

Using JSON Schemas as Metadata Templates in iRODS

June 9, 2020

Venustiano Soancatl Aguilar

Center for Information Technology
University of Groningen, the Netherlands



university of
 groningen

center for
 information technology

Our iRODS Team Groningen

- Simona Stoica
- John Mc Farland
- Andrey Tsyganov
- Aria Babai
- Ger Strikwerda
- Venustiano Soancatl
- Alex Pothar
- Jelmer Builthuis



Json Schema

- Describes your existing data format(s).
- Provides clear human- and machine- readable documentation.
- Validates data which is useful for:
 - Automated testing.
 - Ensuring quality of the data.



Template related tasks (command line approach)

- Define template
- List current templates
- Inspect structure of a template
- Associate template with iRODS objects
- Ingest metadata validated by a template
- Display template metadata



Defining a Json Schema template

Structured string

- Data types
 - String, number, array, objects, Boolean, null
- Nested structures/objects
- Constraints
 - Length, range (min, max), ...



Storing templates

- Elasticsearch,
- Relational database,
- Online repositories
- iRODS AVUs



Listing templates

```
$irule -F list_metadata_templates.r
{
  "hits": [
    {
      "template_id": "yI9yP3EBwqBWH8n46J-b",
      "title": "T2"
    },
    {
      "template_id": "yY9yP3EBwqBWH8n46p_z",
      "title": "T3"
    },
    {
      "template_id": "x49yP3EBwqBWH8n45p9n",
      "title": "T1"
    }
  ],
  "total": 3
}
```



Displaying the structure of a template

```
$irule -F display_template_structure.r 't_uid=' yY9yP3EBwqBWH8n46p_Z' "  
{  
  "title": "T2",  
  "$id": "Unique identifier",  
  "required": [  
    "f"  
  ],  
  "type": "object",  
  "properties": {  
    "e": {  
      "type": "string",  
      "description": "This is attribute e."  
    },  
    "f": {  
      "minimum": 0,  
      "type": "integer",  
      "description": "This is attribute f, must be equal to or greater than zero."  
    }  
  }  
}
```



Associating templates with iRODS objects

Ideally

```
$ itemplate add folder1 upFYQnEBwqBWH8n4E-rS ih rec
```

but

```
$ imeta add -C folder1 MD_TEMPLATES '[{ "t_id":  
"upFYQnEBwqBWH8n4E-rS", "ih": "T", "rec": "T"  }]'
```



Ingesting json metadata validated by json schemas

- Metadata must be in json format
- Metadata must be validated against the associated template
- Metadata must be converted into iRODS AVUs



Converting json metadata to iRODS AVUs

```
{  
  "title": "Hello World!",  
  "parameters": {  
    "size": 42,  
    "readOnly": false  
  },  
  "authors": ["Foo", "Bar"],  
  "references": [  
    {  
      "title": "The Rule Engine",  
      "doi": "1234.5678"  
    }  
  ]  
}
```



Attribute	Value	Unit
title	Hello World!	root_0_s
parameters	o1	root_0_o1
size	42	root_1_n
readOnly	False	root_1_b
authors	Foo	root_0_s#0
authors	Bar	root_0_s#1
references	o2	root_o_o2#0
title	The Rule Engine	root_2_s
doi	1234.5678	root_2_s

Source: <https://irods.org/uploads/2019/vanSchayck-Maastricht-JSON2AVU-slides.pdf>
https://github.com/MaastrichtUniversity/irods_avu_json



Converting json metadata to iRODS AVUs

```
def json2avu(ds, parent):
    # Start without an array index
    index = 0
    out = []
    if isinstance(ds, dict):
        for key, item in ds.items():
            ot = json2avu(item, parent+'.'+key)
            out.extend(ot)
    elif isinstance(ds, list):
        for element in ds:
            lot = json2avu(element, parent+'.'+str(index))
            index = index + 1
            out.extend(lot)
    else:
        out.append([parent, str(ds)])
    return out
```



