantics of FHIR Data

Harold R. Solbrig[1], Dazhi Jiao[1], Eric Prud'hommeaux[2,3], David Booth[4], Cory M. Endle[5], Daniel J. Stone[5], Guoqian Jiang[5]
[1]Johns Hopkins University, Baltimore, MD; [2]Janeiro Digital, Boston, MA; [3]W3C/MIT, Cambridge, MA; [4]Yosemite Project, Somerville, MA; [5]Mayo Clinic, Rochester, MN

Abstract

This study developed and evaluated a JSON-LD 1.1 approach to automate the Resource Description Framework (RDF) serialization and deserialization of Fast Healthcare Interoperability Resources (FHIR) data, in preparation for updating the FHIR RDF standard. We first demonstrated that this JSON-LD 1.1 approach can produce the same output as the current FHIR RDF standard. We then used it to test, document and validate several proposed changes to the FHIR RDF specification, to address usability issues that were uncovered during trial use. This JSON-LD 1.1 approach was found to be effective and more declarative than the existing custom-code-based approach, in converting FHIR data from JSON to RDF and vice versa. This approach should enable future FHIR RDF servers to be implemented and maintained more easily.

Introduction

HL7 Fast Healthcare Interoperability Resources (FHIR)[1] is an emerging open standard for the exchange of electronic healthcare information. The Semantic Web Resource Description Framework (RDF)[2], a W3C standard, has been adopted by the HL7 FHIR as its third interchange format in addition to XML and JavaScript Object Notation (JSON).

To date, the FHIR RDF implementation had been used experimentally in several projects, including fhir.schema.org[3], FHIR in I2B2[4], the FHIR to RDF converter[5], the FHIR/SNOMED integration project[6] and SOLID[7]. This experience has uncovered two difficulties, which this study sought to address:

- Although the FHIR RDF specification[8] is included in the FHIR standard, and has been implemented in a test server, it has yet to be deployed in any of the major FHIR servers such as HAPI[9], Vonk[10], etc. This gap is a significant barrier to wider adoption of FHIR RDF in clinical research informatics communities.
- Several usability issues were discovered in the current FHIR RDF specification, resulting in requests for changes to the FHIR RDF specification.

To address these problems, this study explored the use of JSON for Linked Data, version 1.1 (JSON-LD 1.1)[11] as a means of both documenting and automating conversion of FHIR data to and from RDF.

JSON-LD is a profile of standard JSON that allows JSON-LD data to be interpreted either as plain JSON or as a serialization of RDF. In its role as an RDF format, JSON-LD joins the ranks of Turtle[12] and several other standard W3C RDF formats – one that bridges the worlds of JSON and RDF. Although the FHIR RDF working group[12] previously investigated and rejected the use of JSON-LD 1.0 for FHIR RDF, the 1.0 version lacked critical features that the 1.1 version now includes. In particular, 1.0 did not allow the same FHIR JSON tag, appearing in different FHIR resources, to be mapped to different Internationalized Resource Identifiers (IRIs)[13] in RDF. Since IRIs are used as global identifiers in RDF, this is necessary to allow these properties to be recognized as being semantically distinct. For example, the JSON tag "*name*" currently appears 98 times across 51 different resources in FHIR. JSON-LD 1.1 now enables them the be mapped to distinct IRIs in RDF.

While the existing implementation of FHIR RDF relies on custom code to convert FHIR data to and from RDF, JSON-LD 1.1 enables this conversion to be defined in a more declarative style, through the use of the "@context". This promises to enable easier implementation of future FHIR RDF servers by reducing the amount of custom code that must be written and maintained. It also allows proposed changes to the FHIR RDF specification to be more readily tested and evaluated. To facilitate rapid experimentation and testing, a web-based FHIR JSON-LD

Playground[14, 15] was implemented, based on the existing "JSON-LD Playground"[16], which proved to be enormously helpful. A command-line tool was also created and used to perform batch conversion of FHIR JSON to FHIR RDF. These converters were then used to rapidly test and evaluate proposed changes to FHIR RDF.

