Mobile JSON Work Group Update
OAGi Spring Plenary April 12, 2016

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

• Participants
• Relationship to SRT JSON Production Rules
• Describe JSON/REST profiling process
  – URIs and Verbs
  – Nouns and Components as JSON payloads
• Mappings related to JSON <-> XML
• Goal: Update to NDR
Requirement Statements

- OAGIS Semantic Refinement Tool (SRT) must be able to produce JSON Schema from the OAGIS Repository DB
- Production rules for JSON Schema should 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
- XML <-> JSON mapping should enable round trip mapping with minimal loss; i.e., preserves attributes and namespaces
- XML -> JSON may have loss due to limited set of primitive types in JSON
- URI Resource Paths /Navigation should be supported (ref: Odata)
# IETF JSON Schema

- [http://json-schema.org/documentation.html](http://json-schema.org/documentation.html)
- Internet-Draft at the IETF is v5, published 2016-10-13

| JSON Schema Core | Defines JSON Schema core terminology and mechanisms, including pointing to another JSON Schema by reference, dereferencing a JSON Schema reference, specifying the vocabulary being used, and declaring the minimum functionality necessary for processing an instance against a schema. |
| JSON Schema Validation | Defines the list of validation keywords of JSON Schema |
| JSON Hyper-Schema | Describes how JSON Schema can be used to define hyperlinks on instance data |
| Core/Validation Meta-Schema | Used for schemas written for pure validation; e.g, validating a JSON Schema |
| Hyper Meta-Schema | Used for schemas written for validation and hyper-linking. |
Semantic Refinement

JSON Schema Production Rules

Model Driven Approach

OAGIS Repository

Syntax Independent
OAG Standard

Production Rule for
XML Schema

Production Rule for
JSON Schema

Production Rule for
OWL/RDF Schema

Cloud and Mobile
Platform

Enterprise Integration
Platform

OAG JSON Schema
Standard

OAG XML Schema
Standard

BPCCS?

OAG RDF Schema
Standard
Profiling and Validation Process (Proposed for JSON /Mobile)

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

Serm’s nightmare
Provide Context (need to add BPCCS)

- Classification (currently in MSSRT)
  - Geopolitical
  - Industry
  - Standard schemes (e.g., APQC, ISO, SCOR)
    - Process
    - Activities

- OAGIS Scenarios (UML Sequence diagrams)

- BPMN Introspection H5Ws (who, what, where, when, why)
  - Parsing of the submitted .bpmn file
  - Annotations; related artifacts (TBD)
  - Myriad of associations
Define URI Resource Path and HTTP Verb

- Review sequence diagrams and/or BPMN
- Resource Path can mirror Noun XPATH at times
- Resource Paths are case insensitive
- Debate between OData URI standard and grass-roots convention identification still an issue
- HTTP Verb to OAGIS Verbs
- OAGIS Action codes may be needed
PurchaseOrder Resource Paths

HTTP GET: /MasterData/CustomerParty
HTTP GET: /MasterData/BuyerParty
HTTP GET: /MasterData/BillToParty
HTTP POST: /PurchaseOrder/PurchaseOrderHeader/
HTTP GET: /MasterData/Items?Classification.codes.code eq 'widget'
HTTP PUT: /PurchaseOrder/PurchaseOrderLine/
HTTP PUT: /PurchaseOrder
URI Resource Paths

- NDR currently references OData URI naming convention
- Supports complex IDs
  - Bottom-up: /Grower/{ID}/Farm/{ID}/Field/{ID} : great for Oracle DB Sequence or MS Server @Identity columns
    - E.g., PUT /tickets/12/messages/5 - Updates message #5 for ticket #12
  - ERP integration, composite IDs that are a combination of multiple fields; e.g., OrderNumber, OrderType, CompanyCode (JDEdwards) is the unique key for fields
- Need examples in JSON NDR and REST API Design Guide
Salesforce Composite Resource

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

```json
"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}" 
  }
}
}]
```
## 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>

