JSON Information System Validation

Progress Report

Ste Jones
(Liverpool University, GridPP)

July 2019

Contents

Introduction

- We are striving to change the BDII
- One idea is to use JSON, a common structured file format (like XML)
- Only well formed JSON conforming to a proper schema is useful
- It's impossible to write complex JSON by hand
- Hence we'll need tools to automatically validate the JSON in the system

Where I keep the work

- The work is here:

  https://github.com/sjones-hep-ph-liv-ac-uk/json_info_system

- You can download like this:

  git clone https://github.com/sjones-hep-ph-liv-ac-uk/json_info_system

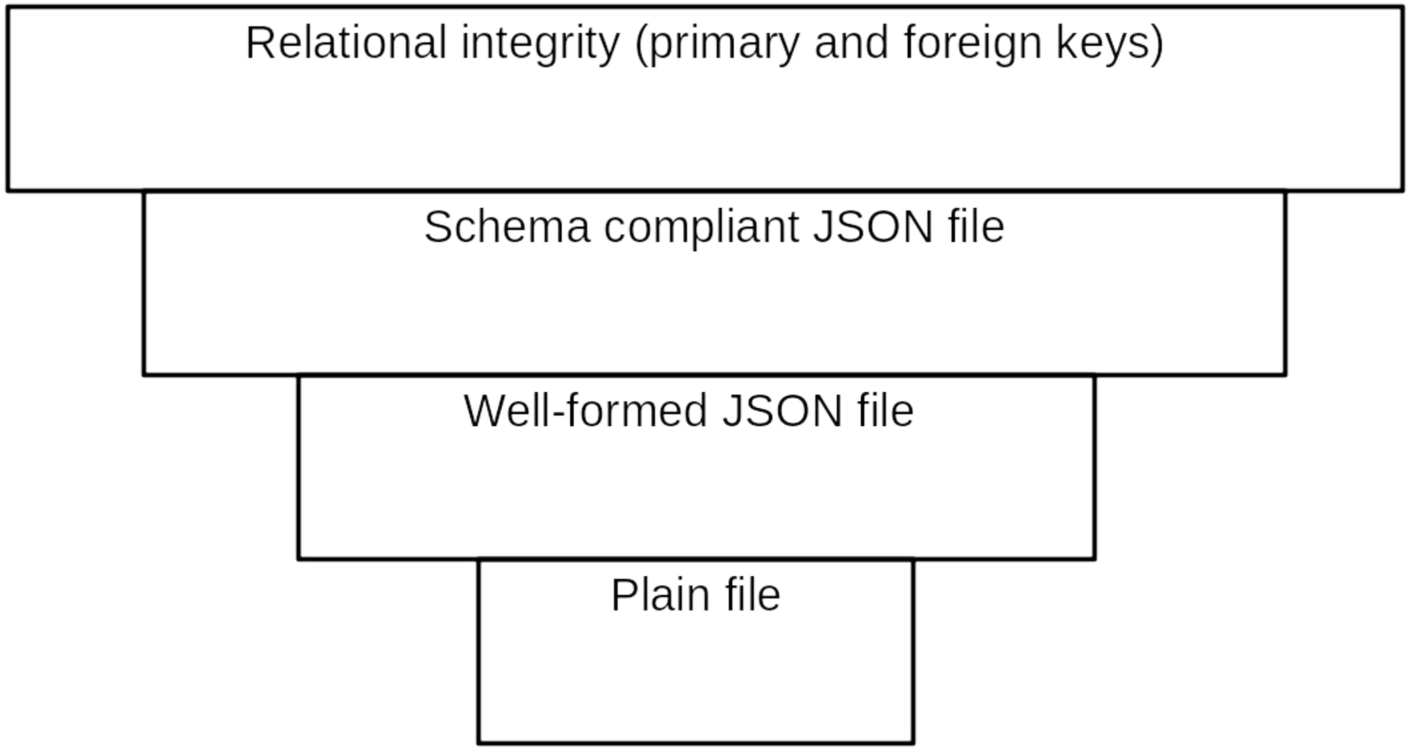
- There are two versions, v1.2 and v1.5, which correspond ~ to versions of this manual schema definition:

  https://docs.google.com/document/d/1t9Hs25B0X7ruhRs4IbdAZ1qDEpBKg2RXBFL9klesGC0

Levels of validation

- JSON must be well formed. It has to comply with the grammar of a JSON file. This is not specific to any application. It’s not so easy to get the grammar correct by hand.
- And JSON must comply with a schema. The schema shows the structure of the document as it relates to a specific application. Schemas can be manual (documented) or automated (using a tool). Automated is better.
- It’s impossible to write well-formed, schema-complaint JSON by hand, hence we’ll need tools.