Background

RDF is the foundational data representation for “Semantic Web”, "Linked Data", or the "Web of Data", originally envisioned by Tim Berners-Lee and articulated in a feature article of Semantic American in 2001[17]. It allows the semantics of data to be made explicit, based on formal logic, allowing data to be understood both by humans and computers. RDF represents information as a set of assertion, or "triples", which together form a graph (or set of graphs). Wikidata[18] is just one example of a resource in which RDF is used to express a large collection of interrelated knowledge that can be traversed and consumed by both humans and computers.

RDF is particularly well suited both to problems that require automated inference, and to problems that involve integration of data that is expressed in diverse data models and vocabularies. Both of these strengths are important in clinical research informatics, and historically the clinical research informatics and life sciences research communities have been leading adopters of RDF. Biomedical, environmental, pharmaceutical communities have developed sophisticated, RDF frameworks that are being used to understand causal pathways and develop novel clinical treatments[19].

Interestingly, when FHIR RDF was first standardized, the working group assumed that FHIR RDF would mainly be used by the FHIR community. However, it has turned out that most of the interest in FHIR RDF has come from *outside* the FHIR community -- typically from clinical research informaticians who do not know much about FHIR, but want to integrate FHIR data with other data using RDF. In some cases, they also want to produce FHIR data from other data that did not originate in FHIR. This realization has now influenced the FHIR RDF working group to place greater emphasis on making FHIR RDF easier for informaticians who are not FHIR-savvy.

As one example, consider the task of integrating FHIR data with data represented as Phenopackets[20] , an open standard for sharing disease and phenotype information. Phenopackets defines a patient/sample schema for "the phenotypic description of a patient/sample in the context of rare disease, common/complex disease, or cancer.” FHIR has also defined a Patient resource schema, but its attributes are focused on "the demographic information necessary to support the administrative, financial and logistic procedures” related to that patient. These data models were designed independently by different communities for different target use cases, resulting in a mismatch of data models and vocabularies that is typical of RDF data integration efforts. An informatician working wishing to represent Phenopackets data in FHIR RDF would face a significant challenge.

Materials and Methods

Materials

FHIR RDF

In our previous project[21], we created a draft standard for the representation of FHIR in RDF. That draft was accepted by the FHIR RDF working group, and evolved to become the current FHIR RDF standard. The current study uses FHIR RDF standard, with two new goals:

- to investigate the use JSON-LD 1.1 as an improved mechanism for converting FHIR to and from RDF; and
- to use this JSON-LD 1.1 mechanism to test and refine proposed changes to the FHIR RDF standard, to improve its usability for RDF applications.

JSON-LD 1.1

JSON-LD 1.1[11] is used to supply a declarative specification -- using the "@context" mechanism -- for interpreting JSON-LD data as RDF.

JSON-LD Playground

The existing "JSON-LD Playground" open source code was used as a basis for developing a "FHIR JSON-LD Playground" described below.

Methods

Phase 1: : Re-implement the existing FHIR RDF specification (R4) using JSON-LD 1.1

Our first step in this project was to re-implement the existing FHIR RDF specification ("R4"), using JSON-LD 1.1 mapping rules, by creating two FHIR JSON-to-RDF conversion applications:

- an interactive web-based application called the FHIR JSON-LD Playground[14, 15], written in JavaScript, that was developed by forking and extending the existing JSON-LD Playground[16]; and
- a command-line tool[22], written in Python, for batch conversion.

This re-implementation of FHIR RDF (R4) provided a baseline, both to perform regression testing for the JSON-LD 1.1 approach, and to facilitate comparison when proposing changes to FHIR RDF, for the next version ("R5").

