Arrowhead Framework

Hello World tutorial

Olivér Nagy

What is the Arrowhead Framework?

The Arrowhead Framework is on open source, generic and comprehensive framework for addressing universal challenges in a successful IIoT infrastructure.

The Arrowhead Framework provides a Service Oriented Architecture (SOA) and several standardized modules for further extension.

The core components of the Arrowhead Framework

To establish the Arrowhead Framework, three mandatory core systems are required. These are the following:

i) Service Registry

ii) Orchestrator

iii) Authorization

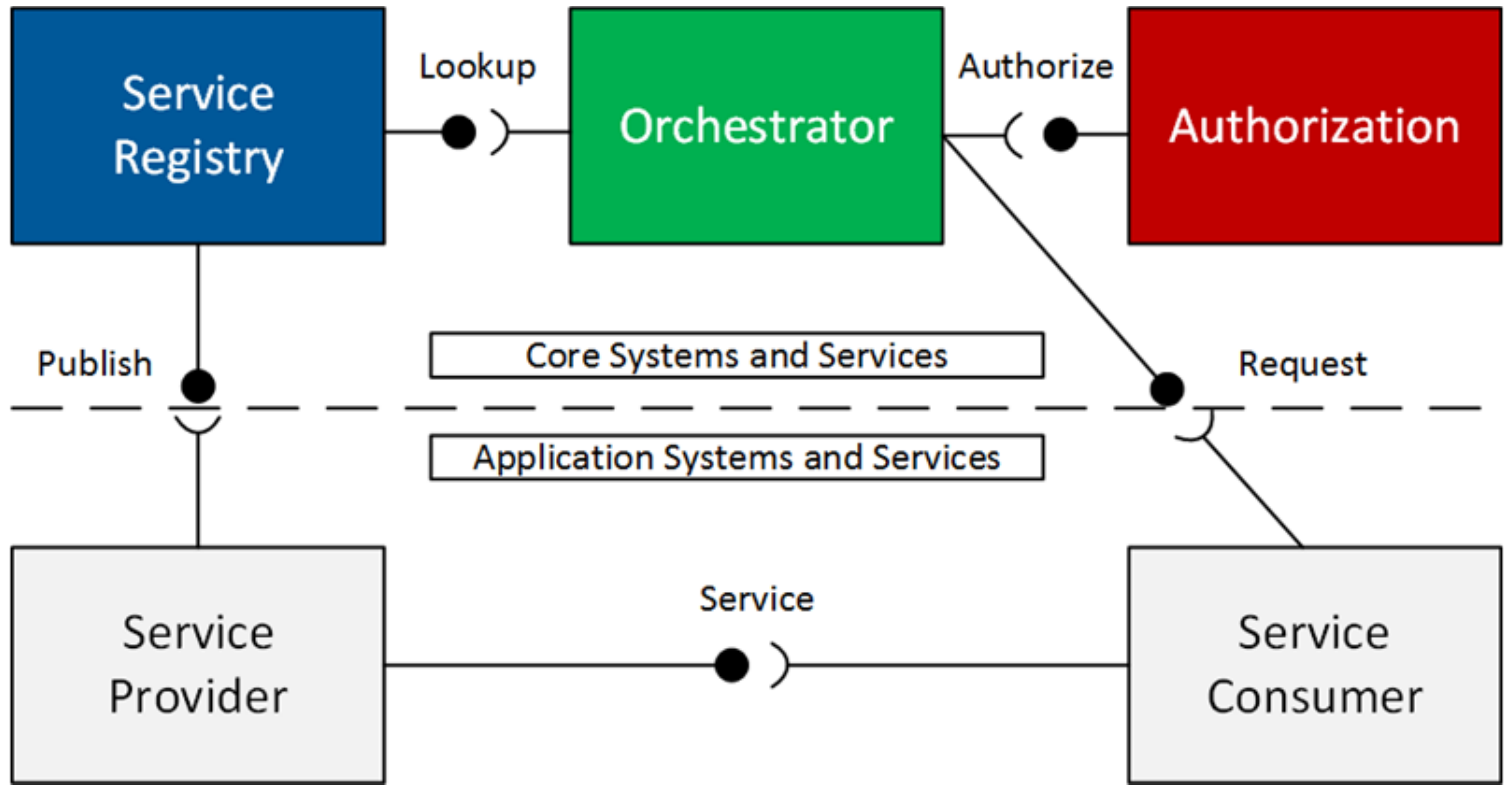

Figure 1. The architecture of the mandatory core systems

Service Registry

Enables publishing services into the local instance and allows other consumers to look up service information.