Levels of validation


Relational integrity (primary and foreign keys)
Schema compliant JSON file
Well-formed JSON file
Plain file

Technology

- There is a standard for JSON Schema.

  https://json-schema.org/
- The newest is draft 7.
- There are many tools that ingest a JSON document and its schema and show if the former complies with the latter.
- I’m using a specific tool called “jsonschema”

  https://pypi.org/project/jsonschema/

Technology

- The jsonschema tool takes a JSON document and its schema.

```
jsonschema -i liv.json ../crr/schema.json
```

- It complains if the JSON document is not well formed.
- If it is well formed, it compares the JSON document to the schema and complains if it does not conform.
- It's the "strong and silent type". If all is well, it says nothing.

The Schema

- Obviously, this needs a schema file. I have written one, in the github:

  json_info_system/v1.5/crr/schema.json

- A section of schema looks like this:

```
  "computingresources": {
    "type": "object",
    "patternProperties": {
      "^.*$": {
        "type": "object",
        "required": ["site", "type", "jobmanager", "jobmanager_version", "status", "number_logical_cpus",
"capacity_hs06", "os", "max_walltime_minutes", "max_memory_gb", "assigned_vos", "publication_time"],
        "properties": {
          "site": { "type": "string", "pattern": "^(.*)$" },
          "type": { "type": "string", "enum": ["batch", "vacuum"] } ,
          "jobmanager": { "type": "string", "enum": ["condor", "torque","VAC","uk.ac.gridpp.vcycle","sge","slurm","lsf"] } ,
          "jobmanager_version": { "type": "string" },
          "status": { "type": "string", "enum": ["development","pre-production","testing","production" ] } ,
          "number_logical_cpus": { "type": "number" },
          "capacity_hs06": { "type": "number" },
```

Limitations

- The JSON Schema technology is pretty good at constraining structure of the documents.
- The fields are typed; structures nested well; everything that must be there must be there; anything that must not be there must not be there; field values can be constrained to an enum list or a regex pattern, etc.
- Unfortunately, there are limitations. Uniqueness is a problem; and relational integrity is another, related problem.

Limitations

- Uniqueness: You can have two entities at the same level with the same name. Problem: identity is lost; one will be overwritten.
- Relational Integrity: Shares in the accesspoints section relate to names in the shares section. Similar with VOs in the resource section versus VOs in the shares section.
- And there is no way to impose these things in JSON Schema, leading to a loss of relational integrity.

The RI Parser

- But we solve all this with a “SAX style” (i.e. event driven) parser.
- Once the JSON file is known to be well formed and to comply with an application schema, it is possible to meaningfully parse the file.
- So I’ve written a python tool to parse a computingresources JSON, and impose the relational integrity required.
- Hence, with the schema and the RI parser taken together, we can keep ~ complete control over the JSON documents, and reject poor ones.

Workflow

- Create a JSON document.
- Validate/edit cycle until it passes schema check.
- Parse/edit until is passes the relational integrity check.
- End result: reliable JSON.
- Obviously if JSON production is done by a computer, then the workflow can be used to test that the tools are good.

Further steps

- Converge on manual schema document(s).
- Update JSON Schema and RI parser to match.
- Schema and RI parser for storage aspects.
- Goal to adopt JSON Schemas and RI parsers as change control element, since they are far less ambiguous than a written document, and written documents tend to drift off.
- Maintain, adopt, promote and document JSONSchema/RIParser workflow on an ongoing basis. There will be new requirements.

Wrap up

- Cheers, end, comments and insults equally welcome, nothing is written in stone ….