Workflow DSL Specification

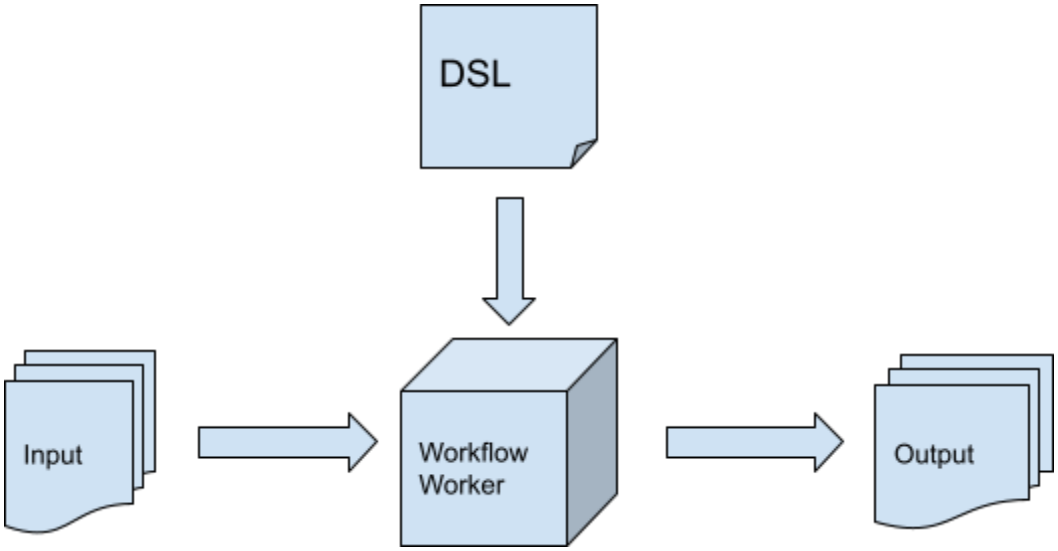
This document outlines a proposed Workflow DSL (Domain Specific Language) specification for the Workflow Orchestration Microservice. Our DSL specification is roughly based on the Serverless Workflow DSL specification (as documented on serverlessworkflow.io), with some modifications as described below.

Workflow DSL Format

The Workflow DSL file may be written in both JSON and YAML format. The workflow consists of a header section for the main Workflow properties, an Activities section which specifies each activity to be executed, and an Messages section which specifies any messages that the Workflow may consume or produce.

Workflow Data

While a Workflow DSL specifies a set of operations, these workflows generally require data to be passed into them as input. At the end of the Workflow execution data is passed out of the Workflow as output.



Data is passed from the Workflow to each Activity as it gets executed, and then back out to the Workflow. Data may also be passed into the Workflow from consumed Messages, or may be passed from the Workflow into produced Messages. This will be described in detail in the next sections.

Data filters may be specified on each of the above sections to restrict what data gets passed on. For example, given the following data:

```
{
  "fruits": [ "apple", "orange", "pear"],
  "vegetables": [ "carrots", "cabbage", "kale"],
  "nuts": [ "peanut", "almond"]
}
```

If the following filter is applied:

```
"activityDataFilter": {
  "input": ${ {fruits: .some-fruits, nuts: .some-nuts} }
}
```

Will select the fruits sub-section resulting in the following input:

```
{
  "some-fruits": [ "apple", "orange", "pear"],
  "some-nuts": [ "peanut", "almond"]
}
```

If the first term in the filter is empty, then the filter will select the entire data set and map it to an index specified by the second term. For example:

```
"activityDataFilter": {
  "input": ${ {.items} }
}
```

Will result in:

```
{
  "items": {
    "fruits": [ "apple", "orange", "pear"],
    "vegetables": [ "carrots", "cabbage", "kale"],
    "nuts": [ "peanut", "almond"]
  }
}
```

If the second term in the filter is empty, then the filter will select the matching sub-arrays and merge the data elements together. For example:

```
"activityDataFilter": {
  "input": ${ {vegetables, nuts} }
}
```

Will result in:

```
{
  [ "carrots", "cabbage", "kale", "peanut", "almond"]
}
```

A blank filter (or if no filter is provided) will pass through all data as is without any transformations.

