DynamicServiceOrchestration Core System

System Description

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

Contents

1 Overview

This document describes the DynamicServiceOrchestration 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 various strategies, but the DynamicServiceOrchestration Core System implements the dynamic orchestration strategy. This recommended system may provide the data storage functionality for the information related to 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. s

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 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 Service Registry to achieve its goal. The current version separates the three strategies into three different systems (but with the same orchestration service operations), and the Local Cloud's administrator can decide which strategy to support for their use case.
- 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 <Strategy>ServiceOrchestration to avoid confusion.
- 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.
- 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 DynamicServiceOrchestration system, it performs the orchestration for the related subscribers.
- The Quality-of-Service (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 dedicated Authentication Core system.

1.2 How This System Is Meant to Be Used

DynamicServiceOrchestration 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 DynamicServiceOrchestration to find one or more accessible providers that meet the necessary requirements. The DynamicServiceOrchestration 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). Whenever a system notifies the DynamicServiceOrchestration system, it does the orchestration and sends the orchestration information to the subscriber on the specified channel. Optionally, subscribers can ask that an orchestration process run and return immediately after the subscription.

The *information* that is provided to the consumer whenever an orchestration is done depends on the orchestration strategy. In case of dynamic orchestration strategy, the DynamicServiceOrchestration system has to contact with the ServiceRegistry (and optionally with other Core/Support systems) to perform the orchestration so in the response it can return everything what is needed for the consumer to perform a service operation consumption (access details, authorization tokens, etc...).

1.3 System functionalities and properties

1.3.1 Functional properties of the system

DynamicServiceOrchestration 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 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

- (*Condition*: Authentication system is present in the Local Cloud: the DynamicServiceOrchestration system will use Authentication system's *identity* service to verify the requester system before responding to its request.
- (*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*: ConsumerAuthorization 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 and Gatepath system is present in the Local Cloud): inter-cloud orchestration is possible between two Local Clouds and the necessary communication tunnel will be built during the orchestration process;
- (*Condition*: Any system with Quality-of-service evaluation capabilities is present in the Local Cloud): during orchestration Quality-of-Service requirements can be considered;
- (*Condition*: TranslationManager system is present in the Local Cloud): interface 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, DynamicServiceOrchestration is capable to store the following information set:

- **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.
- **QoS results**: A storage to save quality of service evaluation results to the performed orchestration processes, when QoS is supported and enabled.

1.4 Important Delimitations

- If the Local Cloud does not contain an Authentication system or does not using X.509 certificates, there is no way for the DynamicServiceOrchestration to verify the requester system. In that case, the DynamicServiceOrchestration will consider the authentication data that comes from the requester as valid.
- If the Local Cloud does not contain a ServiceRegistry, then DynamicServiceOrchestration system is not able to perform the orchestration process.
- To take full advantage of its functionality, other Core/Support systems are required to being deployed.

2 Services produced

2.1 service serviceOrchestration

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 serviceOrchestrationPushManagement

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

2.3 service serviceOrchestrationLockManagement

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 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.

2.4 service serviceOrchestrationHistoryManagement

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

2.5 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 DynamicServiceOrchestration utilizes an other Core/Support 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 DynamicServiceOrchestration 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 DynamicServiceOrchestration may allow for anyone in the Local Cloud to use its services. The following service operations can always be used without any authorization rules:

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

Furthermore, if the ConsumerAuthorization system is deployed in the Local Cloud, the DynamicServiceOrchestration system 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 DynamicServiceOrchestration can decide about the encryption of the connection between the DynamicServiceOrchestration 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	5.2.0		Xxx Yyy

5.2 Quality Assurance

No.	Date	Version	Approved by
1	YYYY-MM-DD	5.2.0	