Mobile JSON Work Group Update
OAGi Annual Plenary November 1, 2017

Mike Rowell, Oracle
David Connelly, OAGi
Scott Nieman, Land O’Lakes
Jim Wilson, AgGateway
Serm Kulvatunyou, NIST
Paul Extance, Tapestry Solutions
Agenda

• Review Requirements
• Discuss Vision for Syntax Neutral Approach
• Describe JSON/REST profiling process
  – Nouns as JSON payloads
  – Components as JSON payloads
  – URIs and Verbs
• Highlight future work that needs to be prioritized
Requirement Statements

• OAGIS Semantic Refinement Tool (SRT) needs to be able to produce JSON Schema from the OAGIS Repository DB (syntax neutral goal)
• Production rules for JSON Schema may be at the *noun* level or *component level* (equivalent to base types)
• First validation rule should ensure generated JSON Schemas are valid using the meta-schema and validation keywords
• OAGIS JSON instances should be in compliance with OAGIS JSON Schema produced by SRT and JSON Schema validation rules
• Provide integration tool vendors guidance to support round-trip XML <-> JSON instance transformation with minimal loss at runtime
  – Preserve attributes and namespaces
  – No reserved characters in programming languages
  – XML -> JSON may have loss due to limited set of primitive types in JSON
• URI Resource Paths, filters, and operators should be supported
Semantic Refinement
JSON Schema Production Rules

Model Driven Approach

OAGIS Repository

Syntax Independent
OAG Standard

Production Rule for
JSON Schema

Production Rule for
XML Schema

Production Rule for
OWL/RDF Schema

Cloud and Mobile
Platform

OAG JSON Schema
Standard

Enterprise Integration
Platform

OAG XML Schema
Standard

BPCCS

OAG RDF Schema
Standard

OAGIS Accelerator to
produce consist models
Efforts to Date

• NIST
  – Data Type mapping spreadsheet (XML Schema data type and OAGIS data types to JSON Schema)
  – Review variety of JSON Schema methods to handle XML attributes and namespace
  – List of JSON schema validation tools; characteristics and limitations
  – List of JSON instance validation tools; characteristics and limitations
  – XML Schema vs. JSON schema feature list

• Land O’Lakes
  – Experimented with Swagger.io (Open API) to incorporate multiple generated JSON Schema files into one documentation file
## OAGIS – JSON Data Type Mapping

