RDF Test Suite: Improving Developer Experience for Building Specification-Compliant Libraries

Ruben Taelman

IDLab, Department of Electronics and Information Systems, Ghent University – imec

Abstract. Guaranteeing compliance of software libraries to certain specifications is crucial for ensuring interoperability within the Semantic Web. While many specifications publish their own declarative test suite for testing compliance, the actual execution of these test suites can add a huge overhead for library developers. In order to remove this burden from these developers, we introduce a JavaScript tool called *RDF Test Suite* that takes care of all the required bookkeeping behind test suite execution. In this report, we discuss the design goals of RDF Test Suite, how it has been implemented, and how it can be used. In practice, this tool is being used in a variety of libraries, and it ensures their specification compliance with minimal overhead.

1. Introduction

One fundamental part of the Linked Data principles [1] is the usage of open Web specifications. Relying on specifications is important for ensuring that Linked Data does not rely only on certain specific tools or services. Instead, anyone should be able to implement these specifications, and that the resulting software should be able to handle Linked Data that may have been produced by other implementations.
When implementing a specification, it is crucial that the resulting implementation is *correct* according to a specification. Concretely, this means that all specified cases must produce the correct result. For example, a parser implementation for the JSON-LD 1.1 syntax specification [2] must produce the expected RDF triples for given JSON-LD documents.
Most Linked Data-related specifications make use of declarative test suites. These test suites are represented in RDF, and are published via the Linked Data principles. Historically, the RDF Data Access Working Group (DAWG) was the first to define such a test suite for the SPARQL query language [3]. The DAWG defined the test manifest vocabulary that allows one or more *tests* to be included in a *test manifest*. Each test defines a certain *action* and a corresponding *result*. In the case of the SPARQL test suite, this action describes a SPARQL query with a dataset, and the result describes a SPARQL query result. In order for implementations to prove their compliance to the SPARQL specification, they need to execute all tests in this test suite. The test results for such test suites are typically represented using the EARL vocabulary [4], which allows test assertions to be represented about whether or not given software passes a certain test. Since this test manifest vocabulary from the DAWG has proven to be a robust basis for representing test manifests, it has been adopted by many other specifications to represent their test suite, such as JSON-LD [2], Turtle [5], and RDFa [6].

Each of these test suites are being maintained by the RDF Tests W3C Community Group *(https://github.com/w3c/rdf-tests)*. An example of such a test suite can be seen in Listing 1.

```
{
  "@context": ["context.jsonld", {"@base": "toRdf-manifest"}],
  "@id": "",
  "@type": "mf:Manifest",
  "name": "Transform JSON-LD to RDF",
  "sequence": [{
    "@id": "#t0001",
    "@type": ["jld:PositiveEvaluationTest", "jld:ToRDFTest"],
    "name": "Plain literal with URIs",
    "purpose": "Tests generation of a triple using full URIs and a
    "input": "toRdf/0001-in.jsonld",
    "expect": "toRdf/0001-out.nq" },
    ...
  ]
}
```

Listing 1: Part of the manifest of the JSON-LD 1.1 parsing test suite.

While the availability of declarative test suites is highly beneficial for ensuring specification compliance, the fact that they are not directly executable can lead to a large overhead for developers. A developer wanting to execute a test suite against an implementing has to go through the following steps:

1. **Download** the test manifest
2. **Parse** the manifest based on a certain RDF syntax
3. **Follow links** to other parts of the manifest if applicable
4. **Interpret** the test definitions as executable logic
5. **Execute** all tests
6. **Report** the test results

Unfortunately, these steps require a significant implementation effort, which adds an additional burden next to the developer's implementation effort of the specification. In order to alleviate this burden, we introduce a JavaScript tool called *RDF Test Suite* that abstracts these steps in a developer-friendly way. This tool allows developers to focus on the part they are really interested in, i.e., implementing a specification. While testing specification compliance is important, it should not require more effort from the developer than calling a single command, which is exactly what RDF Test Suite provides.
In this article, we report on the implementation of RDF Test Suite, how it can be used, and what the future steps are.