The FHIR RDF (R4) specification defines how properties in FHIR JSON must be mapped to RDF triples. This mapping involves two kinds of RDF triples:

- *1:1 triples*: These are RDF triples that correspond directly with JSON properties, in a 1:1 mapping, i.e., each JSON property maps to one RDF triple. These triples could be generated directly from FHIR JSON using the JSON-LD 1.1 @context mechanism.
- *Addtional triples*: These are RDF triples that are not obtainable directly from a 1:1 mapping, but are needed in FHIR RDF either to avoid information loss or to facilitate RDF processing. These triples cannot be generated from FHIR JSON using the @context mechanism alone, because JSON-LD 1.1 still only supports a 1:1 mapping of JSON properties to RDF triples. It does not allow a JSON property to generate more than one RDF triple. For this reason, custom mapping code is still required to generate these additional triples.

For example, the FHIR RDF specification states that a triple supplying a concept IRI should be added to the FHIR Coding[i] element if one can be generated. As another example, the specification also states that JSON list ordering should made explicit by the addition of fhir:index triples in RDF. These examples are illustrated in Figure 1, drawn from a FHIR Observation sample[ii], which shows the JSON representation of a coded concept and the RDF equivalent in Turtle[12].

```
"valueCodeableConcept": {
  "coding": [
    {
      "system": "http://snomed.info/sct",
      "code": "3092008",
      "display": "Staphylococcus aureus"
    }
  ]
},
```

```
fhir:Observation.valueCodeableConcept [
   fhir:CodeableConcept.coding [
     fhir:index 0;
     a sct:3092008;
     fhir:Coding.system [ fhir:value "http://snomed.info/sct" ];
     fhir:Coding.code [ fhir:value "3092008" ];
     fhir:Coding.display [ fhir:value "Staphylococcus aureus" ]
   ]
];
```

Figure 1. The JSON representation of a FHIR CodeableConcept property and the RDF equivalent, in Turtle. The subject of this CodeableConcept property is not shown in these snippets.

Although three of the RDF triples in Figure 1 correspond 1:1 with JSON properties on the left, two additional triples have been inserted to the RDF on the right hand side: one for 'fhir:index 0' and one for 'a sct:3092008'. These triples carry the relative order of this coding (among possibly multiple codings, though this example has only one) and the SNOMED CT IRI for "Staphylococcus aureus" http://snomed.info/id/3092008 (abbreviated as sct:3092008), respectively.

[i] http://hl7.org/fhir/datatypes.html#Coding

[ii] http://hl7.org/fhir/observation-example-f206-staphylococcus.json.html

Since JSON-LD 1.1 @context files can specify *most* of JSON-to-RDF mapping that we need, both of the FHIR JSON-to-RDF converters that we developed leverage this capability by dividing the conversion into two steps:

1. Custom JavaScript or Python pre-processor code takes FHIR JSON data as input, inserts additional JSON properties corresponding to the additional RDF triples that are required (as described above), and produces intermediate JSON-LD 1.1 data.
2. A standard JSON-LD 1.1 processor takes this intermediate JSON-LD 1.1 data as input and, in conjunction with a set of @context files (discussed below), to convert the intermediate data to FHIR RDF.

While the JSON properties in the FHIR JSON input do not correspond 1:1 with triples in the FHIR RDF output (because of the additional RDF triples explained above), the JSON properties in the intermediate JSON-LD 1.1 data *do* correspond 1:1 with triples in the FHIR RDF output.

This two-step approach has two major benefits. First, by using an off-the-shelf JSON-LD 1.1 library to do most of the work in converting JSON to RDF, the amount of custom code required to implement a FHIR JSON-to-RDF converter is reduced. Second, it allows different @context files to be plugged in, to experiment with different conversions.

To produce the @context files required for R4 conversion, we forked the master FHIR build[23] that generates the FHIR specification and added a build step to generate the @context files needed for R4. The FHIR build process is used by the FHIR team to ensure that all FHIR specification artifacts stay in sync as the FHIR specification evolves. Our added step was guided by the existing FHIR Shape Expressions (ShEx) generator that is already included in the FHIR build process, and will eventually be merged back into the official FHIR build process.

