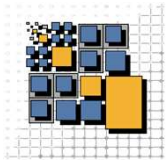
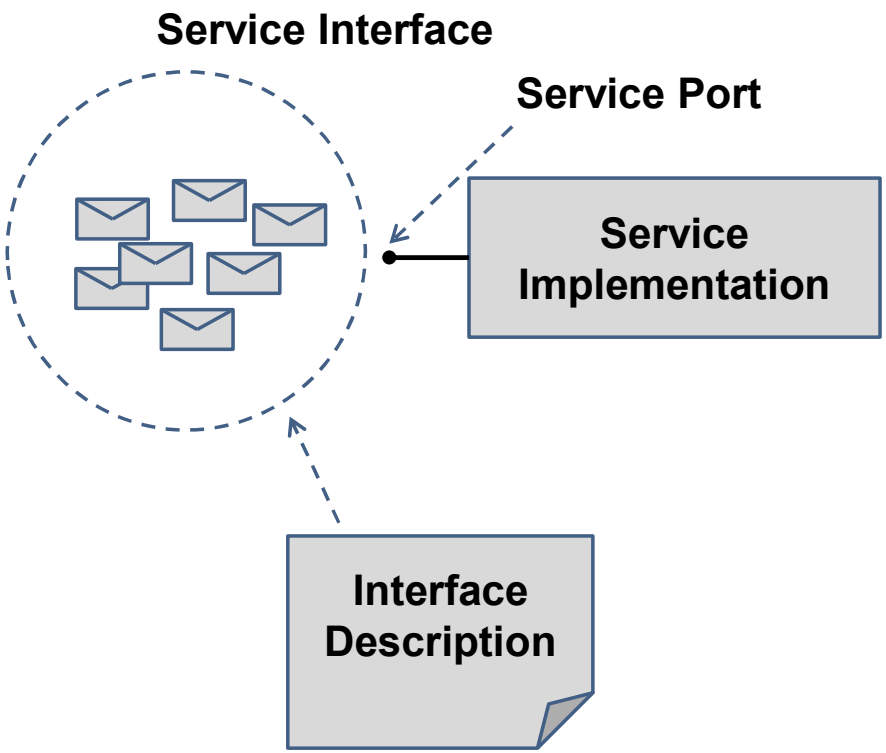


DSG-SOA-M 2024:
- Describing Services -

Dr. Andreas Schönberger

Lehrstuhl für Praktische Informatik
Fakultät WIAI
Otto-Friedrich-Universität Bamberg

What is Service Description?



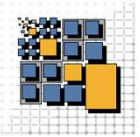
Let's reconsider the service definition of chapter 3!

Service description actually is interface description.

A service
"is what you can do with it", this comprises:

- Supported messages
- Message formats
- Semantics of message exchanges
- Constraints on message contents
- Constraints on interaction sequences (behavior)

Why Service Description?



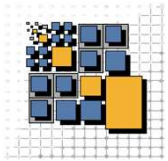
Service description is both a technology and a process topic!

From a **technology** point of view, you want

- Decoupling of service interface and service implementation
- Code-first support
- Versioning
- Automation in terms of testing, mocking, monitoring

From a **process** point of view, you want

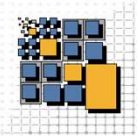
- Contract-first development
- Documentation
  - Easy understanding
  - A single source of truth



- Service Use Cases -

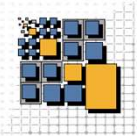
Lehrstuhl für Praktische Informatik
Fakultät WIAI
Otto-Friedrich-Universität Bamberg