<table>
<thead>
<tr>
<th>XSD_QIUI tin_TYPE</th>
<th>JSON_QIUI tin_TYPE</th>
<th>JSON QIUI tin TYPE DEFINITION (Based on JSON schema Draft 4)</th>
<th>value space</th>
<th>Possibility of exact value space?</th>
<th>Current JSON lexical space state</th>
<th>Possibility of exact lexical space?</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xs:dateTime</code></td>
<td><code>json:datetime</code></td>
<td><code>&quot;type&quot;: &quot;string&quot;, &quot;pattern&quot;: &quot;[0-9]{4}-[0-9]{2}-[0-9]{2}[T][0-9]{2}[0-9]{2}Z\&quot;&quot;</code></td>
<td>equivalent</td>
<td>restrictive</td>
<td>exact</td>
<td>yes</td>
</tr>
<tr>
<td><code>xs:duration</code></td>
<td><code>json:duration</code></td>
<td><code>&quot;type&quot;: &quot;string&quot;, &quot;pattern&quot;: &quot;\{24}:1([0-9]{2})\&quot;&quot;</code></td>
<td>equivalent</td>
<td>restrictive</td>
<td>exact</td>
<td>yes</td>
</tr>
<tr>
<td><code>xs:duration</code></td>
<td><code>json:duration</code></td>
<td><code>&quot;type&quot;: &quot;string&quot;, &quot;pattern&quot;: &quot;1(([0-9]{2})\&quot;&quot;</code></td>
<td>equivalent</td>
<td>restrictive</td>
<td>exact</td>
<td>yes</td>
</tr>
<tr>
<td><code>xs:day</code></td>
<td><code>json:day</code></td>
<td><code>&quot;type&quot;: &quot;string&quot;, &quot;pattern&quot;: &quot;2([0-9]{2})\&quot;&quot;</code></td>
<td>equivalent</td>
<td>restrictive</td>
<td>exact</td>
<td>yes</td>
</tr>
<tr>
<td><code>xs:time</code></td>
<td><code>json:time</code></td>
<td><code>&quot;type&quot;: &quot;string&quot;, &quot;pattern&quot;: &quot;2([0-9]{2})\&quot;&quot;</code></td>
<td>equivalent</td>
<td>restrictive</td>
<td>exact</td>
<td>yes</td>
</tr>
<tr>
<td><code>xs:date</code></td>
<td><code>json:date</code></td>
<td><code>&quot;type&quot;: &quot;string&quot;, &quot;pattern&quot;: &quot;2([0-9]{2})\&quot;&quot;</code></td>
<td>equivalent</td>
<td>restrictive</td>
<td>exact</td>
<td>yes</td>
</tr>
<tr>
<td><code>xs:datetime</code></td>
<td><code>json:datetime</code></td>
<td><code>&quot;type&quot;: &quot;string&quot;, &quot;pattern&quot;: &quot;2([0-9]{2})\&quot;&quot;</code></td>
<td>equivalent</td>
<td>restrictive</td>
<td>exact</td>
<td>yes</td>
</tr>
<tr>
<td><code>xs:dateTime</code></td>
<td><code>json:datetime</code></td>
<td><code>&quot;type&quot;: &quot;string&quot;, &quot;pattern&quot;: &quot;2([0-9]{2})\&quot;&quot;</code></td>
<td>equivalent</td>
<td>restrictive</td>
<td>exact</td>
<td>yes</td>
</tr>
<tr>
<td><code>xs:time</code></td>
<td><code>json:time</code></td>
<td><code>&quot;type&quot;: &quot;string&quot;, &quot;pattern&quot;: &quot;2([0-9]{2})\&quot;&quot;</code></td>
<td>equivalent</td>
<td>restrictive</td>
<td>exact</td>
<td>yes</td>
</tr>
<tr>
<td><code>xs:time</code></td>
<td><code>json:time</code></td>
<td><code>&quot;type&quot;: &quot;string&quot;, &quot;pattern&quot;: &quot;2([0-9]{2})\&quot;&quot;</code></td>
<td>equivalent</td>
<td>restrictive</td>
<td>exact</td>
<td>yes</td>
</tr>
<tr>
<td><code>xs:time</code></td>
<td><code>json:time</code></td>
<td><code>&quot;type&quot;: &quot;string&quot;, &quot;pattern&quot;: &quot;2([0-9]{2})\&quot;&quot;</code></td>
<td>equivalent</td>
<td>restrictive</td>
<td>exact</td>
<td>yes</td>
</tr>
<tr>
<td><code>xs:time</code></td>
<td><code>json:time</code></td>
<td><code>&quot;type&quot;: &quot;string&quot;, &quot;pattern&quot;: &quot;2([0-9]{2})\&quot;&quot;</code></td>
<td>equivalent</td>
<td>restrictive</td>
<td>exact</td>
<td>yes</td>
</tr>
</tbody>
</table>
## XML Schema to JSON Schema Feature Mapping

<table>
<thead>
<tr>
<th>XML Schema feature</th>
<th>JSON</th>
<th>JSON IMPLEMENTATION</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cardinality</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min=0, max = 1</td>
<td>Optional Element Structure</td>
<td>&quot;[PROPERTY_TERM]:{&quot;type&quot;:[&quot;object&quot;,&quot;string&quot;,&quot;number&quot;,&quot;integer&quot;,&quot;null&quot;,&quot;boolean&quot;]}&quot;</td>
<td></td>
</tr>
<tr>
<td>min=1, max=1</td>
<td>Required Element Structure</td>
<td>&quot;[PROPERTY_TERM]:{&quot;type&quot;:[&quot;object&quot;,&quot;string&quot;,&quot;number&quot;,&quot;integer&quot;,&quot;null&quot;,&quot;boolean&quot;]},&quot;required&quot;:true&quot;</td>
<td></td>
</tr>
<tr>
<td>min=0, max=unbounded</td>
<td>Unbounded Element Structure</td>
<td>&quot;[PROPERTY_TERM]:{&quot;type&quot;:&quot;array&quot;,&quot;items&quot;:{&quot;items&quot;:[]}}&quot;</td>
<td></td>
</tr>
<tr>
<td><strong>Attribute</strong></td>
<td>prefix the property with certain character or characters</td>
<td>&quot;+Customizable-Character[PROPERTY_TERM][&quot;type&quot;:&quot;any&quot;]&quot;</td>
<td>It is very important to realize that id in JSON is not the same as ID in XML. In JSON, you can create a unique identifier for the schema using absolute URL. Then by creating &quot;id&quot; keywords for the elements inside the that specific schema, you can access it. Unlike XML, it is important to note this because purpose of ID in JSON is for referencing.</td>
</tr>
<tr>
<td>id</td>
<td>id</td>
<td>&quot;id&quot;:&quot;URL&quot;</td>
<td>Not sure what to do with this. Don’t know what the we’re trying to do.</td>
</tr>
<tr>
<td>Namespace</td>
<td>Namespace(part of the header)</td>
<td>???</td>
<td></td>
</tr>
<tr>
<td>Nullable</td>
<td>Null</td>
<td>Adding null to the type, e.g., &quot;type&quot;:{&quot;string&quot;,&quot;null&quot;}</td>
<td></td>
</tr>
<tr>
<td><strong>Text</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xml</td>
<td>multipleOf</td>
<td>maximum with exclusiveMaximum=true</td>
<td></td>
</tr>
<tr>
<td>maxExclusive</td>
<td>maximum with exclusiveMaximum=false</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxInclusive</td>
<td>minimum with exclusiveMinimum=true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>minExclusive</td>
<td>minimum with exclusiveMinimum=false</td>
<td></td>
<td></td>
</tr>
<tr>
<td>minOccurs</td>
<td>maximum with exclusiveMinimum=false</td>
<td></td>
<td></td>
</tr>
<tr>
<td>whitespace</td>
<td>minLength</td>
<td></td>
<td></td>
</tr>
<tr>
<td>minLength</td>
<td>maxLength</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxLength</td>
<td>length</td>
<td>same as minLength and maxLength</td>
<td>In JSON, these constraints apply to string only. In XML schema, they apply to many data types. Further investigation will be needed to determine whether the definitions string length are the same for both XML and JSON.</td>
</tr>
</tbody>
</table>
# Review of JSON Schema Validators