Converting FHIR RDF to FHIR JSON

The approach described above leverages @context files to convert FHIR JSON to FHIR RDF, but what about the other direction? Interestingly, these same @context files could also be augmented to perform the reverse conversion -- from FHIR RDF to FHIR JSON – using JSON-LD's "framing" language[24]. We manually tested this Framing approach, and concluded that it would be sufficient to perform this reverse conversion. However, as of this writing we have not yet expanded the build process to emit the necessary framing directives in the generated @context files. This is a topic that we expect to explore further later.

Phase 2: Use the FHIR JSON-to-RDF converters to evaluate proposed revisions to the FHIR RDF specification.

Several usability issues of FHIR RDF R4 were uncovered by early adopters of FHIR RDF. Our goal in Phase 2 was to use the @context-driven converters developed in Phase 1 to demonstrate, test and evaluate proposed solutions to these issues. The issues included:

1. Literal values are nested under Blank Nodes ("BNodes").
2. FHIR References are nested under BNodes.
3. Ordered lists use an extra BNode to include an explicit fhir:index.
4. FHIR extensions are awkward for RDF users.

For example, consider a literal property "status", in a FHIR Observation, as serialized in JSON:

```
"status": "final"
```

Assuming that the subject of this property is <obs123>, one who is not very familiar with the design of FHIR might (wrongly) assume that this property would be equivalently represented in FHIR RDF as the following triple:

```
<obs123> fhir:Observation.status “final”.          # Not FHIR RDF R4!
```

However, in FHIR RDF R4, it is instead represented as *two* triples, using a BNode to connect them

```
:<subj> fhir:Observation.status _:b1 .

_:b1 fhir:value “final” .
```

The intervening BNode requires every RDF query for the Observation status to make one extra step. In SPARQL[25], the required query pattern would be like this, using a property path:

<obs123> fhir:Observation.status / fhir:value ?STATUS .

instead of the simpler:

<obs123> fhir:Observation.status ?STATUS .

The reason for this complication is that FHIR allows nearly everything to be extended -- including literal properties. The benefit of the intervening BNode is that it allows a FHIR extension, such as a status finalization date, to be attached without impacting existing queries. But this flexibility comes at the cost of more complex queries for everyone. In essence, the cost of the extension mechanism is paid by *all* implementers, whether or not they use or care about extensions on this property.

This added complication may seem small in SPARQL, but when an RDF graph is instead manipulated in a typical programming language, using an RDF library, the extra BNode adds more burden -- sometimes multiple lines of additional code. Furthermore, FHIR RDF users are not just consumers of FHIR data but, as mentioned above with the Phenopackets birthdate example, they are also *producers* of FHIR data. The extra BNode also adds a burden on those producers, whether or not they use a FHIR extension. For these reasons, the FHIR RDF working group is re-considering the above and other design choices, to evaluate the trade-offs of proposed alternatives.

Some of these trade-offs were already recognized when the R4 version of FHIR RDF was designed, but an interesting philosophical shift occurred since R4 was designed. The assumption at the time was that FHIR RDF users would already be familiar with the FHIR modeling language and resource models, and would be aware of the intricacies of underlying things like the FHIR *CodeableConcept* and the FHIR extension mechanisms. However, it has since become apparent that most of the interest in FHIR RDF comes from *outside* the FHIR community, especially from informaticians and researchers who want to use FHIR data in conjunction with other (non-FHIR) health or biological data. For this reason, the FHIR RDF working group[26] is now placing greater emphasis on the usability of FHIR RDF by RDF users who are *not* immersed in FHIR. This shift is likely to affect design choices in R5. Anyone interested in FHIR RDF is encouraged to participate in the working group's efforts.

In phase 2, we first validated the correctness of the FHIR JSON-to-RDF converters developed in phase 1, and then we used the converters to rapidly test and evaluate proposed changes to R4, to address these and other issues.

