

Orchestrator User Manual

1. Concepts

The Orchestrator is a component that allow to define and execute an orchestration of existing services. The services to be orchestrated have to be registered in the Orchestrator through the configuration of specific connectors, that define the way to connect and invoke the service, including a formal description of the required inputs and produced outputs. As soon as the services are registered the orchestration logic can be expressed in JavaScript format, and executed on request through a REST interface.

In the Orchestrator the configuration of such connectors are named Operations, and are grouped in structures named Microservices.

2. User Interface

The main interface is composed of a simple view that allow to select a Microservice group from a list of all the public microservices groups or provide the ID of the microservice group in case of a private one. As soon as a microservice is selected or provided, the list of all its operation is filled. From this point the interface allow to delete the microservice group using the Delete button, edit the microservice operations configuration via the Edit button, starting all its operations using the Start button and stopping all the started operations using the Stop button. When the user selects an operation, its status is displayed as a green, yellow, red indicator indicating respectively if the microservice operation is running correctly, if is running with some errors (reporting them) or if is stopped. Pressing now the Start button the single operation can be started if stopped, while pressing the Stop button can be stopped if started. Using the Test, a Call button is now possible also to test the microservice operation via another interface. At any time is possible to click the Create New button that allow to configure a new microservice starting from scratch.

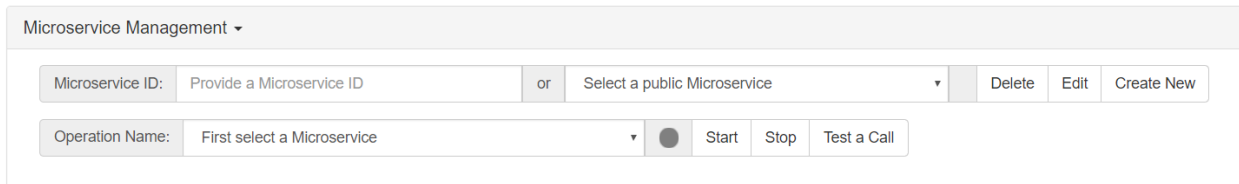


Figure 1 - Orchestrator Microservice Management UI - Main view

The Testing interface that appears when the “Test a Call” button is pressed, allows to easily interact with the microservice REST endpoint, providing to the user a view of the required inputs to provide and of the produced output. All the input fields are generated automatically from the microservice definition and clicking the testing button they will be combined in the format accepted by the microservice, sent to it and the output visualized in the relative view. The interface so (1) visualizes the full REST endpoint to call and the JSON of input data to post in order to use the microservice, (2) allow to test the microservice with the provided inputs and (3) visualize the output in raw JSON with possibility to test a JavaScript algorithm to process it.

Microservice Required Inputs

Append Text world

Test a Call

Microservice ID: 1738cb62-cc55-4abf-8560-feafdb83260c

Microservice Operation: default

POST Endpoint

https://www.adoxx.org/micro-service-controller-rest/rest/msc/callMicroserviceForced?microserviceId=1738cb62-cc55-4abf-8560-feafdb83260c&operationId=default

POST Input Data

{
 "Append Text": {
 "value": "world"
 }
}

Output description:

Output

{
 "dataMIME": "text/plain",
 "dataText": "Hello world",
 "moreInfo": {
 "retrievalTime": "2020-05-19 14:53:18"
 },
 "newField": "test"
}

Custom Rendering Algorithm

```

1 /*
2 Javascript algorithm that "return" a DOM object.
3 The algorithm can access the microservice output content
4 using the variable "output"
5 */

```

Service Output Post-Rendering Preview

{
 "dataMIME": "text/plain",
 "dataText": "Hello world",
 "moreInfo": {
 "retrievalTime": "2020-05-19 14:53:18"
 },
 "newField": "test"
}

Cancel

Continue

Figure 2 - Orchestration UI - Test view

The interface visualized when the Create New button is pressed on the main view is the same of the one visualized when the Edit button on the main view is pressed. The only difference is the content visualized, that in the first case is empty, while in the second case is pre-filled with the data of the existing microservice definition.