<table>
<thead>
<tr>
<th>Name</th>
<th>Language</th>
<th>covered draft</th>
<th>Characteristics</th>
<th>Limitations</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>jsonschema</td>
<td>javascript</td>
<td>draft v4</td>
<td>Schema is ignored, all types supported, all formats supported, allowed to create custom formats, and disable format keywords.</td>
<td>Nested elements inside enum such as one of and anyOf fails, only handles JSON and not javascript object, missing $ref.</td>
<td>MIT</td>
</tr>
<tr>
<td>java-json-tools/json-schema-validator</td>
<td>java</td>
<td>draft v6, draft v8</td>
<td>Pure java, detects $schema and supports $ref. Allows new keyword/format attributes registrations. Use BigDecimal to store decimal numbers instead of double.</td>
<td>No comments are allowed, strings must be wrapped in double quotes, no type inference (&quot;true&quot; is a string not json boolean), id is not trusted by default. URI are fetched only once. Will not accept JSON document larger than 2^31-1 integer.Min_value, Does not support http schema.</td>
<td>LGPLv3 and ASL2.0</td>
</tr>
<tr>
<td>JSON Schema Validator</td>
<td>Java</td>
<td>draft v6</td>
<td>Use this when using org.json API or Gson or java, supports format validators, supports $ref,</td>
<td>Relative $ref against id URL don't work. Currently working on to support draft v6.</td>
<td>Apache 2.0</td>
</tr>
<tr>
<td>eopoberezhkin/ayv</td>
<td>Javascript</td>
<td>draft v4 and v6</td>
<td>Supports draft v6, scores very highly on both speed and standard compliant validator. Validates schema against metaschema. Also support remote ref.</td>
<td>Ajv assigns errors as a property of validation function instead of returning an object with validation results and errors. (This cannot be changed). Fails to validate schemas defined as a $ref by URL. Does not pick up default value if defined in $ref definition.</td>
<td>MIT</td>
</tr>
<tr>
<td>bugventuro/json</td>
<td>javascript</td>
<td>draft v4</td>
<td>Supports few built in formats and allow custom formats. Allow referencing external schema object using $ref. Has built in additional Properties controller.</td>
<td>Broken require field validation when schema contains properties but no sub schema for required property. When there is a validation error for any or one of, then the error path fails to point out exactly where the error is.</td>
<td>MIT</td>
</tr>
</tbody>
</table>
# Review of JSON Instance Tools