Service Use Cases

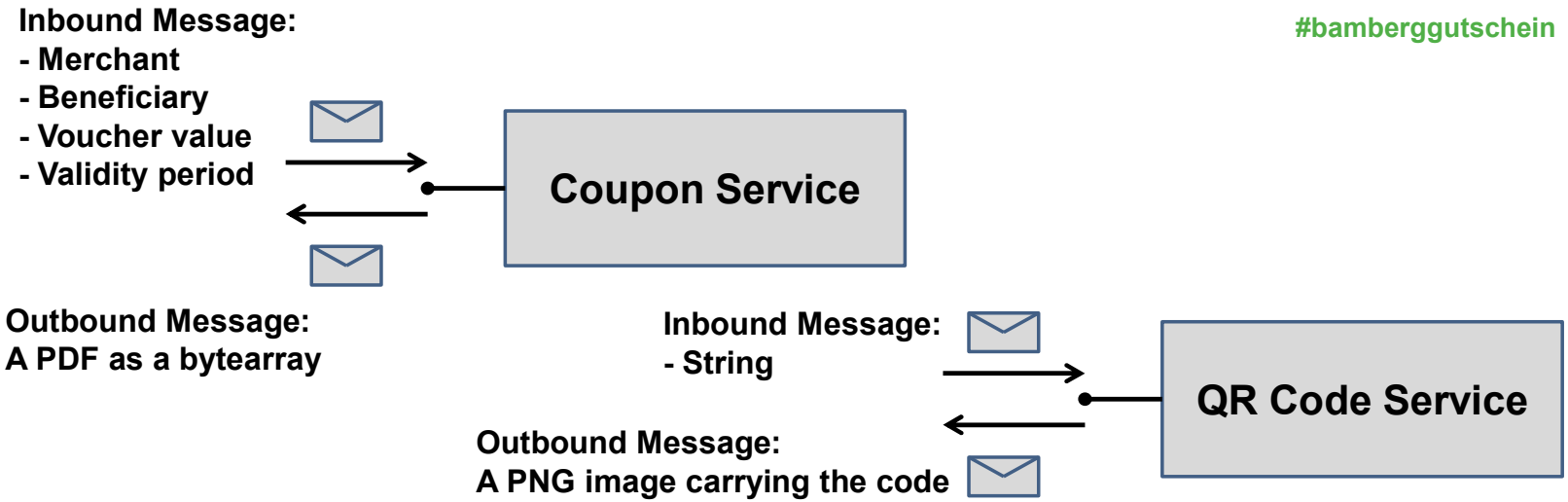


- Service description highly depends on the use case!
  → still, let's try to give a rough classification
- Dedicated API
  - Describe a dedicated, well-defined functionality
  - Need a compact representation
- Business document exchange
  - Describe a legally binding interaction between companies
  - Need a standardized format that can be adapted to concrete companies
- Message hub / Crosscutting service
  - Describe recurring service integration tasks
  - Need a very precise format and interaction description
- What other service use cases can you come up with?

Dedicated API: Example



Let‘s reconsider the coupon service of chapter 3 and compare it to a QR Code service