Converting json metadata to iRODS AVUs

```
json2avu(json_metadata, 'book')
```

```
{  
  "title": "Hello World!",  
  "parameters": {  
    "size": 42,  
    "readOnly": false  
  },  
  "authors": ["Foo", "Bar"],  
  "references": [  
    {  
      "title": "The Rule Engine",  
      "doi": "1234.5678"  
    }  
  ]  
}
```



'book.title'	'Hello World!'
'book.parameters.size'	'42'
'book.parameters.readOnly'	'False'
'book.authors.0'	'Foo'
'book.authors.1'	'Bar'
'book.references.0.title'	'The Rule Engine'
'book.references.0.doi'	'1234.5678'



Ingesting json metadata

Ideally

```
$ itemplate ingest json_metadata object
```

But

```
$ irule -F ingest_json_avus.r
```

```
"*object_path='/rugrdms/home/user/folder1'"
```

```
"*json_path='/rugrdms/home/user/schema_T1_data.json'"
```

```
3 avus ingested successfully
```

```
[[u'T1.a', 'Attribute a'], [u'T1.c', '5'], [u'T1.b', 'Attribute b']]
```



Trying to ingest wrong json metadata

```
irule -F ingest_json_avus.r
```

```
"*object_path='/.../user/folder1/folder1_2/folder1_2_1/mybook.txt'"
```

```
"*json_path='schema_book_data_wrong_title.json'"
```

```
25 is not of type u'string'
```

```
Failed validating u'type' in schema[u'properties'][u'title']:
```

```
  {u'type': u'string'}
```

```
On instance[u'title']:
```

```
  25
```



Displaying metadata

- Consider multiple templates
- Inspect inherited and recursive flags
- Query and store inherited template AVUs

```
irule -F list_object_avus.r
```

```
"*object_path='/rugrdms/home/user/folder1/folder1_1/folder1_1_1'"
```



Displaying metadata

```
irule -F list_object_avus.r
"*object_path='/rugrdms/home/user/folder1/folder1_1/folder1_1_1'"

{
  "vJFYQnEBwqBWH8n4FOr1": {
    "T3.g": "Attribute g",
    "T3.i": "9",
    "T3.h": "Attribute h"
  },
  "upFYQnEBwqBWH8n4E-rS": {
    "T1.c": "5",
    "T1.b": "Attribute b",
    "T1.a": "Attribute a"
  },
  "u5FYQnEBwqBWH8n4FOo_": {
    "T2.d": "Attribute d",
    "T2.e": "Attribute e",
    "T2.f": "7"
  }
}
```



.r, .py and .re files

.r

```
display_template_structure.r  
ingest_json_avus.r  
list_metadata_templates.r  
list_object_avus.r
```



.re

```
list_meta_templates(*templates) {  
}  
display_template_structure(*template_id,*t_structure) {  
}  
ingest_json_avu(*object_path,*json_path,*avus) {  
}  
list_object_avus(*object_path,*avus) {  
}
```



.py

```
def list_md_templates(rule_args,callback, rei):  
def template_structure(rule_args,callback, rei):  
def json2avu(ds, parent):  
def rec_metadata(object_path,level,callback):  
def object_template_metadata(rule_args,callback,  
rei):  
def ingest_AVUs_fromjson(rule_args,callback, rei):
```



.r, .py and .re files

.r

```
display_template_structure.r  
ingest_json_avus.r  
list_metadata_templates.r  
list_object_avus.r
```



.re

```
def list_md_templates(*templates) {  
}  
def display_template_structure(*structure) {  
}  
def ingest_json_avu(*object_path,*json_path,*avus) {  
}  
def list_object_avus(*object_path,*avus) {  
}
```



.py

```
def list_md_templates(rule_args,callback, rei):  
def template_structure(rule_args,callback, rei):  
def json2avu(ds, parent):  
def rec_metadata(object_path,level,callback):  
def object_template_metadata(rule_args,callback,  
rei):  
def ingest_AVUs_fromjson(rule_args,callback, rei):
```

- Microservices, Great!
- icommands, FANTASTIC!!



Building blocks

- Template storage
- Template policies
 - Who can create/remove/modify/share templates?
 - Inheritance
- Template management
 - **microservices**
 - **itemplate [-vVhz] [command]**



Questions/suggestions/comments

Thank you for your attention



university of
 groningen

center for
 information technology