Evaluation design

Before using our two FHIR JSON-to-RDF converters to experiment with potential changes to FHIR RDF R4, we first verified the correctness of the JSON-LD 1.1 @context files that were emitted by our augmented FHIR build process. We verified our R4 @context files in two ways:

- An example-based review was performed, using the FHIR JSON-LD Playground and R4 @context files. Five FHIR JSON examples were taken from the FHIR specification, converted from FHIR JSON to FHIR RDF using the FHIR JSON-LD Playground. The resulting RDF was manually compared with the R4 RDF Turtle files already included in the FHIR specification, and found to be the same. These examples included one for each of the following FHIR resources: Patient, Observation, CodeSystem, Medication, and AllergyIntolerance.

- A batch process-based comparison was performed, using our FHIR JSON-to-RDF command-line converter. Since the FHIR build process already generates FHIR examples in both JSON and RDF Turtle (as well as XML), we were able to use these files as a baseline for comparison against the results produced by our command-line converter, using a python script to automate the comparison. A total of 759 FHIR JSON example files, emitted by the R4 FHIR build process, were found to have corresponding RDF Turtle files also emitted by the R4 FHIR build process. Of these, the majority of them were immediately identical to what our command-line tool produced, but some differences were discovered. After investigation, most of the differences were found to reflect minor gaps in our pre-processor, which inserts additional properties into the intermediate JSON-LD data prior to its conversion to RDF. Interestingly, we also found some differences that uncovered flaws in the FHIR RDF R4 baseline files, as further described below.

After verifying the correctness of the @context files as described, our FHIR JSON-to-RDF converters were evaluated by using them to demonstrate, test and evaluate R5 design alternatives being considered by the FHIR RDF working group. As of this writing, these converters have been actively used for several weeks by members of the FHIR RDF working group, and continue to be used by the group to demonstrate and compare FHIR RDF R5 design alternatives.

Results

FHIR JSON-LD Playground

Our first FHIR JSON-to-RDF converter, the FHIR JSON-LD Playground[14, 15] is an interactive web-based application, written in JavaScript, for converting FHIR JSON to FHIR RDF. It was developed in Phase 1 by forking and extending the existing open source JSON-LD Playground[16], and is shown in Figure 2. Readers can try the FHIR JSON-LD Playground themselves, either by visiting the fhircat.org website or by running the Playground locally, based on the source files.

