Orchestration HTTP/TLS/JSON

Interface Design Description

Service ID: *"orchestrationl"*

Abstract
This document describes a HTTP/TLS/JSON variant of the Orchestration service.

ARTEMIS Innovation Pilot Project: Arrowhead
THEME [SP1-JTI-ARTEMIS-2012-AIPP4 SP1-JTI-ARTEMIS-2012-AIPP6]
[Production and Energy System Automation Intelligent-Built environment and urban infrastructure for sustainable and friendly cities]

Contents

1 Overview

This document describes the Orchestration Eclipse Arrowhead service, which provides Application Systems with orchestration information: where they need to connect to. The outcome of the Orchestration Service include rules that will tell the Application System what Service provider System(s) it should connect to and how (acting as a Service Consumer).

This document exists as a complement to the *Orchestration – Service Description* document. For further details about how this service is meant to be used, please consult that document. The rest of this document describes how to realize the Orchestration service using HTTP [1], TLS [2] and JSON [3], both in terms of its functions (Section 2) and its information model (Section 3).

2 Service Functions

This section lists the functions that must be exposed by the Orchestration service in alphabetical order. In particular, each subsection first names the HTTP method and path used to call the function, after which it names an abstract function from the Orchestration SD document, as well as input and output types. All functions in this section respond with the HTTP status code `200 Created` if called successfully. The error codes are, `400 Bad Request` if request is malformed, `401 Unauthorized` if improper client side certificate is provided, `500 Internal Server Error` if Service Registry is unavailable.

2.1 GET /orchestrator/echo

Interface: Echo
Output: StatusCodeKind

Called to check the core systems availability, as exemplified in Listing 1.

```
GET /authorization/echo HTTP/1.1

Got it!
```

Listing 1: An Echo invocation response.

2.2 POST /orchestrator/orchestration/

Interface: Orchestration
Input: ServiceRequestForm
Output: OrchestrationResponse

Called to start the orchestration process. 2.

Orchestrator can be used in two ways. The first one uses predefined rules (coming from the Orchestrator Store DB) to find the appropriate providers for the consumer. The second option is the dynamic orchestration in which case the core service searches the whole local cloud (and maybe some other clouds) to find matching providers.

Store Orchestration:

- requester system is mandatory,
- requested service and all the other parameters are optional,
- if requested service is not specified, then this service returns the top priority local provider of all services contained by the orchestrator store database for the requester system. if requested service is specified, then you have to define the service definition and exactly one interface (all other service requirements are optional). In this case, it returns all accessible providers from the orchestrator store database that provides the specified service via the specified interface to the specified consumer.

Dynamic Orchestration:

- requester system is mandatory,
- requested service is mandatory, but just the service definition part, all other parameters of the requested service are optional,
- all other parameters are optional

```
POST/orchestrator/orchestration HTTP/1.1

{
  "requesterSystem": {
    "systemName": "string",
    "address": "string",
    "port": 0,
    "authenticationInfo": "string"
  },
  "requestedService": {
    "serviceDefinitionRequirement": "string",
    "interfaceRequirements": [
      "string"
    ],
    "securityRequirements": [
      "NOT_SECURE", "CERTIFICATE", "TOKEN"
    ],
    "metadataRequirements": {
      "additionalProp1": "string",
      "additionalProp2": "string",
      "additionalProp3": "string"
    },
    "versionRequirement": 0,
    "maxVersionRequirement": 0,
    "minVersionRequirement": 0
  },
  "preferredProviders": [
    {
      "providerCloud": {
        "operator": "string",
        "name": "string"
      },
      "providerSystem": {
        "systemName": "string",
        "address": "string",
        "port": 0
      }
    }
  ],
  "orchestrationFlags": {
    "additionalProp1": true,
    "additionalProp2": true,
    "additionalProp3": true
  }
}
```

Listing 2: An Orchestration invocation with ServiceRequestForm payload.

Response of the call above:

```
{
  "response": [
    {
      "provider": {
        "id": 0,
        "systemName": "string",
        "address": "string",
        "port": 0,
        "authenticationInfo": "string",
        "createdAt": "string",
        "updatedAt": "string"
      },
      "service": {
        "id": 0,
        "serviceDefinition": "string",
        "createdAt": "string",
        "updatedAt": "string"
      },
      "serviceUri": "string",
      "secure": "TOKEN",
      "metadata": {
        "additionalProp1": "string",
        "additionalProp2": "string",
        "additionalProp3": "string"
      },
      "interfaces": [
        {
          "id": 0,
          "createdAt": "string",
          "interfaceName": "string",
          "updatedAt": "string"
        }
      ],
      "version": 0,
      "authorizationTokens": {
        "interfaceName1": "token1",
        "interfaceName2": "token2"
      },
      "warnings": [
        "FROM_OTHER_CLOUD", "TTL_UNKNOWN"
      ]
    }
  ]
}
```

