ServiceOrchestration Core System

System Description

Abstract
This document provides system description for the **ServiceOrchestration Core System**.

Contents

1 Overview

This document describes the ServiceOrchestration Core System, which exists to find matching providers for the consumer's specification within an Eclipse Arrowhead Local Cloud (LC) and optionally, in other Arrowhead clouds by collaborating with other Core/Support Systems. This can be achieved by using stored matching rules or applying a more dynamic strategy using the Service Registry Core System. This recommended system may provide the data storage functionality for the information related to matching rules and/or provider reservation.

The rest of this document is organized as follows. In Section 1.1, we reference major prior art capabilities of the system. In Section 1.2, we describe the intended usage of the system. In Section 1.3, we describe fundamental properties provided by the system. In Section 1.4, we describe delimitations of capabilities of the system. In Section 2, we describe the abstract services produced by the system. In Section 3, we describe the security capabilities of the system.

1.1 Significant Prior Art

The strong development on cloud technology and various requirements for digitisation and automation has led to the concept of Local Clouds (LC).

"The concept takes the view that specific geographically local automation tasks should be encapsulated and protected." [1]

A service orchestration system is a central component in any Service-Oriented Architecture (SOA). In applications, the use of SOA for a massive distributed System of Systems requires orchestration. It is utilised to dynamically allow the re-use of existing services and systems in order to create new services and functionality.

There are some key differences, even on conceptual level, between the previous versions (Orchestrator 4.6.x) and this version:

- The previous versions were named the system as Orchestrator. Because this expression has a different meaning in some related domains, it is decided that the current version uses the name ServiceOrchestration to avoid confusion.
- The previous versions contained three (or two in earlier versions) kind of orchestration strategies: a store containing simple, peer-to-peer rules, an other store containing more flexible rules and a dynamic orchestration method which used the live data of the ServiceRegistry to achieve its goal. The current version separates the three strategies into three different ServiceOrchestration systems (but with the same orchestration service operations), and the Local Cloud's administrator can decide which strategy want to support for their use case.
- There was no data storage separation requirement: the Orchestrator's data storage was interconnected to other systems' storage. In the current version, data storage separation is mandatory.
- You could only create an orchestration rule if all related entities (consumer system, provider system, service definition and interface) are already existed (in the ServiceRegistry's data storage). The current version supports rules referencing non-existent entities for future usage.
- Only an administrator were able to add and remove rules (in case of simple store orchestration). The current version allows any system (only those with the proper permissions, of course) to add and remove orchestration rules.
- Only the orchestration pull was supported in which the consumer could start an orchestration process for itself. The current version also supports orchestration push: the consumers can subscribe to a service orchestration (or an other support/application system can subscribe them) and after the subscription and

whenever a system notifies the Service Orchestration system, it performs the orchestration for the related subscribers.

- The QualityOfService (QoS) Manager component was embedded into the Orchestrator (only QoS data comes from a support system). The current version moves these functionalities into a separate support system (which also be responsible to collect and store QoS data).
- X.509 certificate trust chains was used as authentication mechanism. The current version can support any type of authentication methods by using a dedicate Authentication Provider core system.

1.2 How This System Is Meant to Be Used

ServiceOrchestration is a recommended core system of Eclipse Arrowhead Local Cloud and is responsible for finding and pairing service consumers and providers.

There are two ways to use the offered functionality:

- An application that want to consume a service should ask the ServiceOrchestration to find one or more accessible providers that meet the necessary requirements. The ServiceOrchestration returns the information that the application needs to consume the specified service.
- An application can subscribe for orchestration with the necessary requirements (or can be subscribed by an other application/support system). The ServiceOrchestration performs the orchestration process and returns the information on the specified channel. Additionally, whenever a system notifies the ServiceOrchestration system, it does the orchestration again and returns the updated information the the subscriber.

The *information* that is provided to the consumer whenever an orchestration is done depends on the strategy that the ServiceOrchestration system has implemented.

- In the case of simple store orchestration, the response contains service instance names. The consumer can use these names to query the ServiceRegistry and acquire access information of the service it should consume.
- In the case of flexible store orchestration or dynamic orchestration, the ServiceOrchestration system has to contact the ServiceRegistry to perform the orchestration so in the response it can return everything what is needed for the consumer to perform a service operation consumption.

1.3 System functionalities and properties

1.3.1 Functional properties of the system

ServiceOrchestration solves the following needs to fulfill the requirements of orchestration.

- Enables the application and Core/Support systems to find the appropriate providers to consume their services.
- Enables the application and Core/Support systems to subscribe/unsubscribe to repeated orchestration (orchestration push).
- Enables other application and Core/Support systems to notify the ServiceOrchestration system to re-orchestrate for the related subscribers.

- Enables other application and Core/Support systems to subscribe/unsubscribe consumers to repeated orchestrations.

1.3.2 Non functional properties of the system

If an Authentication Provider (AP) is present in the Local Cloud, the ServiceOrchestration system will use AP's service(s) to verify a requester system before responding to its request.