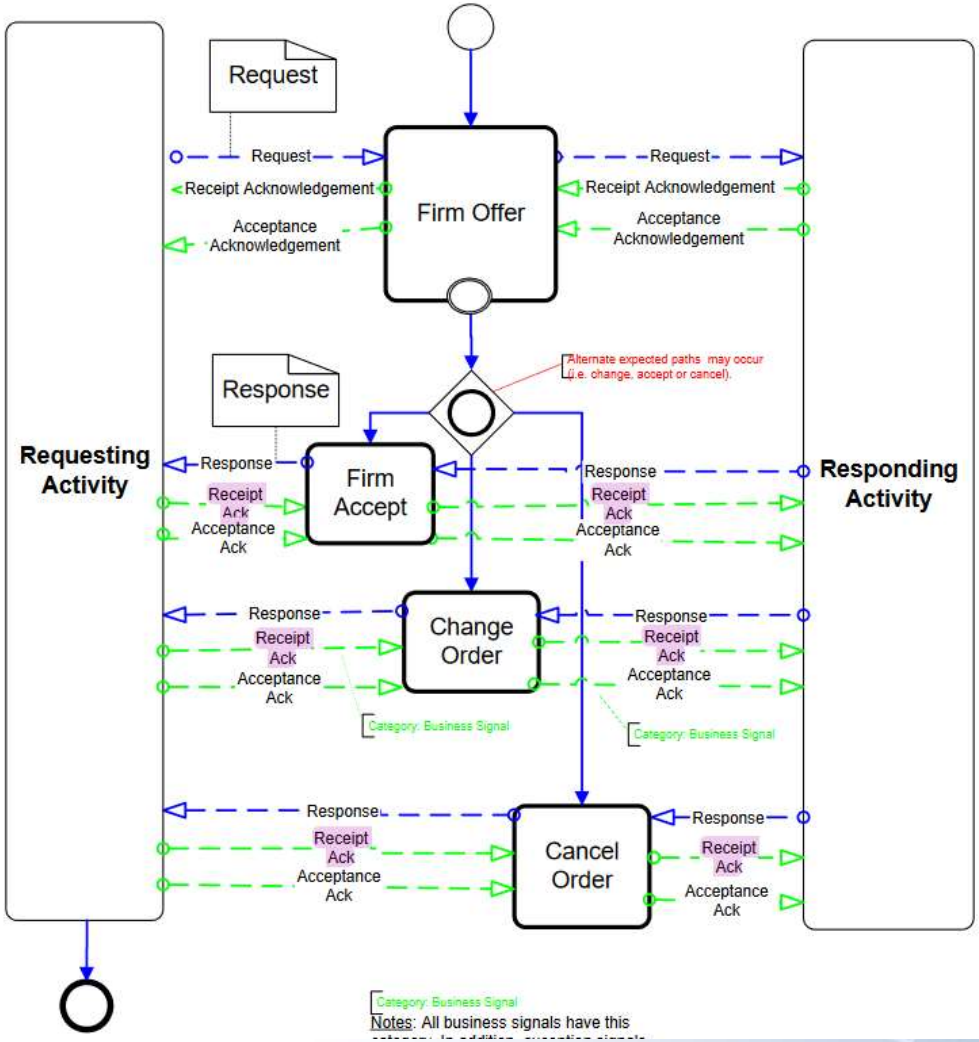
- Limited complexity
- Context-free?
- Idempotent?

Business Document Exchange

- Exchange an Offer and an Order
- ReceiptAcknowledgement to signal structural validity
- AcceptanceAcknowledgement to signal business validity

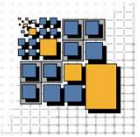
→ Is this actually a service?

btw: business docs may carry 1000s of attributes



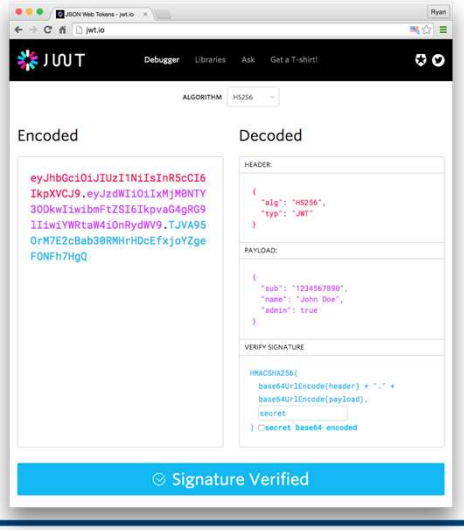
Src.: http://docs.oasis-open.org/ebxml-bp/2.0.4/OS/spec/ebxmlbp-v2.0.4-Spec-os-en.pdf