This interface allows to specify first the details of the microservice with its name and description and if it must be visualized in the list of the public microservices available in the main view or if is a private one and accessible only knowing its ID. Then allow to add operations to the microservice. The operations created can be deleted pressing the delete button available on the right side of the operation header. Clicking on the header it will expand allowing to configure all the details of the operation.

Microservice Definition

✕

Microservice Name	Description	VirtualWeatherServices	Is Public?	Show Details
WeatherStation			<input checked="" type="checkbox"/>	

New Operation

Operation weatherInformation ▾

Delete

Cancel

Continue

Figure 3- Orchestrator Management UI - Edit/New Microservice view collapsed.

Expanding each operation section the user can modify the operation id, its visualized named and description and if the operation should start automatically on startup. If not selected the user must start it manually before being able to use it. If the default checkbox is checked means that this operation is the

entry point of the microservice and can be executed without providing the operation id but only the microservice id. Only one operation per microservice can be set as default. After this general configuration, the user must select the connector to use for this operation. The select box provide a list of all the connectors available and on its selection change, all the subsequent configurations will change accordingly.

The start configuration view will visualize all the inputs required by the connector during its initialization. These vary from connectors to connectors and may also be empty. In the case of the REST Connector this section will contain the definition of the REST method to use, the expected mime content type and optionally a list of additional HTTP headers to setup. The Call Configuration view will display all the inputs required by the connector during the execution phase. These also vary from connectors to connectors and may also be empty. In the case of the REST Connector this view will display the configuration of the REST endpoint with optionally its query string and in case of POST or PUT methods optionally the data to send. This section is the only one that can be affected by the microservice input from the user, using the placeholders defined in the microservice inputs section. The Call Configuration Inputs view allow to define the microservice inputs to ask to the final users and the placeholder to replace. Every row in this section defines a microservice input. New inputs can be added using the “Add New Call Configuration Input” button and removed pressing the X button relative to the row. Every input requires a unique ID, the name of the placeholder to match for the replacements with the inputs provided by the final users, a description and an example of working input value. This value will be used also during the microservice status check to evaluate the correctness of the microservice output. The next section to configure is the output description of your microservice that may be the same of the connector output if no adaptation algorithm is defined. In case the output must be adapted, a JavaScript code can be provided in the Output Adaptation Algorithm section. This JavaScript code can access the original output of the connector through the variable `output` and the microservice inputs through the variable `input`. Both values are in form of JSON objects. Additionally a function “`out ({...})`” is available and must be called as last instruction of the algorithm in order to return the new value provided as parameter to the out function in JSON object format. In this code also the `callMicroservice(microserviceId, operationId, microserviceInputs)` function can be used in order to call another microservice operation and use its output. The last section allows to provide an algorithm to process the output during the microservice status check. The last instruction in this case must be a Boolean value identifying if the output is correct or not.

Microservice Name	WeatherStation	Description	VirtualWeatherServices	Is Public?	<input checked="" type="checkbox"/>	Show Details
-------------------	----------------	-------------	------------------------	------------	-------------------------------------	--------------

New Operation

Operation weatherInformation Delete

Operation ID	weatherInformation	Name	weatherInformation	Description	Retrieve base weather information	Is Default?	<input checked="" type="checkbox"/>	Autostart?	<input checked="" type="checkbox"/>
--------------	--------------------	------	--------------------	-------------	-----------------------------------	-------------	-------------------------------------	------------	-------------------------------------

Connector REST Connector Get data from a REST service

Start configuration

Method GET

Content Type application/json

Additional Headers

Call configuration

Endpoint `http://api.openweathermap.org/data/2.5/weather?q=%CITY%&APPID=ee97a2fe49de48925d8a8b78dc95d836&units=metric`

