orchestration

Service Description

Abstract
This document provides service description for the **orchestration** service.

Contents

1 Overview

This document describes the **orchestration** service, which provides runtime (late) binding between application systems. Its primary purpose is to find matching service instances according to the consumer's needs within an Eclipse Arrowhead Local Cloud (LC) and optionally, in other Arrowhead clouds. The matching service instances can be orchestrated with different strategies:

- **simple-store:** when the matching service instances are stored in a database as peer-to-peer rules that are maintained by higher entities and no actual service details are obtained from the Local Cloud;
- **flexible-store:** when the service requirements are stored in a database as service and provider attribute rules that are maintained by higher entities and actual service details are obtained from the Local Cloud;
- **dynamic:** when the matching service instances and their actual service details are discovered on the fly from the Local Cloud based on the service requirements specified by the initiator party (consumer system or higher entity).

The **orchestration** service contains the following operations:

- *pull* performs the orchestration process and returns the matching service instances;
- *subscribe* creates a subscription that can be triggered anytime to perform the orchestration process and push the matching service instances for the subscriptor (push orchestration);
- *unsubscribe* removes a subscription;

The rest of this document is organized as follows. In Section 2, we describe the abstract message operations provided by the service. In Section 3, we end the document by presenting the data types used by the mentioned operations.

1.1 How This Service Is Meant to Be Used

There are two ways to use this service:

- **Pull orchestration:** A consumer system consumes the *pull* operation in order to immediately obtain the matching service instances.
- **Push orchestration:** A consumer system consumes the *subscribe* operation in order to receive new orchestration results every time when a higher entity triggers a related orchestration process. Hence, the actual matching service instances are being sent to the consumer without its direct request.

Always the same orchestration process is being performed in case of both method (*pull* and *push*), however the behavior of the orchestration process itself differs between the orchestration strategies.

In case of **simple-store** strategy the orchestration process results in providing the corresponding service instance identifiers for all of the (or targeted) service definitions associated to the consumer system and stored in the rule database. Multiple rules with different priorities can exist for the same service definition and consuming system. Based on the provided service instance identifiers the consumer system has to perform every other required task to prepare for the successful service consumption. Hence, it has to lookup for all the actual service instance details (like access addresses, service and provider metadata, etc...) by using the *serviceDiscovery* service from the implementing system. Also, service instances are not being filtered

on the consumption permissions during the orchestration process and authorization tokens for the service consumptions (if necessary) are also should be requested by the consumer system via the *authorizationToken* service from the implementing system. (See the *serviceDiscovery* and *authorizationToken* SD documents.)

In case of **flexible-store** strategy the orchestration process results in providing the matching service instances, discovered based on the rules stored in the database for all of the (or targeted) service definitions that are associated to the consumer system. Multiple rules with different priorities can exist for the same service definition and consuming system. Service instances are retrieved with all the actual service and provider details (like access addresses, metadata, etc...) directly from the Local Cloud and are also filtered on the consumption permissions (if necessary). Authorization tokens for the service consumptions (if necessary) are generated in advance and provided within the orchestration results.

In case of **dynamic** strategy the orchestration process results in providing the matching service instances, discovered based on the given service requirements for a certain service definition. Service instances are retrieved with all the actual service and provider details (like access addresses, metadata, etc...) directly from the Local Cloud and are also filtered on the consumption permissions (if necessary). Authorization tokens for the service consumptions (if necessary) are generated in advance and provided within the orchestration results.

1.2 Important Delimitations

The consumer has to identify itself to use any of the operations.

1.3 Access policy

Available for anyone within the local cloud.

2 Service Operations

This section describes the abstract signatures of each operations of the service. The **orchestration** service is used to perform a *pull* orchestration and *subscribe*/*unsubscribe* for push orchestrations. In particular, each subsection names an operation, an input type and one or two output types (unsuccessful operations can return different structure), in that order. The input type is named inside parentheses, while the output type is preceded by a colon. If the operation has two output types, they are separated by a slash. Input and output types are only denoted when accepted or returned, respectively, by the operation in question. All abstract data types named in this section are defined in Section 3.

2.1 operation pull (OrchestrationRequest) : OrchestrationResponse / ErrorResponse

The behavior of this operation differs between the orchestration strategies:

In case of **simple-store** strategy the *pull* operation requires at least the consumer system's identity in order to perform the orchestration process. As a result of the orchestration process the consumer system receives back the matching service instance identifiers sorted by the service definition and priority. Targeting one specific service definition is also possible.

In case of **flexible-store** strategy the *pull* operation requires at least the consumer system's identity in order to perform the orchestration process. As a result of the orchestration process the consumer system receives back the matching service instances with all the details sorted by the service definition and priority. Targeting one specific service definition is also possible.

In case of **dynamic** strategy the *pull* operation requires at least the consumer system's identity and the targeted service definition. As a result of the orchestration process the consumer system receives back the matching service instances with all the details.