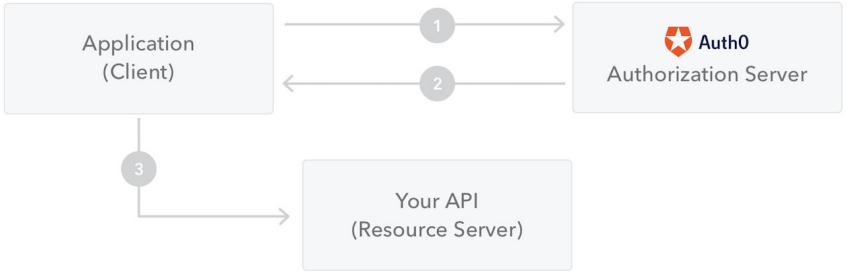
Message hub / Crosscutting service



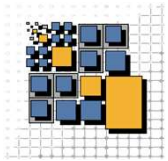
As an example, let‘s look a JSON Web Tokens (JWT) that are used to represent security claims in service calls.

- Describing JWT obviously must comprise two aspects
  - Describing the actual format
  - Describing the process for generating, checking JWT tokens
- That is what https://jwt.io/introduction/ actually does





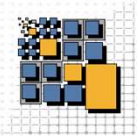
Src: https://jwt.io/introduction/



- Service Description Frameworks -

Lehrstuhl für Praktische Informatik
Fakultät WIAI
Otto-Friedrich-Universität Bamberg

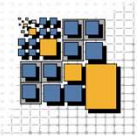
What‘s on Offer?



- Actual service description frameworks
  - **OpenAPI**
  - RAML
- Schema description frameworks
  - **JSON Schema**
  - Google Protocol Buffers
  - XML Schema
- Classical approaches
  - WSDL := Web Services Description Language
  - WADL := Web Application Description Language
- Prose

  “If it (the interface description) was deployed as a WADL file, there is a strong chance that clients would be compiled against that WADL, and those clients would break every time the WADL changed.”
  (http://bitworking.org/news/193/Do-we-need-WADL)

OpenAPI

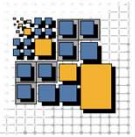


“The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection.”
(http://spec.openapis.org/oas/v3.0.3)

- Essentially focuses on HTTP interactions
- OpenAPI documents
  - describe APIs
  - can be represented as JSON or as YAML
- Ships with predefined types / schemes
  - Primtive types and advanced scheme objects suchs as OAuth flow objects
  - *Based on JSON Schema*
- What is missing?