The most important client endpoints of the Service Registry:

Function	**URI**	**Method**	**Input**	**Output**
Echo	/echo	GET	-	OK
Query	/query	POST	ServiceQueryForm	ServiceQueryList
Register	/register	POST	ServiceRegistryEntry	ServiceRegistryEntry
Unregister	/unregister	DELETE	Address, Port, Service Definition, System Name in query parameters	OK
Get all entries	/mgmt/	GET	-	ServiceRegistryEntryList

Table 1. Client endpoints

Example using the Service Registry

List every registered entity in the registry:

curl -v -s --insecure --cert-type P12 --cert {path to sysop certificate}:{passcode} -X GET https://localhost:8443/serviceregistry/mgmt

The request should return a ServiceRegistryEntryList object:

```
{
  "serviceDefinition": "string",
  "providerSystem": {
    "systemName": "string",
    "address": "string",
    "port": 0,
    "authenticationInfo": “string” …}
…}
```

Example using the Service Registry

Registering a new service provider:

curl -v -s --insecure --cert-type P12 --cert {path to provider certificate}:{passcode} -X POST -H "Content-Type: application/json" -d {service registry entry form} https://127.0.0.1:8443/serviceregistry/register

The request should return a ServiceRegistryEntry object:

```
{
  "serviceDefinition": "string",
  "providerSystem": {
    "systemName": "string",
    "address": "string",
    "port": 0,
    "authenticationInfo": “string” …}
…}
```

Note that the previous response held a list of these service elements.

Authorization

The Authorization is responsible for service consumers accessibility to service providers.

Each of the registered providers and consumers must be authorized to request orchestration process.

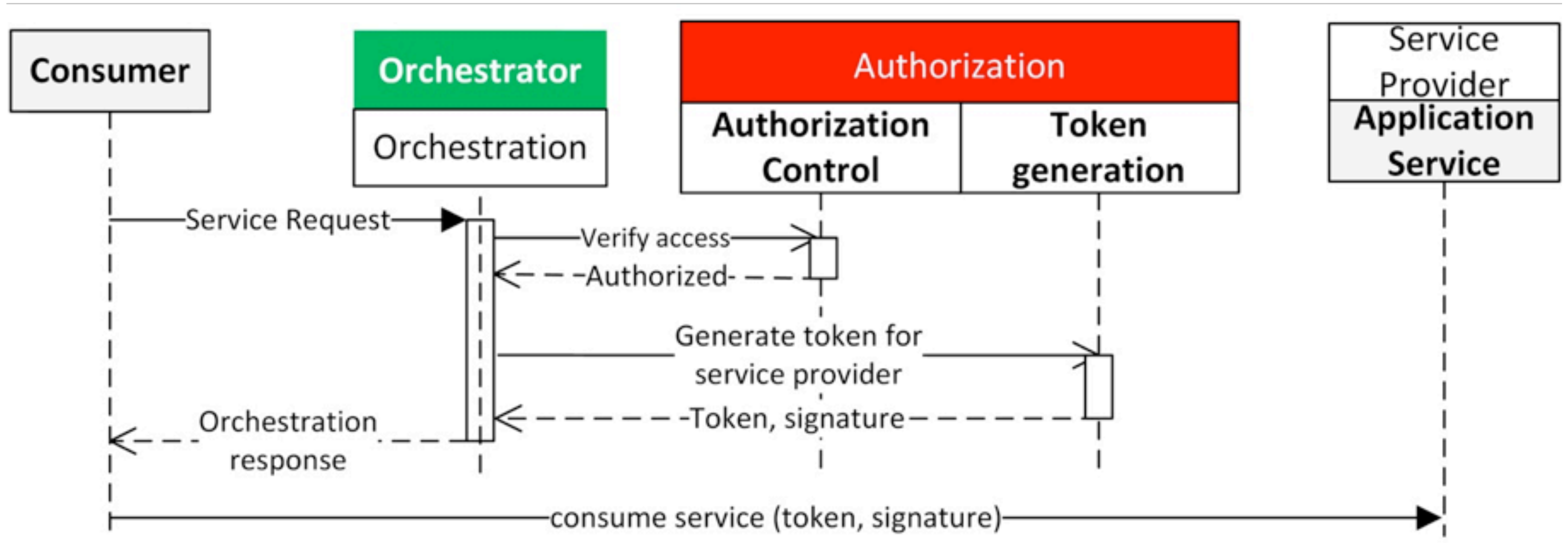

Figure 2. Authorization crosscheck during orchestration process

Authorization

In this demo we do not discuss the token generation process, furthermore, we insert the intracloud rule manually.

The

Function	URL subpath	Method	Input	Output
Add Intracloud rules	/mgmt/ intracloud	POST	IntracloudRuleForm	IntracloudRuleList

Table 2. Authorization endpoint (list not complete)

Authorization example

Let’s say we want to authorize the Service A (id:11) to access Service B(id:12). To be more specific, Service A wants to consume Service B’s “current time” service, defined with “current_time_service” definition(id:20).