Workflow Headers

Workflow headers are specific to the Workflow itself. The following are allowed header properties.

- id - a unique identifying string for the Workflow
- namespace - the namespace of the Workflow
- name - the name of the Workflow
- version - version of the workflow
- specVersion - specification version of the workflow
- hash - a hash string corresponding to the workflow contents
- isChild - a boolean value that specifies whether this workflow may be called as a child workflow

- firstActivity - string value that corresponds to the name of the first activity to be executed. If not provided, then the workflow will execute the Activity in the ‘activities’ list.
- workflowDatafilter - specifies what data filter to be applied to the incoming data input as well as outgoing data. For example:

```
“workflowDataFilter”: {
  “input”: ${ {user : .data} },
  “output”: ${ {data: .customer} },
}
```

Activities

Activities are the building blocks of Workflows. Each Activity may be executed in turn by the Workflow sequentially, or may be triggered by a Message. After an Activity executes, the Workflow may terminate, transition to a different state, or wait for additional Messages.

Activity properties include the following:

- type - May be ‘message’, ‘operation’, ‘switch’, ‘inject’, or ‘subFlow’. Each type is described in more detail later.
- name - decorative name of the activity
- transition - Defines how the workflow will transition to the next state.
- sleep - Defines time periods that the activity can sleep, either before or after.in ISO 8601 duration format. For example:

```
"sleep": {
  "before": "PT1M",
  "after": "PT3M"
}
```

- timeout - Defines timeout periods for the activity. Can have the following sub-properties
  - scheduleToClose - Seconds between when activity is scheduled to when it terminates
  - scheduleToStart - Seconds between when activity is scheduled to when it starts
  - startToClose - Seconds between when activity starts to when it terminates

Example:

```
"timeout": {
  "startToClose": "10",
  "scheduleToStart": "15"
}
```

- retry - Defines retry policy for the activity
  - maxAttempts - Number of retry attempts
  - initialInterval - Seconds to wait for the first retry

    - backoffCoefficient - Coefficient used to calculate the next retry interval. The next retry interval is previous interval multiplied by this coefficient.
    - maximumInterval - Maximum interval between retries. Exponential backoff leads to interval increase. This value is the cap of the increase.

Example:

```
"retry": {
  "maxAttempts": "10",
  "initialInterval": "2",
  "backoffCoefficient": "1.2",
  "maximumInterval": "60"
}
```

- end - True if the workflow should end after completion of this Activity
- activityDataFilter - specifies the data filter to be applied to the incoming data input as well as outgoing data. For example:

```
"activityDataFilter": {
  "input": ${ {emailfields: .data} },
  "output": ${ {output: .result} },
}
```

Activity Actions

Each Activity may include a set of Actions which specify what operations to execute. Each 'actions' block include a list of Action objects, each of which has the following properties. Actions within an action block are executed sequentially.

- type - May be 'rest' for restful calls, 'sleep', or 'produceMessage' to produce a Message
- sleep - Defines time periods that the action can sleep, either before or after.in ISO 8601 duration format
- operation - For 'rest' Actions, specifies the URL of the rest call
- method - Also for 'rest' Actions, specifies the HTTP method to be used, e.g. 'POST', 'GET'
- query-parameters - List of arguments to be passed to the rest call
- header - List of headers to be passed to the Rest call

Example of a Rest Action

```
{
  "actions": [
  {
    "type": "rest",
    "sleep": {
      "before": "PT15S",
      "after": "PT1S"
    },
```

```
    "operation": "http://streetlightsapi.com/object",
    “method”: “GET”,
    "query-parameters": {
      "id": "${ .currentLight.id }",
      "lumens": "${ .currentLight.lumens }",
      "sentAt": "${ now }"
    },
    “header”: {}
  }]
}
```

- messageRef - List of message to be produced by the ‘produceMessage’ Action.
- data - For ‘produceMessage’ Actions, specifies the data filter to be passed on to the Message.