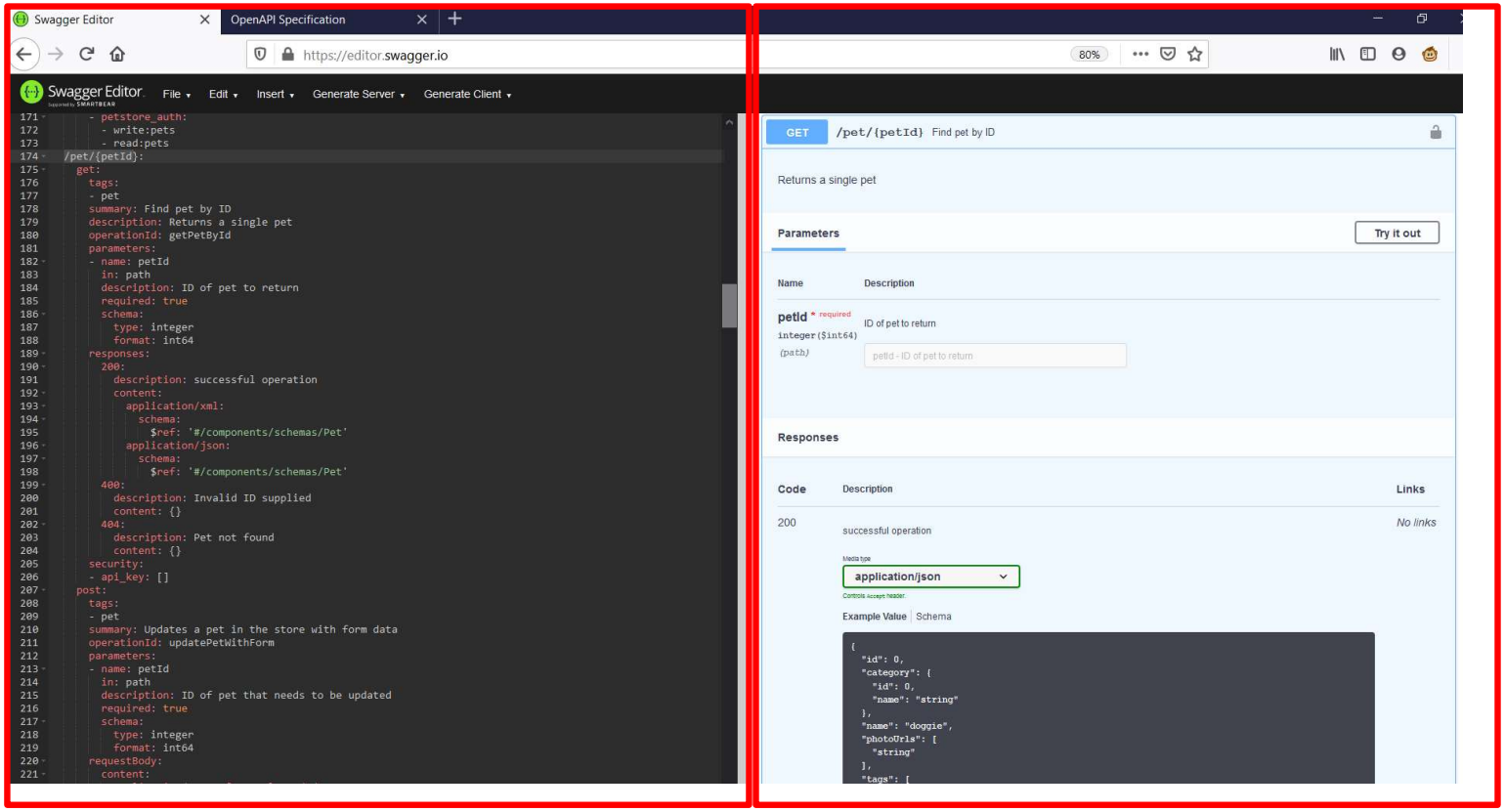
We look at OpenAPI version 3

OpenAPI Example I



Try https://editor.swagger.io/ for learning

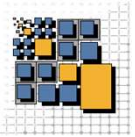
Examples taken from https://editor.swagger.io/



Specification **Tool Presentation**

OpenAPI Example II

Basic API information …



Examples taken from https://editor.swagger.io/

```yaml
openapi: 3.0.1
info:
  title: Swagger Petstore
  description: 'This is a sample server Petstore server.  You ca
    at [http://swagger.io](http://swagger.io) or on [irc.freenod
         For
    this sample, you can use the api key `special-key` to test t
  termsOfService: http://swagger.io/terms/
  contact:
    email: apiteam@swagger.io
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0.html
  version: 1.0.0
externalDocs:
  description: Find out more about Swagger
  url: http://swagger.io
servers:
- url: https://petstore.swagger.io/v2
- url: http://petstore.swagger.io/v2
tags:
- name: pet
  description: Everything about your Pets
  externalDocs:
    description: Find out more
    url: http://swagger.io
- name: store
```

Specification version

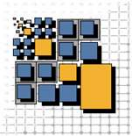
API version

Endpoint information

Grouping information for tools

OpenAPI Example III

… and path/operation description



Examples taken from
https://editor.swagger.io/

```yaml
paths:
  /pet:
  /pet/findByStatus:
  /pet/findByTags:
    get:
  /pet/{petId}:
    get:
      tags:
      - pet
      summary: Find pet by ID
      description: Returns a single pet
      operationId: getPetById
      parameters:
      - name: petId
      responses:
        200:
          description: successful operation
          content:
            application/xml:
              schema:
                $ref: '#/components/schemas/Pet'
            application/json:
              schema:
                $ref: '#/components/schemas/Pet'
        400:
        404:
      security:
      - api_key: []
    post:
      tags:
      - pet
```