The authorization template should be filled with the following parameters:

```
{
  "consumerId": 0,
  "providerIds": [
    0
  ],
  "interfaceIds": [
    0
  ],
  "serviceDefinitionIds": [
    0
  ]
}
```

→

```
{
  "consumerId": 11,
  "providerIds": [
    12
  ],
  "interfaceIds": [
    1
  ],
  "serviceDefinitionIds": [
    20
  ]
}
```

Authorization example

Add intracloud rule to the Authorization:

curl -v -s --insecure --cert-type P12 --cert {path to sysop certificate}:{passcode} -X POST -H "Content-Type: application/json" -d {intracloud form} https://localhost:8445/authorization/mgmt/intracloud

The request should return an IntracloudRuleList object:

```
{
  "count": 0,
  "data": [
   {
     "id": 0,
     "consumerSystem": {
       "id": 0,
       "systemName": "string",
       "address": "string",
       "port": 0,
… }
```

Orchestrator

The Orchestrator enables connections between service provider and service consumer. It provides late binding in runtime.

Systems can request the connection process through the Orchestrator.

The most important client endpoints of the Service Registry:

Function	URL subpath	Method	Input	Output
Echo	/echo	GET	-	OK
Orchestration	/orchestration	POST	ServiceRequestForm	Orchestration Response
Start store Orchestration by ID	/orchestration/{id}	GET	StoreEntryID	Orchestration Response

Table 3. Orchestrator client endpoints

Orchestrator example

To request orchestration process the consumer systems must fill the ServiceRequestForm[1]. The most important parameters are the following:

- requesterSystem: holds the information about the consumer.
- requestedService: hold the information about the providing service.
- securityRequirements: allowed security levels

[1] https://github.com/eclipse-arrowhead/core-java-spring#datastructures_servicerequestform

Orchestrator example

Requesting orchestration process:

curl -v -s --insecure --cert-type P12 --cert {path to consumer certificate}:{passcode} -X POST -H "Content-Type: application/json" -d @Orchestrator/orchestration.request.test https://localhost:8441/orchestrator/orchestration

The request should return the information about the provider service.

```
{
  "response": [
    {
      "provider": {
        "id": 12,
        "systemName": "string,
        "address": "string",
        "port": “string”,
        “authenticationInfo": …
…
}
```

How to progress through the demo?

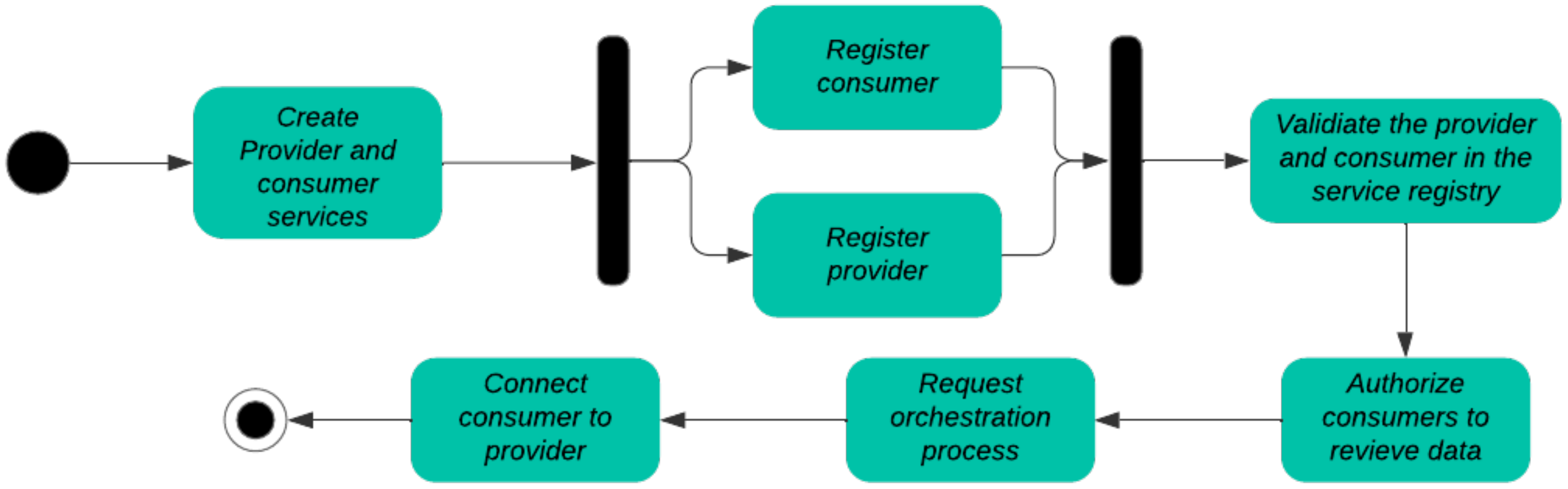
Create Provider and consumer services
Register consumer
Register provider
Validiate the provider and consumer in the service registry
Authorize consumers to revieve data
Request orchestration process
Connect consumer to provider

Figure 3. The Hello World demo activity diagram