Listing 3: An OrchestrationResponse

2.3 GET /orchestrator/orchestration/{id}

Interface: Start store Orchestration by ID
Output: OrchestrationResponse

If the consumer knows its' ID, it can used this service as shortcut for store-based orchestration when the service returns the top priority local provider of all services contained by the orchestrator store database for the requester system (identified by the ID) Listing 4.

```
GET /orchestrator/orchestration/{id} HTTP/1.1

{
  "response": [
    {
      "provider": {
        "id": 0,
        "systemName": "string",
        "address": "string",
        "port": 0,
        "authenticationInfo": "string",
        "createdAt": "string",
        "updatedAt": "string"
      },
      "service": {
        "id": 0,
        "serviceDefinition": "string",
        "createdAt": "string",
        "updatedAt": "string"
      },
      "serviceUri": "string",
      "secure": "TOKEN",
      "metadata": {
        "additionalProp1": "string",
        "additionalProp2": "string",
        "additionalProp3": "string"
      },
      "interfaces": [
        {
          "id": 0,
          "createdAt": "string",
          "interfaceName": "string",
          "updatedAt": "string"
        }
      ],
      "version": 0,
      "authorizationTokens": {
        "interfaceName1": "token1",
        "interfaceName2": "token2"
      },
      "warnings": [
        "FROM_OTHER_CLOUD", "TTL_UNKNOWN"
      ]
    }
  ]
}
```

Listing 4: A Store orchestration by ID invocation.

3 Information Model

Here, all data objects that can be part of the service calls associated with this service are listed in alphabetic order. Note that each subsection, which describes one type of object, begins with the *struct* keyword, which is meant to denote a JSON Object that must contain certain fields, or names, with values conforming to explicitly named types. As a complement to the primary types defined in this section, there is also a list of secondary types in Section 3.13, which are used to represent things like hashes, identifiers and texts.

3.1 struct InterfaceObject

This structure is used to describe an Interface Object.

Object Field	Value Type	Description
”id”	RandomID	ID.
”interfaceName”	Interface	Interface Name.
”createdAt”	DateTime	Created At.
”updatedAt”	DateTime	Updated At.

3.2 struct Metadata

A JSON Object which maps String key-value pairs.

3.3 struct OrchestrationResponse

This structure is used to describe an OrchestrationResponse Object.

Object Field	Value Type	Description
”provider”	Provider	Provider.
”service”	Service	Service.
”serviceUri”	URI	Service URI.
”secure”	SecureType	Secure Type.
”metadata”	Metadata	Metadata.
”interfaces”	Array<InterfaceObject>	Interfaces
”version”	Version	Version number
”authorizationTokens”	AuthorizationTokens	Authorization Tokens
”warnings”	Warnings	Warnings

3.4 struct Provider

This structure is used to describe a Provider Object.

3.5 struct Service

This structure is used to describe a Service Object.

Object Field	Value Type	Description
”id”	RandomID	ID.
”systemName”	Name	System Name.
”address”	String	Address.
”port”	Port	Port.
”authenticationInfo”	String	Authentication Info.
”createdAt”	DateTime	Created At.
”updatedAt”	DateTime	Updated At.

Object Field	Value Type	Description
”id”	RandomID	ID.
”serviceDefinition”	String	Service Definition.
”createdAt”	DateTime	Created At.
”updatedAt”	DateTime	Updated At.

Object Field	Value Type	Description
”requesterSystem”	RequesterSystem	Requester Systems own data.
”requestedService”	RequestedService	Service Definition.
”preferredProviders”	Array<PreferredProvider>	Created At.

3.6 struct ServiceRequestForm

This structure is used to describe a ServiceRequestForm Object.

3.7 struct PreferredProvider

This structure is used to describe a PreferredProvider Object.

Object Field	Value Type	Description
”providerCloud”	ProviderCloud	Provider Cloud.
”providerSystem”	ProviderSystem	Provider System.

3.8 struct ProviderCloud

This structure is used to describe a ProviderCloud Object.

Object Field	Value Type	Description
”operator”	Name	Name of the operator company.
”name”	Name	Name of the cloud.

3.9 struct ProviderSystem

This structure is used to describe a ProviderSystem Object.

Object Field	Value Type	Description
"systemName"	Name	System Name.
"address"	String	Address.
"port"	Port	Port.

3.10 struct RequesterSystem

This structure is used to describe a RequesterSystem Object.