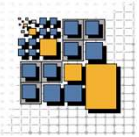
Paths: /pet, /pet/findByStatus

Operations available per path

Description of possible responses

Typing information

JSON-Schema



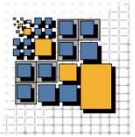
- see http://json-schema.org/specification.html for details
- IETF standard for describing the schema of JSON documents
- First draft in 2010, but really popular in recent years
- Nice online tooling for generating schemas / document instances: https://www.jsonschema.net/

You will probably have to do an edit after generation

- Binding libraries for almost any language available
- No built-in support for namespaces as opposed to XMLSchema; use prefixes for disambiguation
  → so federation of really complex data models might become challenging

Using JSON Schema is a choice, not an obligation

JSON-Schema Example I



For introducing JSON-Schema, let's look at the following use case:

- JSON documents shall be used to carry machine monitoring data
- Machines have status data sets and operation data sets
- Status and operation data sets share a certain set of attributes

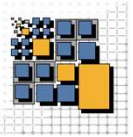
→ Schema file and a sample operation data set are available online.

Try and feed a json schema generator with the sample document!

MachineDatasetSchema.json | operationDataset.json

```
{
  "machineSerialNumber": "sH5f3L8uIn4W",
  "mac": "08-00-27-8E-0A-A5",
  "datasetType": "Operation dataset",
  "surveyStart": "2020-06-19T05:22:32",
  "surveyStop": "2020-06-19T05:23:32",
  "counts": [
    {
      "category": "X-Product",
      "taskReference": {
        "name": "X12-L34"
      },
      "timeStamp": "2020-06-19T05:23:05"
    },
    {"category": "Y-Product"...}
  ],
  "componentStates": [
    {
      "componentName": "C14",
      "communicationFailure": false,
      "counterFailure": false
    },
    {"componentName": "C32"...}
  ],
  "absoluteRunningNumber": 2,
  "runningNumber": 2,
  "systemStatus": {"ntpCorrupt": false...},
  "format": "https://districscale.com/operationDataSet",
  "version": "1.0.6",
  "operationDatasetCreationDate": "2020-06-19T05:30:07"
}
```