<table>
<thead>
<tr>
<th>Name</th>
<th>roundtrippable</th>
<th>Language</th>
<th>Default special characters</th>
<th>Flexibility</th>
<th>Handles multiple namespaces</th>
<th>Able to be used in RESTful sense</th>
<th>characters</th>
<th>Example xml</th>
<th>Example JSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>xml2js</td>
<td>Yes</td>
<td>Javascript</td>
<td>Namespace and attribute = « $, content»</td>
<td>Yes: Able to change attribute key and content key</td>
<td>Yes</td>
<td>Yes, uses node.js</td>
<td>Retains attributes as element but people are able to recognize it.</td>
<td></td>
<td>&lt;book&gt;&lt;title isbn=&quot;13115115&quot;&gt;This book is &lt;emph&gt;bold&lt;/emph&gt;&lt;/title&gt;&lt;/book&gt;</td>
</tr>
<tr>
<td>ObjTree</td>
<td>Yes</td>
<td>Javascript</td>
<td>Handles namespaces as attribute = « $, content»</td>
<td>Yes: Able to change attribute key and content key</td>
<td>Yes</td>
<td>Yes, uses node.js</td>
<td>Able to retain attributes smoothly.</td>
<td></td>
<td>&lt;book&gt;&lt;title isbn=&quot;13115115&quot;&gt;This book is &lt;emph&gt;bold&lt;/emph&gt;&lt;/title&gt;&lt;/book&gt;</td>
</tr>
<tr>
<td>JSON-java</td>
<td>No</td>
<td>Java</td>
<td>N/A</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td></td>
<td></td>
<td>&lt;book&gt;&lt;title isbn=&quot;13115115&quot;&gt;This book is &lt;emph&gt;bold&lt;/emph&gt;&lt;/title&gt;&lt;/book&gt;</td>
</tr>
</tbody>
</table>
Painful process to get from OAGIS XSD to Swagger.io

1. Use GEFEG to create ProcessInspectionOrder profile
2. Use XMLspy to create XML instance
3. Transform XML instance to JSON instance
4. Use XMLspy to create JSON schema off instance
5. Repeat with ShowInspectionOrder
6. Incorporate into Swagger.io
7. Resolve conflicting shared types
8. Add metadata
Profiling and Validation Process
(Proposed for JSON / Mobile)

Provide Context (+BPCCS)
Profile Noun or Component
Define URI Resource Path and HTTP Verb
Select Generate JSON Schema
Validate to Meta-Schema
Share to OAGi Community
Incorporate in Middleware / API Gateway

Swagger
Serm’s nightmare (later iterations)
Goal: Provide Additional Context

• Current: Classification (currently in MSSRT)
  – Geopolitical
  – Industry
  – Standard schemes (e.g., APQC, ISO, SCOR)
    • Process
    • Activities

• Goal: reference to OAGIS Scenarios (UML Sequence diagrams)

• Goal: reference BPMN
  – Introspection H5Ws (who, what, where, when, why)
  – Parsing of the submitted .bpmn file
  – Annotations; related artifacts (TBD)
  – Myriad of associations

(later iterations)
Define URI Resource Path and HTTP Verb

• Review sequence diagrams and/or BPMN
• Resource Path mirror Noun XPATH at times
• Optionally add Business Process Category
• Debate between OData URI standard and grass-roots convention identification still an issue
• OAGIS Action for PATH

(later iterations)
PurchaseOrder Resource Paths

HTTP GET: /MasterData/CustomerParty
HTTP GET: /MasterData/BuyerParty
HTTP GET: /MasterData/BillToParty
HTTP POST:/Procurement/PurchaseOrder/PurchaseOrderHeader/
HTTP GET: /MasterData/Item?Classification.codes.code eq ‘widget’
HTTP PUT: /Procurement/PurchaseOrder/PurchaseOrderLine/
HTTP PUT: /Procurement/PurchaseOrder

(later iterations)
## Verb Mapping: Table and references

<table>
<thead>
<tr>
<th>Method</th>
<th>Scope</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>collection</td>
<td>Retrieve all resources in a collection</td>
</tr>
<tr>
<td>GET</td>
<td>resource</td>
<td>Retrieve a single resource</td>
</tr>
<tr>
<td>HEAD</td>
<td>collection</td>
<td>Retrieve all resources in a collection (header only)</td>
</tr>
<tr>
<td>HEAD</td>
<td>resource</td>
<td>Retrieve a single resource (header only)</td>
</tr>
<tr>
<td>POST</td>
<td>collection</td>
<td>Create a new resource in a collection</td>
</tr>
<tr>
<td>PUT</td>
<td>resource</td>
<td>Update a resource; replacing entire resource</td>
</tr>
<tr>
<td>PATCH</td>
<td>resource</td>
<td>Update a resource by deltas, with action</td>
</tr>
<tr>
<td>DELETE</td>
<td>resource</td>
<td>Delete a resource</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>any</td>
<td>Return available HTTP methods and other options</td>
</tr>
</tbody>
</table>