Object Field	Value Type	Description
"systemName"	Name	System Name.
"address"	String	Address.
"port"	Port	Port.
"authenticationInfo"	String	Authentication Info.

3.11 struct RequestedService

This structure is used to describe a RequestedService Object.

Object Field	Value Type	Description
"serviceDefinitionRequirement"	String	Required Service Definition.
"interfaceRequirements"	Array<Interface>	List of required interfaces.
"securityRequirements"	Array<SecureType>	List of Secure Type.
"metadataRequirements"	Metadata	Required Metadata.
"versionRequirement"	Version	Version Requirement
"maxVersionRequirement"	Version	Maximum version
"minVersionRequirement"	Version	Minimum version

3.12 struct Warnings

A JSON Array which contains String values.
It can contain the following values:

- **FROM_OTHER_CLOUD** (if the provider is in an other cloud)
- **TTL_EXPIRED** (the provider is no longer accessible)
- **TTL_EXPIRING** (the provider will be inaccessible in a matter of minutes),
- **TTL_UNKNOWN** (the provider does not specified expiration time)

3.13 Primitives

As all messages are encoded using the JSON format [3], the following primitive constructs, part of that standard, become available. Note that the official standard is defined in terms of parsing rules, while this list only concerns syntactic information. Furthermore, the Object and Array types are given optional generic type parameters, which are used in this document to signify when pair values or elements are expected to conform to certain types.

JSON Type	Description
Value	Any out of Object, Array, String, Number, Boolean or Null.
Object <A>	An unordered collection of [String: Value] pairs, where each Value conforms to type A.
Array <A>	An ordered collection of Value elements, where each element conforms to type A.
String	An arbitrary UTF-8 string.
Number	Any IEEE 754 binary64 floating point number [4], except for *+Inf*, *-Inf* and *NaN*.
Boolean	One out of `true` or `false`.
Null	Must be `null`.

With these primitives now available, we proceed to define all the types specified in the Service Discovery Register SD document without a direct equivalent among the JSON types. Concretely, we define the Service Discovery Register SD primitives either as *aliases* or *structs*. An *alias* is a renaming of an existing type, but with some further details about how it is intended to be used. Structs are described in the beginning of the parent section. The types are listed by name in alphabetical order.

3.13.1 alias DateTime = String

Pinpoints a moment in time in the format of "YYYY-MM-DD HH:mm:ss", where "YYYY" denotes year (4 digits), "MM" denotes month starting from 01, "DD" denotes day starting from 01, "HH" denotes hour in the 24-hour format (00-23), "MM" denotes minute (00-59), "SS" denotes second (00-59). " " is used as separator between the date and the time. An example of a valid date/time string is "2020-12-05 12:00:00"

3.13.2 alias Interface = String

A String that describes an interface in *Protocol-SecurityType-MimeType* format. *SecurityType* can be SECURE or INSECURE. *Protocol* and *MimeType* can be anything. An example of a valid interface is: "HTTPS-SECURE-JSON" or "HTTP-INSECURE-SENML".

3.13.3 alias Name = String

A String that is meant to be short (less than a few tens of characters) and both human and machine-readable.

3.13.4 alias id = Number

An identifier generated for each Object that enables to distinguish them and later to refer to a specific Object.

3.13.5 alias RandomID = Number

An integer Number, originally chosen from a secure source of random numbers. When new RandomIDs are created, they must be ensured not to conflict with any relevant existing random numbers.

3.13.6 alias Name = String

A String that is meant to be short (less than a few tens of characters) and both human and machine-readable.

3.13.7 alias SecureType = String

A String that describes an the security type. Possible values are *NOT_SECURE* or *CERTIFICATE* or *TOKEN*.

3.13.8 alias URI = String

A String that represents the URL subpath where the offered service is reachable, starting with a slash ("/"). An example of a valid URI is "/temperature".

3.13.9 alias Version = Number

A Number that represents the version of the service. And example of a valid version is: 1.

4 References

[1] R. Fielding and J. Reschke, “Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing,” RFC 7230, 2018, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC7230

[2] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” RFC 8446, 2018, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC8446

[3] T. Bray, “The JavaScript Object Notation (JSON) Data Interchange Format,” RFC 7159, 2014, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC7159

[4] M. Cowlishaw, “IEEE Standard for Floating-Point Arithmetic,” *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, July 2019. [Online]. Available: https://doi.org/10.1109/IEEESTD.2019.8766229

5 Revision History

5.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1	2020-12-05	1.0.0		Szvetlin Tanyi

5.2 Quality Assurance

No.	Date	Version	Approved by
1	2021-01-28	4.3.0.	Jerker Delsing