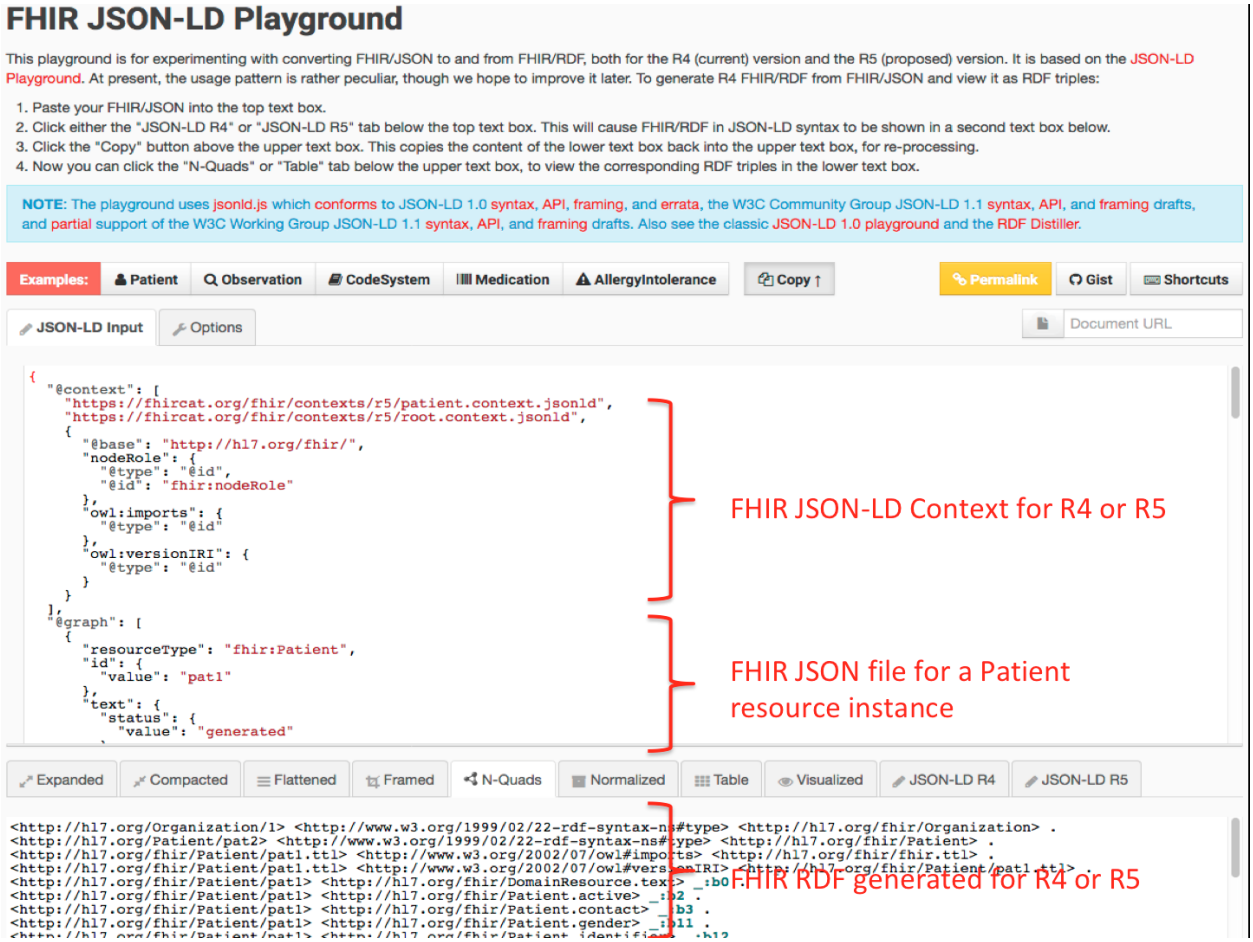

Figure 2. A screenshot of the FHIR JSON-LD Playground with an example for a Patient resource instance.

The following steps demonstrate its use on a sample FHIR Patient record:

1) Click on the *Patient* tab. This loads a sample FHIR JSON Patient resource into the top text box, as input.
2) Click on *JSON-LD R4* in the lower part of the window. This generates intermediate JSON-LD 1.1 data having the structural transformations and additional properties that will be needed in the resulting RDF, and displays that intermediate JSON-LD 1.1 data in the lower text box.
3) Click the *Copy* button near the center of the upper window. This copies the intermediate JSON-LD 1.1 data from the lower text box back to the upper text box, overwriting what was previously in the upper text box, and converts that intermediate JSON-LD 1.1 data to RDF (or other formats) for display in the lower text box.
4) The various tabs in the lower text box (*N-Quads, Table, Visualized*) show different representations of the resulting RDF.

FHIR JSON-to-RDF command-line tool

Our second FHIR JSON-to-RDF converter, the command-line tool, was also developed in Phase 1. It is written as a Python script that invokes the same JSON-LD 1.1 library (written in JavaScript) as is used in the FHIR JSON-LD Playground. It takes FHIR JSON files as input and, driven by a given set of @context files, produces FHIR RDF files.

Evaluation results

The initial validation that we performed on our R4 @context files – by converting 759 FHIR JSON files to FHIR RDF -- turned out to be quite useful in exposing some previously undetected issues in the existing FHIR RDF R4 examples. For example, we discovered that when a FHIR JSON example recursively contained a FHIR "item" nested within another FHIR "item", the corresponding FHIR RDF example (generated by the R4 FHIR build process) used an incorrect IRI: <http://hl7.org/fhir/Questionnaire.**item.item**> instead of <http://hl7.org/fhir/Questionnaire.**item**>.