2.2 operation subscribe (OrchestrationSubscriptionRequest) : OrchestrationSubscriptionId / ErrorResponse

This operation requires at least the consumer's identity, the targeted service definition and a notify interface. The result of this operation is an identifier that refers to the created subscription record. Optionally a push orchestration also can be initiated at the same time.

2.3 operation unsubscribe (OrchestrationSubscriptionId) : OperationStatus / ErrorResponse

This operation requires the identifier pointing to the subscription record to be deleted.

3 Information Model

Here, all data objects that can be part of the **orchestration** service are listed and must be respected by the hosting System. Note that each subsection, which describes one type of object, begins with the *struct* keyword, which is used to denote a collection of named fields, each with its own data type. As a complement to the explicitly defined types in this section, there is also a list of implicit primitive types in Section 3.17, which are used to represent things like hashes and identifiers.

3.1 struct OrchestrationRequest

Field	Type	Mandatory	Description
authentication	Identity	yes	The requester of the operation.
orchestrationForm	OrchestrationForm	yes	Orchestration request details.

3.2 struct Identity

An Object which describes the identity of a system. It also contains whether the identified system has higher level administrative rights.

3.3 struct OrchestrationForm

Field	Type	Mandatory	Description
serviceRequirement	ServiceRequirement	yes	Details of the targeted service.
orchestrationFlags	Set<OrchestrationFlag>	no	Set of orchestration flags.
qosRequirements	QoSRequirementMap	no	Quality of service requirements map.
exclusivityDuration	Number	no	The interval the service wanted to be exclusive.

Note: *simple-store* strategy supports only to provide the `serviceRequirement` and `orchestrationFlags` fields.

3.4 struct ServiceRequirement

Field	Type	Mandatory	Description
serviceDefinition	ServiceName	yes/no	The required service definition name. Mandatory in case of **dynamic** strategy.
operations	List<OperationName>	yes/no	The required service operation names. Exactly one operation must be defined, when the following orchestration flags are true: `ONLY_INTERCLOUD`, `ALLOW_INTERCLOUD`, `ALLOW_TRANSLATION`
versions	List<Version>	no	The required service versions.
alivesAt	DateTime	no	The orchestrated service must be alive by this time.
metadataRequirements	List<MetadataRequirements>	no	The orchestrated service must meet at least to one of the specified metadata requirement.
interfaceTemplateNames	List<InterfaceName>	no	The orchestrated service must offer at least one from the specified interface template names.
interfaceAddressTypes	List<AddressType>	no	The orchestrated service must offer at least one from the specified interface address types.
interfacePropertyRequirements	List<MetadataRequirements>	no	The orchestrated service must offer at least one interface that meets with one of the specified property requirements.
securityPolicies	List<SecurityPolicy>	no	The orchestrated service must meet with one of the specified security policies.
preferredProviders	List<SystemName>	no	Provider system names specified here have priority.

Note: *simple-store* strategy supports only to provide the `serviceDefinition` field.

3.5 struct MetadataRequirements

A special Object which maps String keys to Object, primitive or list values, where

- Keys can be paths (or multi-level keys) which access a specific value in a Metadata structure, where parts of the path are delimited with dot character (e.g. in case of ”key.subkey” path we are looking for the key named ”key” in the metadata, which is associated with an embedded object and in this object we are looking for the key named ”subkey”).

- Values are special Objects with two fields: an operation (e.g. less than) and an actual value (e.g. a number). A metadata is matching a requirement if the specified operation returns true using the metadata value referenced by a key path as first and the actual value as second operands.
- Alternatively, values can be ordinary primitives, lists or Objects. In this case the operation is equals by default.

3.6 struct OrchestrationFlag

Specific String:Boolean pair to control the orchestration process. Possible values:

- `MATCHMAKING`:

  If `true`, orchestration process includes a matchmaking process in order to return only one matching service instance if any.

  If `false`, orchestration process returns all the matching service instance if any.

  Supporting strategies: *simple-store*, *flexible-store*, *dynamic*

- `ONLY_PREFERRED`:

  If `true`, orchestration process considers only those matching service instances that are provided by a preferred provider if any.

  If `false`, but preferred providers are specified and have matching service instances, then orchestration process considers only those service instances that are provided by a preferred provider. Otherwise, non-preferred providers are considered.

  Supporting strategies: *flexible-store*, *dynamic*

- `ONLY_EXCLUSIVE`:

  If `true`, orchestration process considers only those matching service instances that are allows exclusivity. It automatically results `MATCHMAKING:true` as well.

  If `false`, but exclusivity duration is specified and there are matching services with exclusivity allowed, then orchestration process considers only those service instances that allows exclusivity. Otherwise, service instances without exclusivity are considered.

  Supporting strategies: *flexible-store*, *dynamic*

- `ALLOW_INTERCLOUD`:

  If `true`, orchestration process considers matching service instances from neighbor clouds when there are no local hits. Orchestrating from neighbor clouds automatically results `MATCHMAKING:true`

  If `false`, orchestration process doesn't consider matching service instances from neighbor clouds when there are no local hits.

  Supporting strategies: *flexible-store*, *dynamic*

- `ONLY_INTERCLOUD`:

  If `true`, orchestration process considers matching service instances only from the neighbor clouds. It automatically results `MATCHMAKING:true` as well.