<table>
<thead>
<tr>
<th>OAGIS Verb</th>
<th>REST Verb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process (with Add)</td>
<td>POST</td>
</tr>
<tr>
<td>Process (Replace)</td>
<td>PUT</td>
</tr>
<tr>
<td>Process (Delete)</td>
<td>DELETE</td>
</tr>
<tr>
<td>Process (Change)</td>
<td>No Equivalent</td>
</tr>
<tr>
<td>Sync (with options selected for Add, Change, Delete, or replace)</td>
<td>POST or PUT, depending on who owns the data</td>
</tr>
<tr>
<td>Sync (with Add)</td>
<td>POST</td>
</tr>
<tr>
<td>Sync (Replace)</td>
<td>PUT</td>
</tr>
<tr>
<td>Sync (Delete)</td>
<td>DELETE</td>
</tr>
<tr>
<td>Sync (Change)</td>
<td>No Equivalent</td>
</tr>
<tr>
<td>Post (synonym for Process in financial scenarios)</td>
<td>POST</td>
</tr>
<tr>
<td>Load (Synonym for Sync in financial scenarios)</td>
<td>POST</td>
</tr>
<tr>
<td>Change</td>
<td>PUT</td>
</tr>
<tr>
<td>Update</td>
<td>PATCH</td>
</tr>
<tr>
<td>Cancel</td>
<td>PUT for setting a status or marking for Delete; DELETE physically removes the record via the URI. The data has no meaning. If using URI resource use Delete else if using Data Model use PUT For now PUT</td>
</tr>
<tr>
<td>Get</td>
<td>GET</td>
</tr>
<tr>
<td>Notify</td>
<td>Not applicable. The Notify has Add, Change, Replace, and Delete indicating that it is like the Process/Sync above. Review WebHooks.</td>
</tr>
<tr>
<td>Notify (with Add)</td>
<td>POST</td>
</tr>
<tr>
<td>Notify (Replace)</td>
<td>PUT</td>
</tr>
<tr>
<td>Notify (Delete)</td>
<td>DELETE</td>
</tr>
<tr>
<td>Notify (Change)</td>
<td>PATCH</td>
</tr>
</tbody>
</table>
HTTP PATCH VERB support lacking

- SR 3-14590923001 : Is there a PATCH for the HTTP PATCH verb?
- Bug [23333915](http://williamdurand.fr/2014/02/14/please-do-not-patch-like-an-idiot/) is logged for enhancement request. It has not been implemented.
- Usage is debatable:
PATCH Verb – HTTP vs JSON PATCH


PATCH /file.txt HTTP/1.1
Host: www.example.com
Content-Type: application/example
If-Match: "e0023aa4e"
Content-Length: 100

[description of changes]


PATCH /my/data HTTP/1.1
Host: example.org
Content-Length: 326
Content-Type: application/json-patch+json
If-Match: "abc123"

[{
  "op": "test", "path": "/a/b/c", "value": "foo" },
  "op": "remove", "path": "/a/b/c" },
  { "op": "add", "path": "/a/b/c", "value": [ "foo", "bar" ] },
  { "op": "replace", "path": "/a/b/c", "value": 42 },
  { "op": "move", "from": "/a/b/c", "path": "/a/b/d" },
  { "op": "copy", "from": "/a/b/d", "path": "/a/b/e" }
]

Bottom-line: OAGIS Action Codes needed
Salesforce-> JDE Integration: PATCH and ActionCriteria for SyncPurchaseOrder

```
"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]"
      } 
   ] } 

• Not ideal: uses @ and $

✓ We can claim compliance with RFC5789


• We used dot notation for path (more JavaScript)
```
Query Options

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

• 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
## Operators

<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>
Profile Noun or Component

• Choice depends on the context
• Message between Pools is likely a noun
• DataObject within lane may be a component, a measurement from a sensor
• DataObject may be output from one activity and referenced by another activity later in the business process
Review: PurchaseOrder Objects

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

• GET returns JSON component profile
• POST sends JSON noun profile
• PUT sends JSON noun profile
Case notation for Fields