This validation also helped us identify several small gaps in our pre-processor. For example, FHIR date and dataTime data types must be converted by the pre-processor to the standard XML data types of date, dateTime, gYear or gYearMonth that are used in RDF. These conversions depend on the actual data values being translated from FHIR JSON to FHIR RDF.) Although this complicates the pre-processor, it has little or no impact on the @context files. Another example involves the FHIR *Canonical* datatype, which includes an embedded URL, such as this one for the "instantiates" property.

```
"instantiates": [
   "http://ihe.org/fhir/CapabilityStatement/pixm-client"
 ],
```

In FHIR RDF R4, the URL is duplicated to provide *two* triples – one as a string and one as an RDF node (again, the subject is not shown):

```
fhir:CapabilityStatement.instantiates [
   fhir:value "http://ihe.org/fhir/CapabilityStatement/pixm-client";
   fhir:link <http://ihe.org/fhir/CapabilityStatement/pixm-client>
] .
```

The pre-processor can easily insert an additional fhir:value property as shown, with little or no impact on the @context files. But in this case the pre-processor must already know the datatype of the "CapabilityStatement.instantiates" property, because the datatype is not explicitly indicated anywhere in the data itself. Again, although this adds work for the pre-processor, it has little or no impact on the @context files. Since the fhir:value and fhir:link properties always contain exact same URLs, we might decide to simplify this transformation in the R5 proposal. The validation that we performed was quite helpful in identifying cases like this.

Overall, the most important part of evaluating our FHIR JSON-to-RDF converters involved their daily use in allowing us to easily demonstrate and test proposed changes to the FHIR RDF specification. The FHIR JSON-LD Playground conveniently allows conversions to be tested and viewed interactively, and the command-line converter allows us to quickly generate larger amounts of FHIR RDF data for experimenting with SPARQL queries and other tests. These converters have been – and continue to be -- quite helpful to the FHIR RDF working group.

Discussion and Conclusion

FHIR RDF has the potential to serve as a bidirectional bridge between the FHIR community and external communities that wish to consume or produce FHIR. In order to realize this potential, however, we need to overcome (at least) two significant hurdles, which this work addressed:

1) The cost of RDF implementation *and* maintenance needs to be reduced, making it easier for FHIR Server implementations to support RDF read *and* update services “out of the box”.

2) The existing RDF representation needs to be revised to better serve the needs of those outside of the core FHIR community.

The approach described in this paper can help address both of these hurdles:

- Once JSON-LD 1.1 implementations become widely available -- anticipated by September 2020 -- this approach could substantially reduce the effort required to implement and maintain a FHIR RDF server. FHIR server developers will be able to call vetted, open-source JSON-LD 1.1 libraries to do most of the work in converting FHIR JSON to FHIR RDF, using standard @context files that will be emitted by the FHIR build process. Implementations that use this approach should also be able to incorporate most changes to the FHIR RDF specification simply by pointing to the latest @context library.

- This approach has also proven to be an effective in reducing the effort required to experiment with alternative FHIR RDF representations, because different @context files can be plugged in to achieve different conversions.

- This allows us to rapidly test and evaluate proposed changes to the FHIR RDF specification. It also raises the possibility that we could eventually publish updates and bug-fixes to the FHIR RDF specification that would be immediately realized on any FHIR server, by simply changing the base URL for the JSON-LD @context library.

In summary, JSON-LD 1.1 is an effective aid in helping to automate the RDF serialization and deserialization of FHIR data, and could substantially reduce the cost of future FHIR RDF implementations. In the process, we also discovered that JSON-LD 1.1 also has some interesting potential for mapping between FHIR and other information models.

Acknowledgements

This work is supported by funding from NIH BD2K (U01 HG009450) and FHIRCat (R56 EB028101).