Example of a ‘produceMessage’ Action:

```
{
  "actions": [
  {
    "type": "produceMessage",
    "sleep": {
      "after": "PT6S"
    },
    "messageRef": { "emailSent" },
    "data": "${ .emailInfo }"
  }]
}
```

Message Activities

Message activities listen to a particular Message and then execute based on the input. The Message must be declared in the ‘messages’ section of the Workflow DSL.

Properties for Message Activities include the following:

- onMessages - Contains the following:
    - messageRefs - List of Message that the Activity is listening for
    - actions - List of actions that may be performed, specified above.
- exclusive - if true any message in the list triggers the specified actions to happen. If false,then all of the message must be received in order to trigger the specified actions.

Example Message Activity:

```
{
  "type": “message”,
  "name": “Listen for Signal”,
  "exclusive": true,
  "onMessages": [{
    "messageRefs": ["SignalOne", "EventTwo"],
    "actions": [{
      "type": "rest",
      "operation": "http://streetlightsapi.com/object",
      “method”: “GET”,
      "arguments": {
        "id": "${ .currentLight.id }",
        "lumens": "${ .currentLight.lumens }"
      }
    }]
  }]
}
```

Operation Activity

An operation activity consists of a simple list of actions to be executed.

Example Operation Activity:

```
{
  "type": “operation”,
  "name": “Process checkout”,
  "actions": [{
    "type": "rest",
    "operation": "http://api.com/checkout",
    “method”: “POST”,
    "arguments": {
      "id": "${ .cid }",
      "items": "${ cart.items }"
    }
  }]
}
```

Switch Activity

Switch Activities define transitions depending on the state of data in the Workflow.

- dataConditions - list of data conditions to trigger transitions to other states. Each element in the dataConditions has the following properties
    - name - decorate name of condition
    - condition - logical clause to evaluate for the condition to be true
    - transition - state transition that should happen if the condition evaluates to be true
- defaultCondition - defines the default transition that must happen if no conditions match.

Example of a Switch Activity

```
{
  "name":"CheckVisaStatus",
  "type":"switch",
  "dataConditions": [
    {
      "name": "Eighteen or older",
      "condition": "${ .applicant | .age >= 18 }",
      "transition": "StartApplication"
    },
    {
      "name": "Eighteen or younger",
      "condition": "${ .applicant | .age < 18 }",
      "transition": "RejectApplication"
    }
  ],
  "defaultCondition": {
    "transition": "HandleNoVisaDecision"
  }
}
```

Inject Activity

The inject activity simply injects data into the Workflow data state.

- data - specifies the data to be directly injected into the workflow data.

Example Inject Activity:

```
{
  "name":"Hello World",
  "type":"inject",
  "data": {
    "result": "Hello"
  },
   "end": true
}
```

SubFlow Activity

SubFlow Activities invoke other workflows as child workflows of the current running Workflow.

- workflowId - unique ID of the Workflow to be invoked
- version - specific version string to invoke

Example SubFlow Activity:

```
{
  “type”: “subFlow”,
  "workflowId": "handleApprovedVisaWorkflowID",
  "version": "2.0",
  “activityDataFilter”: {
    “input”: ${ {emailfields: .data} },
    “output”: ${ {output: .result} },
  }
}
```

Messages

Messages are data payloads may be consumed by the Workflow or may be produced by the Workflow execution. All Messages must be declared in the ‘messages’ section of the DSL in order to be referenced elsewhere.

The following properties are allowed for messages.

- name - decorate name of the message
- messageType - ‘event’ or ‘signal’
- kind - ‘produced’ or ‘consumed’
- type - Cloud Event type
- source - Source of the Cloud Event

Example Messages:

```
{
  "messagess": [
    {
      "name": "HeartRateReadingSignal",
      "messageType": "Signal"
      "source": "hospitalMonitorSystem",
      "type": "com.hospital.patient.heartRateMonitor",
      "kind": "consumed",
      “messageDataFilter”: {
        “input”: ${ {heartrate : .rate} },
        “output”: ${ {data: .heartrate} },
      }
    }
  ]
}
```