QueryString

POST Data

Call Configuration Inputs

Add new call configuration input

Input ID	cityname	Matching Name	%CITY%	Description	City name	Working Sample	Vienna	X
----------	----------	---------------	--------	-------------	-----------	----------------	--------	---

Connector Output Description

A JSON object in the following format:

```
{
  dataMIME: 'text/' + 'application/json' / 'all the other cases',
  dataText / dataJson / dataBase64: '_PlainText_' / '_JsonObject_' / '_ContentBase64_',
  moreInfo: {
    retrievalTime: "
  }
}
```

Output Description

Output Adaptation Algorithm

```
1 out({
2   temperature: output.dataJson.main.temp
3 });
```

Status Check Algorithm

```
1
```

Cancel Continue

Figure 4 - Orchestrator Management UI - Edit/New Microservice view expanded.

2.1. Connectors Collection

The Orchestrator provides out-of-the-box the following connectors:

- **ADOxx Classic Connector:** This connector allows to communicate with the ADOxx Modeler Classic (desktop application) exploiting its SOAP interface. This allow to execute custom AdoScripts (only a restricted set is allowed) remotely and create microservices that interacts with models.
- **Content Provider Connector:** This connector allows to provide an arbitrary content to download. It is used to create microservices that return previously uploaded data.
- **Content Receiver Connector:** This connector allows to upload an arbitrary content and store internally in the platform. It is used to create microservices that upload data needed by other microservices.
- **KPIs Engine Connector:** This connector allows to calculate the KPIs, and metrics defined in a KPI model and return their value. The connector can interpret the model, identifying dependencies between KPIs and evaluate their value and success status.
- **Excel Connector:** This connector allows to read values available inside an Excel document, evaluating formulas and controlling the cells to read.
- **JavaScript Engine Connector:** This connector allows to execute a JavaScript code evaluating it using the Java Nashorn engine. The JavaScript is executed in a restricted environment for security reasons, and this allow to create general purpose microservices that perform any kind of operation available in the JavaScript programming language. This connector allows to create orchestration of existing microservices as it expose the same `callMicroservice` function available in the “Output Adaptation Algorithm”, that invoke a microservice and return its output.
- **JMS Publisher Connector:** This connector is used to connect to a message bus compatible with the JMS standard and publish some content on a specific topic.
- **JMS Subscriber Connector:** This connector is used to connect to a message bus compatible with the JMS standard and listen on new messages on a specific topic. This is a connector that follow the asynchronous communication pattern.
- **Microservice Connector:** This connector is used to schedule the execution of an existing Microservice and run it in background. This is used to convert every synchronous microservice in an asynchronous one.
- **MySQL Connector:** This connector allows to perform a generic SQL query in a remote MySQL database. This allows to both insert and retrieve data from the database.
- **REST Connector:** This connector allows to communicate with existing remote services exposed through the REST protocol. It can be used to create microservices that exploit the features of external services through a wrapping/proxying around the original service functionalities.
- **SMTP Connector:** This connector allows to interact with a remote SMTP server to send an e-mail programmatically.
- **SOAP Connector:** This connector allows to communicate with existing remote services exposed through the SOAP protocol. It can be used to create microservices that exploit the features of external services through a wrapping/proxying around the original service functionalities.

3. Deployment

The Orchestrator provide a Docker image that simplify the deployment on production servers and Linux environment. The repository <https://github.com/Modapto/orchestrator> contains the Dockerfile that generate a Docker image ready to be executed containing the last version of the Orchestrator.

In order to start a Docker container, you have to perform the following steps:

1. Download the Dockerfile from the repository <https://github.com/Modapto/orchestrator>

2. Build the image using the command:

```
sudo docker build --no-cache -t orchestrator-msc .
```

3. Run the container using a folder as volume for persistence:

```
mkdir ./msc-data  
sudo docker run -d -p 8080:8080 --name orchestrator -msc -v ${PWD}/msc-data:/opt/msc-data/ orchestrator-msc
```

The management web interface of the Orchestrator will be now available at <http://127.0.0.1:8080/micro-service-controller-rest/>.