  If `false`, orchestration process considers matching service instances from the local could in first hand and only considers matching service instances from the neighbor clouds when `ALLOW_INTERCLOUD:true`.

  Supporting strategies: *flexible-store*, *dynamic*

- `ALLOW_TRANSLATION`:

  If `true`, orchestration process considers matching, but non-native service instances when there are no native hits. Matching, but non-native service instance means that all the requirements are fulfilled except the interface related requirements (protocol, data-format, etc.). It automatically results `MATCHMAKING:true` as well.

  If `false`, orchestration process considers only native matching service instances.

  Supporting strategies: *flexible-store*, *dynamic*

3.7 struct QoSRequirementMap

An Object which maps String keys String values.

3.8 struct OrchestrationResponse

Field	Type	Description
results	List<OrchestrationResult>	List of matching service instances.
warnings	List<String>	Warning message list.

3.9 struct OrchestrationResult

Field	Type	Description
serviceInstanceId	ServiceInstanceID	Unique identifier of the service instance.
providerName	SystemName	Unique identifier of the provider system.
serviceDefinition	ServiceName	Unique identifier of the service definition.
cloudIdentifier	CloudID	Unique identifier of the provider cloud.
version	Version	Version of the service instance.
aliveUntil	DateTime	The service instance is available until this time.
exclusiveUntil	DateTime	The service instance is reserved until this time.
metadata	Metadata	Additional information about the service instance.
interfaces	List<ServiceInterfaceDescriptor>	Available access interfaces of the service instance.

Note: *simple-store* strategy supports only to provide the `serviceInctanceId` field.

3.10 struct Metadata

An Object which maps String keys to primitive, Object or list values.

3.11 struct ServiceInterfaceDescriptor

Field	Type	Description
templateName	InterfaceName	The name of the interface template that describes the interface structure.
protocol	Protocol	The communication protocol of the interface.
policy	SecurityPolicy	The security of the interface.
properties	Metadata	Interface template-specific data.

3.12 struct ErrorResponse

Field	Type	Description
status	OperationStatus	Status of the operation.
errorMessage	String	Description of the error.
errorCode	Number	Numerical code of the error.
type	ErrorType	Type of the error.
origin	String	Origin of the error.

3.13 struct OrchestrationSubscriptionRequest

Field	Type	Mandatory	Description
authentication	Identity	yes	The requester of the operation.
orchestrationForm	OrchestrationForm	yes	Orchestration request details.
notifyInterface	NotifyInterface	yes	Interface details for sending push notifications.
duration	Number	no	The interval while the subscription is active.
trigger	Boolean	no	Whether or not a push orchestration should be initiated upon a successful subscription.

3.14 struct NotifyInterface

Field	Type	Mandatory	Description
protocol	Protocol	yes	Communication protocol to be used for sending notification.
properties	NotifyInterfacePropertyMap	yes	Interface properties belonged to the specified protocol.

3.15 struct NotifyInterfacePropertyMap

An Object which maps String keys String values.

3.16 struct OrchestrationSubscriptionId

Unique String identifier.

3.17 Primitives

Types and structures mentioned throughout this document that are assumed to be available to implementations of this service. The concrete interpretations of each of these types and structures must be provided by any IDD document claiming to implement this service.

Type	Description
AddressType	Any suitable type chosen by the implementor of service.
CloudID	A composite string identifier that is intended to be both human and machine-readable. It consists of the cloud name and the organization name, separated by a special delimiter character. Each part must follow PascalCase naming convention.
DateTime	Pinpoints a specific moment in time.
ErrorType	Any suitable type chosen by the implementor of service.
InterfaceName	A string identifier of an interface descriptor. Must following snake$_{c}asenamingconvention.$
List<A>	An *array* of a known number of items, each having type A.
Number	Decimal number.
Object	Set of primitives and possible further objects.
OperationName	A string identifier that is intended to be both human and machine-readable. Must following kebab-case naming convention.
OperationStatus	Logical, textual or numerical value that indicates whether an operation is a success or a failure. Multiple values can be used for success and error cases to give additional information about the nature of the result.
Protocol	A string representation of a communication protocol.
SecurityPolicy	Any suitable security policy chosen by the implementor of service.
ServiceInstanceID	A composite string identifier that is intended to be both human and machine-readable. It consists of the instance's provider name, service definition and version, each separated by a special delimiter character. Each part must follow its related naming convention.
ServiceName	A string identifier that is intended to be both human and machine-readable. Must following camelCase naming convention.
Set<A>	A collection of items that contains no duplicate elements and each having type A.
String	A chain of characters.
SystemName	A string identifier that is intended to be both human and machine-readable. Must following PascalCase naming convention.
Version	Specifies a service instance version. Version must follow the Semantic Versioning.

4 References

5 Revision History

5.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1	YYYY-MM-DD	5.0.0		Xxx Yyy

5.2 Quality Assurance

No.	Date	Version	Approved by
1	YYYY-MM-DD	5.0.0	