References

1. HL7 Fast Healthcare Interoperability Resources (FHIR). 2020 [January 15, 2020]. Available from: http://hl7.org/fhir/index.html.
2. W3C Resource Description Framework (RDF). 2020 [January 15, 2020]. Available from: http://www.w3.org/RDF/.
3. FHIR Schema.org Extension. 2020 [March 7, 2020]. Available from: https://github.com/lushacao/fhir_to_sdo.
4. Solbrig HR, Hong N, Murphy SN, Jiang G. Automated Population of an i2b2 Clinical Data Warehouse using FHIR. AMIA Annual Symposium proceedings / AMIA Symposium AMIA Symposium. 2018;2018:979-88. PubMed PMID: 30815141; PMCID: PMC6371332.
5. FHIR to RDF Converter. 2020 [March 7, 2020]. Available from: https://github.com/BD2KOnFHIR/fhirtordf.
6. Solbrig HR, Prud'Hommeaux E, Jiang G, editors. Blending FHIR RDF and OWL. SWAT4LS - CEUR Workshop Proceedings; 2017.
7. SOLID Project. 2020 [March 7, 2020]. Available from: https://solid.inrupt.com.
8. FHIR RDF. 2020 [January 15, 2020]. Available from: https://www.hl7.org/fhir/rdf.html.
9. HAPI FHIR Java API. 2020 [March 7, 2020]. Available from: http://hapi.fhir.org/.
10. Vonk FHIR Server. 2020 [March 7, 2020]. Available from: https://fire.ly/products/vonk/.
11. JSON-LD 1.1 Specification. 2020 [March 7, 2020]. Available from: https://w3c.github.io/json-ld-syntax/.
12. RDF 1.1 Turtle. 2016 [March 16, 2020]. Available from: https://www.w3.org/TR/2014/REC-turtle-20140225/.
13. Internationalized Resource Identifier. 2020 [March 16, 2020]. Available from: https://en.wikipedia.org/wiki/Internationalized_Resource_Identifier.
14. FHIR JSON-LD Playground Github Site. 2020 [March 7, 2020]. Available from: https://github.com/fhircat/json-ld.org.
15. FHIR JSON-LD Playground Demo Site. 2020 [March 7, 2020]. Available from: https://fhircat.org/jsonld/playground/.
16. JSON-LD Playground. 2020 [March 16, 2020]. Available from: https://json-ld.org/playground/.
17. Berners-Lee T, Hendler J, Lassila O. The Semantic Web - A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. Sci Am. 2001;284(5):34-+. PubMed PMID: WOS:000168217200023.

18. Wikidata. 2020 [March 7, 2020]. Available from: https://www.wikidata.org/wiki/Wikidata:Main_Page.
19. NCATS Biomedical Data Translator Progrom. 2018 [June 17, 2018]. Available from: https://ncats.nih.gov/translator.
20. Phenopackets Schema. 2020 [March 7, 2020]. Available from: https://phenopackets-schema.readthedocs.io/en/latest/.
21. Solbrig HR, Prud'hommeaux E, Grieve G, McKenzie L, Mandel JC, Sharma DK, Jiang G. Modeling and validating HL7 FHIR profiles using semantic web Shape Expressions (ShEx). J Biomed Inform. 2017;67:90-100. doi: 10.1016/j.jbi.2017.02.009. PubMed PMID: 28213144; PMCID: PMC5502481.
22. JSON-LD 1.1 Processor. 2020 [March 7, 2020]. Available from: https://github.com/fhircat/FHIRJsonLDAmiaPaper.
23. A fork of master FHIR build. 2020 [March 7, 2020]. Available from: https://github.com/fhircat/org.hl7.fhir.core.
24. JSON-LD 1.1 Framing. 2020 [March 16, 2020]. Available from: https://www.w3.org/TR/json-ld11-framing/.
25. SPARQL Query Language for RDF. 2020 [January 15, 2020]. Available from: https://www.w3.org/TR/rdf-sparql-query/.
26. HL7 ITS/W3C HCLS RDF Workgroup. 2020 [January 15, 2020]. Available from: http://wiki.hl7.org/index.php?title=RDF_for_Semantic_Interoperability.