- snake_case = Ruby, python, popular for JSON APIs
- UpperCamelCase = OAGIS
- camelCase = Java, C#, C++, JavaScript

- snake_case increases payload size, and contradicts the use of JSON vs XML
- Benefit of readable are debatable
Granular OAGIS JSON snippets

- Resource Paths are mappable to XPATH Statements
- POST /PurchaseOrder/PurchaseOrderLine
- Ideally, the SR DB allows creation of these snippets and resource path specifications; another set of production rules
OAGIS JSON production rules should not include XSD primitive types (snippet from 10.1)

"getPriceList" : 
   {"xsi:schemaLocation" : "http://www.openapplications.org/oagis/10
   ../..//Standalone/Local/BODs/GetPriceList.xsd",
   "releaseID" : "xsd:normalizedString",
   "versionID" : "xsd:normalizedString",
   "systemEnvironmentCode" : "Production",
   "languageCode" : "xsd:normalizedString"}
Round-trippable Transformations

XML:
<book>
<title
isbn="15115115">This book is 
<emph>b</emph></title>
</book>

JSON:
{
  "book": {
    "children": [
      {
        "title": {
          "attributes": {
            "isbn": "15115115"
          }
        },
        "children": [
          "This book is ",
          {
            "emph": "b"
          }
        ]
      }
    ]
  }
}

Similar approach for XML namespaces
Select Generate JSON Schema

- Option for SRT user
- JSON Schema would be stored in SRT or File system
- Downloadable for incorporation into Middleware tools
- Referenceable via URI at design time and runtime in middleware tools
- Validated by SRT application using JSON Schema parser
JSON Problem Statements

- JSON Schema support is not 100%; Swagger and RAML are subsets
- JSON Schema is need for instance validation in many API Gateways
- No clear standard for XML to JSON transformation at an instance level
- Balancing act of Friendly vs Round-trippability is critical
- Finding a decent JSON Schema ‘parser’ for validating…Apache Arvo, Node.js
- Validating a JSON Schema is different than validating an instance; the JSON meta schema
Validate to Meta-Schema

- Once SRT application exports JSON Schema, the validator is invoked
- User is presented with results to ensure confidence
- Exceptions reported to support and / or development
- Appropriately logged in log4j
- Harvested in sumologic