If flexible store or dynamic orchestration strategy is used, the following are true as well:

- (*Condition*: ConsumerAuthorization system is present in the Local Cloud): if the consumer is not authorized to use a service provider the orchestration service removes the appropriate provider from the response;
- (*Condition*: Authorization system is present in the Local Cloud): orchestration service automatically adds every necessary tokens to the response (if the related provider requires it);
- (*Condition*: Gatekeeper system is present in the Local Cloud): inter-cloud orchestration is possible;
- (*Condition*: Gatekeeper and Gatepath system is present in the Local Cloud): inter-cloud orchestration is possible even between two closed local clouds and the necessary communication tunnel will be built during the orchestration process;
- (*Condition*: QualityOfService Evaluator system is present in the Local Cloud): during orchestration Quality-of-Service requirements can be considered;
- (*Condition*: TranslationManager system is present in the Local Cloud): protocol and data model translation can be used to fulfill orchestration requirements.

1.3.3 Data stored by the system

In order to achieve the mentioned functionalities, ServiceOrchestration is capable to store the following information set:

- **Simple orchestration rules** (*only used in the simple store orchestration strategy*): A rule consists a consumer system name, a service definition name, a service instance name and a priority. Multiple service instance can assign to the same consumer-service definition pair as long as the priority is different.
- **Flexible orchestration rules** (*only used in the flexible store orchestration strategy*): A rule consists a consumer system name and/or consumer system metadata requirements, a provider system name (with cloud identification data, if the provider is in an other cloud) and/or provider system metadata requirements, a service definition name, an optional service interface requirement, optional service metadata requirements, optional minimum and maximum service version requirements and an optional priority.
- **Orchestration locks**: A storage to manage ongoing provider reservations or manual orchestration locks. It consists of a service instance ID, the owner system name and a timestamp which marks the end of lock.
- **Orchestration history**: A storage to save data about the performed orchestration processes. It consists of an orchestration type, a requester system name, a target system name, a service definition and a timestamp.

1.4 Important Delimitations

- There are three orchestration strategies (for now). A ServiceOrchestration implementation must support only one of those (but it may support multiple strategies). The administrator of the Local Cloud can decide what kind of orchestration strategy is used in the cloud, a consumer can not.
- If the Local Cloud does not contain an Authentication system, there is no way for the ServiceOrchestration to verify the requester system. In that case, the ServiceOrchestration will consider the authentication data comes from the requester as valid.
- If the Local Cloud does not contain a ServiceRegistry, then ServiceOrchestration system with simple store strategy can be used. The two other strategies need the continuous support of the ServiceRegistry system to provide their services.
- The flexible store and dynamic orchestration strategies need other core/support systems to provide their full functionalities, see details above.

2 Services produced

2.1 service orchestration

The purpose of this service is to find information about providers that meet the requirements. It also provided subscription functionality to repeated orchestrations (orchestration push). The service is offered for both application and Core/Support systems.

2.2 service orchestrationPushManagement

The purpose of this service is to manage orchestration push subscriptions in bulk. It also allows to signal the Service Orchestration system to orchestrate to related subscribers. The service is offered for both application and Core/Support systems.

2.3 service simpleStoreManagement

Recommended service (and only if the ServiceOrchestration system is using the simple store strategy). The purpose of this service is to add, remove and query simple store orchestration rules in bulk. The service is offered for Core/Support systems.

2.4 service flexibleStoreManagement

Recommended service (and only if the ServiceOrchestration system is using the flexible store strategy). The purpose of this service is to add, remove and query flexible store orchestration rules in bulk. The service is offered for Core/Support systems.

2.5 service orchestrationLockManagement

The purpose of this service is to add, remove and query active orchestration locks. An orchestration lock can be made automatically during the orchestration process in order to reserve an actual service (only if the Service Orchestration system is using the dynamic or flexible store strategy) or manually if a higher entity with management access has any reason to prevent a service from being orchestrated. The service is offered for Core/Support systems.

2.6 service orchestrationHistoryManagement

Recommended service. Its purpose is to give information about the orchestration processes performed by the system. The service is offered for Core/Support systems.

2.7 service monitor

Recommended service. Its purpose is to give information about the provider system. The service is offered for both application and Core/Support systems.

3 Security

For authentication, the ServiceOrchestration utilizes an other core system, the Authentication system's service to verify the identities of the requester systems. If no Authentication system is deployed into the Local Cloud, the Service Orchestration trusts the requester system self-provided identity.

For authorization, the system uses an other Core System, the ConsumerAuthorization system to decide whether a consumer can use its services or not. If the ConsumerAuthorization Core System is not present in the Local Cloud, then the ServiceOrchestration allows for anyone in the Local Cloud to use its services. The following service operations can always be used without any authorization rules:

- *orchestration* service's *pull* operation,
- *orchestration* service's *subscribe* operation,
- *orchestration* service's *unsubscribe* operation.

Furthermore, if the ConsumerAuthorization system is deployed in the Local Cloud, a ServiceOrchestration system with flexible store or dynamic orchestration strategy, will use the appropriate service to check whether the consumer can consume the required service of a specific provider during the orchestration process.

The implementation of the ServiceOrchestration can decide about the encryption of the connection between the ServiceOrchestration and other systems.

4 References

[1] J. Delsing and P. Varga, *Local automation clouds*. Boca Raton: Taylor & Francis Group, 2017, p. 28. [Online]. Available: https://doi.org/10.1201/9781315367897

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	