(later iterations)
Leverage ActionCriteria for HTTP PATCH Verb

"sync": {
  "ActionCriteria": [
  {
    "ActionExpression": [
    {
      "@actionCode": "Change",
      "$": "DataArea.PurchaseOrder.PurchaseOrderHeader"
    },
    {
      "@actionCode": "Change",
      "$": "DataArea.PurchaseOrder.PurchaseOrderLine[1]"
    },
    {
      "@actionCode": "Delete",
      "$": "DataArea.PurchaseOrder.PurchaseOrderLine[2]"
    },
    {
      "@actionCode": "Add",
      "$": "DataArea.PurchaseOrder.PurchaseOrderLine[3]"
    }
  ]
  }
}

✓ Compliant with RFC5789


- We used dot notation for path (more JavaScript)

(later iterations)
Consider adoption of OData Functions

• More useful for traditional GET/SHOW messages where SHOW is a list of business objects; subset of data should be returned

• Used in URL Query Options

• Standard set of functions; OData has defined the following:
  – $value; returns raw value as a string (not JSON)
  – $filter=FirstName eq 'Scott'; uses an operator
  – $orderby=EndsAt desc
  – $top=2; returns first two
  – $skip=18; skips the first eighteen and starts at 19
  – $count; returns a count off
  – $expand; navigation property to get related information (associated table)
  – $select=FirstName, LastName; returns limited set of fields

(later iterations)
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>eq</td>
<td>Equal</td>
<td>/AccountSet?$filter=Address/City eq 'Redmond'</td>
</tr>
<tr>
<td>ne</td>
<td>Not equal</td>
<td>/AccountSet?$filter=Address/City ne null</td>
</tr>
<tr>
<td>gt</td>
<td>Greater than</td>
<td>/AccountSet?$filter=CreditLimit/Value gt 1000</td>
</tr>
<tr>
<td>ge</td>
<td>Greater than or equal</td>
<td>/AccountSet?$&amp;$filter=CreditLimit/Value ge 1000</td>
</tr>
<tr>
<td>Lt</td>
<td>Less than</td>
<td>/AccountSet?$filter=CreditLimit/Value lt 1000</td>
</tr>
<tr>
<td>le</td>
<td>Less than or equal</td>
<td>/AccountSet?$filter=CreditLimit/Value le 1000</td>
</tr>
<tr>
<td>and</td>
<td>Logical and</td>
<td>/AccountSet?$filter=CreditLimit/Value ge 1000 and Address1/StateOrProvince eq 'TX'</td>
</tr>
<tr>
<td>or</td>
<td>Logical or</td>
<td>/AccountSet?$filter=AccountCategoryCode/Value eq 2 or AccountRatingCode/Value eq 1</td>
</tr>
<tr>
<td>not</td>
<td>Logical Negation</td>
<td>/AccountSet?$filter=(AccountCategoryCode/Value ne null) and not (AccountCategoryCode/Value eq 1)</td>
</tr>
</tbody>
</table>
Open API Support (aka Swagger)

- Swagger has growing support
- Now Open API Initiative https://www.openapis.org/
- Metadata about the APIs (note: plural)
- Annotations were removed
- Need to figure out shared type conflicts; these may have been profiled differently between components

(later iterations)
Online Tool Available, or Download

(later iterations)
New Capabilities need to be Prioritized into Sprints

https://triso.tech/94kn8
Incorporate in Middleware/ API Gateway

- JSON Schema can be imported into API Gateway
- Swagger can be imported into API Portal
  - Provides some documentation for developers on usage
  - External developers / trading partners then can subscribe to APIs
  - Developers can obtain security token and incorporate into HTTP Header or URL
  - API can be metered by number of API calls
  - APIs can be monetized if desired
- Gateway validates token on each call to ensure authentication / authorization (two-legged Oauth 2.0)
FUTURE CONSIDERATIONS
Salesforce Composite Resource

- https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/resources_composite_composite.htm

```
"compositeRequest" : [{
  "method" : "POST",
  "url" : "/services/data/v38.0/sobjects/Account",
  "referenceId" : "refAccount",
  "body" : { "Name" : "Sample Account" }
}, {
  "method" : "POST",
  "url" : "/services/data/v38.0/sobjects/Contact",
  "referenceId" : "refContact",
  "body" : {
    "LastName" : "Sample Contact",
    "AccountId" : "@{refAccount.id}"
  }
}]
```
THANK YOU!