https://github.com/json-schema-org/JSON-Schema-Test-Suite
https://github.com/json-schema-org/json-schema-org.github.io
Meta-schema

```json
{    "id": "http://json-schema.org/draft-04/schema#",    "$schema": "http://json-schema.org/draft-04/schema#",    "description": "Core schema meta-schema",    "definitions": {        "schemaArray": {            "type": "array",            "minItems": 1,            "items": { "$ref": "#" }        },        "positiveInteger": {            "type": "integer",            "Minimum": 0        },        "positiveIntegerDefault0": {            "allOf": [                { "$ref": "#/definitions/positiveInteger" }, { "default": 0 }            ]        },        "simpleTypes": {            "type": "array",            "items": { "type": "string" }        },        "stringArray": {            "type": "array",            "items": { "type": "string" },            "minItems": 1,            "uniqueItems": true        },        "boolean": {            "default": false        },        "integer": {            "type": "number",            "MultipleOf": 0,            "ExclusiveMaximum": true        },        "number": {            "Maximum": {            "type": "number"        },        "exclusiveMaximum": {            "true": false        },        "length": {            "type": "integer",            "MinLength": 0,            "MaxLength": 0        },        "additionalItems": {            "anyOf": [                { "type": "boolean" },                { "$ref": "#" }            ]        },        "items": {            "anyOf": [                { "$ref": "#" },                { "$ref": "#/definitions/schemaArray" }            ]        },        "maxItems": { "$ref": "#/definitions/positiveInteger" },        "minItems": { "$ref": "#/definitions/positiveIntegerDefault0" },        "uniqueItems": {            "type": "boolean",            "default": false        },        "maxProperties": {            "$ref": "#/definitions/positiveInteger" },        "minProperties": {            "$ref": "#/definitions/positiveIntegerDefault0" },        "required": {            "$ref": "#/definitions/schemaArray" },        "additionalProperties": {            "anyOf": [                { "$ref": "#" }            ]        },        "dependencies": {            "anyOf": [                { "$ref": "#" },                { "$ref": "#/definitions/schemaArray" }            ]        },        "enum": {            "type": "array",            "minItems": 1,            "uniqueItems": true        },        "type": [            "array",            { "$ref": "#/definitions/simpleTypes" },            { "minItems": 1,            "uniqueItems": true }        ]    }    }    }"```
Share to OAGi Community

• Allow all profiles and resulting exports to be stored and referenced
• Allow sharing to others
• Provide contextual information with profile to enable relationship to BPCCS content
Incorporate in Middleware/ API Gateway

- JSON Schema can be imported into API Gateway
- Current JSON requires pass-thru or conversion to XML
- Lack of Industry standard for round-trip mapping (JSON<-> XML)
- Some are introducing ‘reserved characters’ causing interoperability problems between programming languages/ middleware tools (#, $)
- Some middleware tools are XML internal processing
- Some newer middleware tools are JSON internal processing
- Native JavaScript clearly handles this, but not a rapid development tool
Which should be our approach?

**Approach 1**
- JSON to XML
- Focus on friendliness

**Approach 2**
- JSON to XML
- Focus on round-tripping

**Approach 3**
- XML to JSON
- Focus on friendliness

**Approach 4**
- XML to JSON
- Focus on round-tripping

Ref: IBM paper
### Round-trip vs Friendly Pro/Con Analysis

<table>
<thead>
<tr>
<th>Approach</th>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
</table>
| Focus on Round Tripping | • XML-centric middleware support  
• Addresses interoperability | • Criticized by the JSON developers that prefer bottom-up  
• Not isomorphic as JSON has few primitive types  
• snake_case support is challenging to generate |
| Focus on Friendliness   | • Attracts core developer  
• Still can have validation | • Criticized by top-down, standards developers  
• More challenging to interoperate, except in simpler cases |

*Needs to be finalize and included in report*
## XSD to JSON type mapping

<table>
<thead>
<tr>
<th>XML</th>
<th>JSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>double-precision floating-point format in JavaScript</td>
</tr>
<tr>
<td>xsd:string</td>
<td>String</td>
</tr>
<tr>
<td>xsd:normalizestring</td>
<td>Boolean</td>
</tr>
<tr>
<td></td>
<td>Array</td>
</tr>
<tr>
<td>Cardinality by types</td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td>Object</td>
</tr>
<tr>
<td></td>
<td>Whitespace</td>
</tr>
<tr>
<td>Nillable</td>
<td>null</td>
</tr>
</tbody>
</table>
# OAGIS data types to JSON type mapping

<table>
<thead>
<tr>
<th>OAGIS types</th>
<th>JSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>double-precision floating-point format in JavaScript</td>
</tr>
<tr>
<td>String</td>
<td>double-quoted Unicode with backslash escaping</td>
</tr>
<tr>
<td>Boolean</td>
<td>true or false</td>
</tr>
<tr>
<td>Array</td>
<td>an ordered sequence of values</td>
</tr>
<tr>
<td>Value</td>
<td>it can be a string, a number, true or false, null etc</td>
</tr>
<tr>
<td>Object</td>
<td>an unordered collection of key:value pairs</td>
</tr>
<tr>
<td>Whitespace</td>
<td>can be used between any pair of tokens</td>
</tr>
<tr>
<td>null</td>
<td>empty</td>
</tr>
</tbody>
</table>
Some XML <-> JSON Mapping references

- https://www.xml.com/pub/a/2006/05/31/converting-between-xml-and-json.html
- http://goessner.net/articles/xmljson/
- https://www.p6r.com/articles/2010/04/05/xml-to-json-and-back/
- http://badgerfish.ning.com/
Friendly JSON – one developer’s view


GET
/tickets?fields=id,subject,customer_name,updated_at&state=open&sort=-updated_at

Single endpoint: dynamic selection
THANK YOU!