2. Implementation

A first design goal of our tool is that it should *easily fit* into the developer's tool chain, and should behave in a manner that is familiar to developers. As such, we designed RDF Test Suite to behave similar to JavaScript testing frameworks such as Jest *(https://jestjs.io/)* and Mocha *(https://mochajs.org/)*. A second design goal is that the tool should be *generic* enough so that it can be used for different types of test suites, instead of being hardcoded to support only one very specific test suite. For this, we opted for a *modular* architecture in which support for spec-specific features can be added without having to change any core logic.
We have implemented RDF Test Suite in TypeScript, which is a typed superset of JavaScript. It is available under the MIT license on GitHub *(https://github.com/rubensworks/rdf-test-suite.js)*. Furthermore, it is available for download on the npm packager *(https://www.npmjs.com/package/rdf-test-suite)*. At the time of writing, it is confirmed to handle the test suites of SPARQL 1.1 Query, RDF/XML, N-Triples, N-Quads, Turtle, Trig, JSON-LD, RDFa, and N3.

3. Usage

In order to use RDF Test Suite on the command line, at least two arguments need to be passed:

1. Path to a JavaScript engine file
2. URL of a test manifest

The JavaScript file must point to an implementation of a specific interface, which depends on the test suite. For example, if syntax parsing tests are being executed, an `IParser` interface is required, while `IQueryEngine` must be passed for query execution tests. The exact implementation is up to the developer, as only the specified interface is relevant to RDF Test Suite. Since RDF Test Suite runs on Node.js, other programs can be invoked from JavaScript via the `child_process` module, which means that non-JavaScript implementations can also be tested with this tool.
By default, *all* tests from the manifest and its dependencies will be executed against the given implementation. Optionally, the user can decide to filter tests if only one or a few tests need to be executed. Since large test suites can require many HTTP requests to be executed, the test suite can optionally be cached. By default test results will be printed in a compact human-readable format, but the results can optionally also be formatted in an EARL report. Additional details on more advanced features can be found in the RDF Test Suite readme *(https://github.com/rubensworks/rdf-test-suite.js#readme)*.
So far, RDF Test Suite is actively being used in more than 15 JavaScript libraries, such as the Comunica SPARQL query engine [7], the parsers of N3.js *(https://github.com/rdfjs/N3.js)*, and a streaming JSON-LD parser *(https://github.com/rubensworks/jsonld-streaming-parser.js)*. Listing 2 shows an example of RDF Test Suite being executed by the streaming JSON-LD parser.

```
$ rdf-test-suite spec/parser.js \
  https://w3c.github.io/json-ld-api/tests/toRdf-manifest.jsonld
...
✔ @import may not be used in an imported context.
✔ @import can only reference a single context
✔ @type: @none is illegal in 1.0.
ℹ A context may not include itself recursively (direct)
✔ Triples including invalid subject IRIs are rejected
✔ Triples including invalid predicate IRIs are rejected
✔ Triples including invalid object IRIs are rejected
✔ 460 / 460 tests succeeded! (skipped 11)
```

Listing 2: Part of the JSON-LD 1.1 parsing test suite results when executed against the streaming JSON-LD parser.

4. Conclusions

The RDF Test Suite has been shown to aid developers in achieving and testing specification compliance, by removing the overhead of test suite interpretation and execution. It is in use by multiple key RDF libraries for JavaScript. Not only does it simplify test suite *execution*, it also simplifies the *development* of new test suites.
In future work, we will add support for more test suites, such as SPARQL 1.1 Update and JSON-LD 1.1 framing. Furthermore, we aim to set up a service around this tool in which test suites can be executed against registered libraries, with the option to publicly display the test results. Such a service would allow developers to register their library for continuous test suite evaluation, and library users to easily choose certain tools based on their test results and specification compliance.

References

1. Berners-Lee, T.: Linked Data. https://www.w3.org/DesignIssues/LinkedData.html (2009).
2. Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., Champin, P.-A., Lindström, N.: SON-LD 1.1–a JSON-based serialization for Linked Data. W3C, https://www.w3.org/TR/json-ld11/ (2020).
3. Harris, S., Seaborne, A., Prud'hommeaux, E.: SPARQL 1.1 Query Language. W3C, https://www.w3.org/TR/2013/REC-sparql11-query-20130321/ (2013).
4. Abou-Zahra, S., Squillace, M.: Evaluation and report language (earl) 1.0 schema. W3C recommendation, W3C. (2006).
5. Beckett, D., Berners-Lee, T., Prud'hommeaux, E., Carothers, G.: RDF 1.1 Turtle. W3C, https://www.w3.org/TR/turtle/ (2014).
6. Herman, I., Adida, B., Sporny, M., Birbeck, M.: RDFa 1.1 Primer - Third Edition. W3C, https://www.w3.org/TR/rdfa-primer/ (2015).
7. Taelman, R., Van Herwegen, J., Vander Sande, M., Verborgh, R.: Comunica: a Modular SPARQL Query Engine for the Web. In: Proceedings of the 17th International Semantic Web Conference (2018).