JSON-Schema Example II



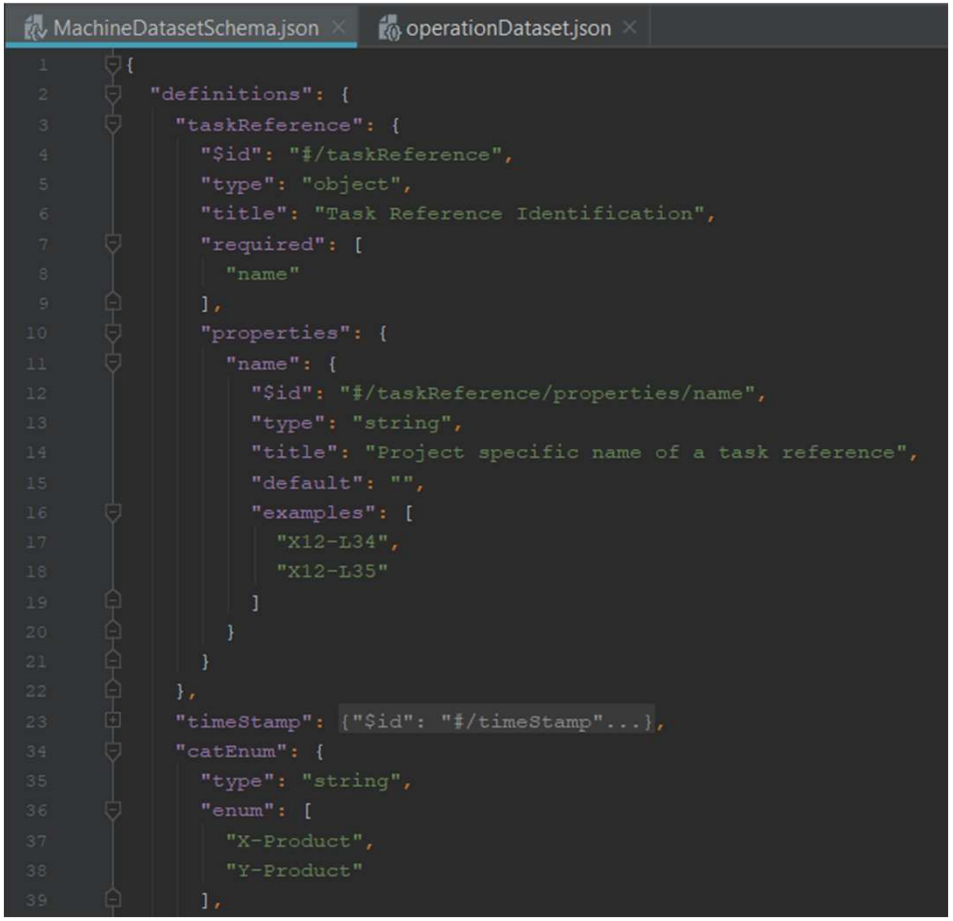
MachineDatasetSchema.json | operationDataset.json

```
{
  "definitions": {...},
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "https://distriscale.com/machineDatasetSchema",
  "format" : "https://distriscale.com/machineDatasetSchema",
  "version" : "1.0.6",
  "type": "object",
  "title": "Machine counting dataset schema",
  "allOf": [
    {"$ref": "#/definitions/datasetHeader"...},
    {
      "description": "A dataset either includes counting data (operat
      "oneOf": [
        {
          "$ref": "#/definitions/operationProperties"
        },
        {
          "$ref": "#/definitions/statusProperties"
        }
      ]
    },
    {
      "required": [
        "systemStatus"
      ],
      "systemStatus": {
        "$ref": "#/definitions/systemStatus"
      }
    },
    {...}
  ]
}
```

This is the basic anatomy of JSON schema:

- „definitions“ to define reusable document elements
- Meta data to identify the schema
- Definition of the top-level elements, e.g., datasetHeader, systemStatus etc.

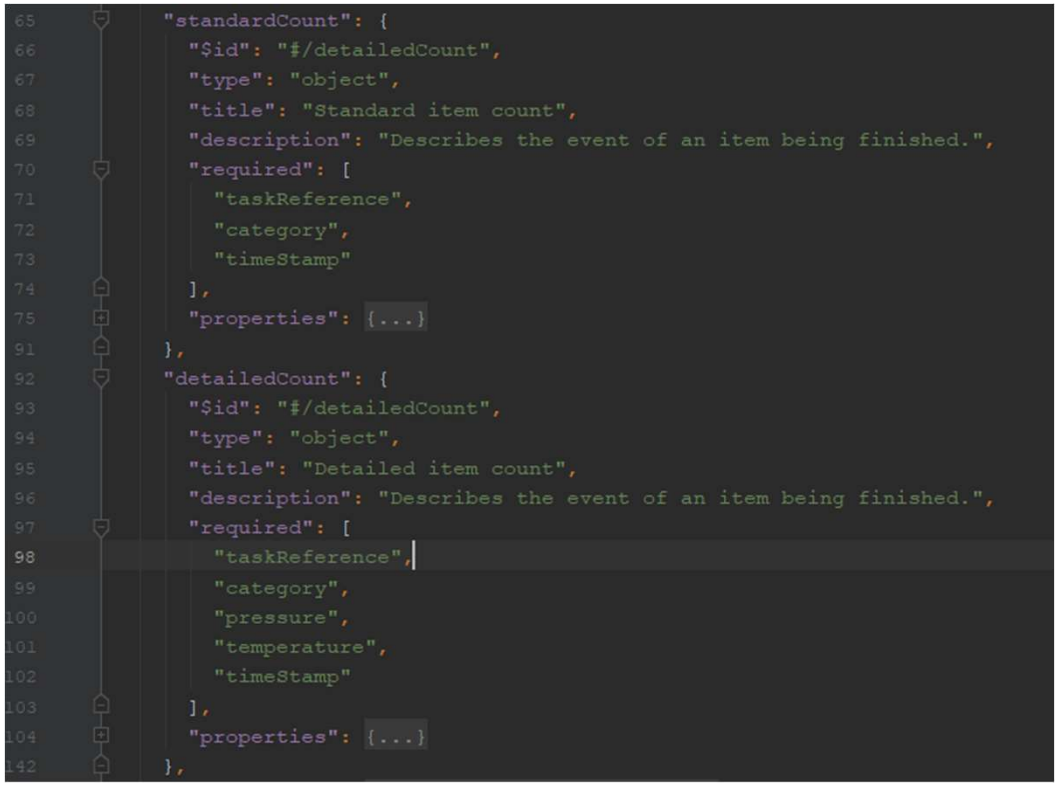
JSON-Schema Example III



- Use „definitions“ to create a set of object definitions that can be reused across properties of the same or different documents
- $id defines a local name relative to its context
  → taskReference can be referred to as "$ref": "#/definitions/taskReference"

Note: „definitions“ has been renamed to „defs“.

JSON-Schema Example IV



```
65    "standardCount": {
66      "$id": "#/detailedCount",
67      "type": "object",
68      "title": "Standard item count",
69      "description": "Describes the event of an item being finished.",
70      "required": [
71        "taskReference",
72        "category",
73        "timeStamp"
74      ],
75      "properties": {...}
91    },
92    "detailedCount": {
93      "$id": "#/detailedCount",
94      "type": "object",
95      "title": "Detailed item count",
96      "description": "Describes the event of an item being finished.",
97      "required": [
98        "taskReference",
99        "category",
100       "pressure",
101       "temperature",
102       "timeStamp"
103     ],
104     "properties": {...}
142   },
```

This example here solves the problem that a complete group of attributes is optional.

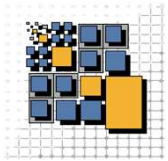
We want to define that if an optional attribute is used (pressure) than all attributes must be used (temperature as well)…

JSON-Schema Example V

```
337     "operationProperties": {
338       "$id": "#/operationProperties",
339       "title": "Specific operation data set properties.",
340       "required": [
341         "counts"
342       ],
343       "allOf": [
344         {
345           "properties": {
346             "counts": {
347               "$id": "#/operationProperties/properties/counts",
348               "type": "array",
349               "title": "The count objects",
350               "description": "Potentially empty list of count events",
351               "items": {
352                 "anyOf": [
353                   {
354                     "$ref": "#/definitions/standardCount"
355                   },
356                   {
357                     "$ref": "#/definitions/detailedCount"
358                   }
359                 ]
360               }
361             }
362           }
```

…

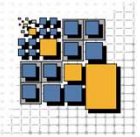
So we have a standardCount or a detailedCount, but not a standardCount with just pressure.



- Summary -

Lehrstuhl für Praktische Informatik
Fakultät WIAI
Otto-Friedrich-Universität Bamberg

Summary



- Service description practices need to match the development task at hand
- Bigger development teams need frameworks that support automation, versioning and exploration of service descriptions
- Dedicated APIs may get by with simple input/output relations
- Complex document formats will need a reasonable schema description
- Behavior / complex multi-step service interaction is still an issue
  - How do you determine the validity of message sequences?
    - To-Be Behavior (StateCharts, Sequence Diagrams, …)
    - Error Behavior (StateCharts, Sequence Diagrams, …)
  - Choreography and orchestration languages have been designed for this purpose, but have not been widely adopted