Orchestrator System Description

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]

1 System Description and Overview

The Orchestrator provides runtime (late) binding between Application Systems.

The primary purpose for the Orchestrator System is to provide 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). Such orchestration rules include:

- Accessibility information details of a Service provider (e.g network address and port),
- Details of the Service instance within the provider System (e.g. base URL, IDD specification and other metadata),
- Authorization-related information (e.g. access token and signature),
- Additional information that is necessary for establishing connection.

This orchestration rule information can reach the given Application System (consumer) in two different ways: the System itself can request it ("pull") or the Orchestrator itself can update the System when it is needed ("push method"). However, in both cases, there shall be an underlying, hidden process (*"orchestration process"*), which ensures the consistence of state between the various Core Systems.

In G3.2, only the pull method is supported and the Orchestrator shall negotiate with the other Core Systems while trying to facilitate the new service request (or trying to push a new status). This is necessary for the following cases and requirements (basically, when ad hoc, unsupervised connections are not allowed):

- When accountability is required for all Systems in the Local Cloud: connections cannot be established without the knowledge, approval and logged orchestration events of the Core Systems ("central governance").
- QoS and resource management reasons: ad hoc peer-to-peer connections cannot be allowed in certain managed networks and deployment scenarios. Every connection attempt shall be properly authorized and its QoS expectations (resource reservations) handled.
- Inter-Cloud orchestration can only happen via negotiations between the two Core System sets. Ad hoc inter-cloud connections shall not be allowed in the Arrowhead framework.

In these cases, when the Orchestrator is the sole entry point to establishing new connections within the Local Cloud, Application Systems do not have the possibility to skip any of the control loops with all the appropriate Core Systems. When such security and safety concerns are not present, the orchestration process might be cut back or these interactions between Core Systems might be limited. Within G3.2, this is not the primary use case, but it is allowed. With the proper self-implemented (modified) and a self-compiled Orchestrator can fit the deployment best.

Therefore, the Orchestrator provides two core Services and may consume many other ones, but at least two – again, depending on its deployment. Figure 1 depicts the mandatory and optional interfaces of this System.

In here, the provided Services are:

1. Orchestration Service
2. OrchestrationStoreManagement Service (primarily used by HMI and Plant Description Engine)

Note: G3.2 Milestone 2 currently doesn't implement the Management Service, it will be release in the Milestone 3 version. As a temporary solution, manual modifications in the MySQL database is advised.

Meanwhile the consumed Services can vary, depending on the instantiation/installation of this System. For example, the Orchestrator can utilize the services of:

1. ServiceDiscovery Service from the ServiceRegistry,
2. AuthorizationControl Service from the Authorization System,

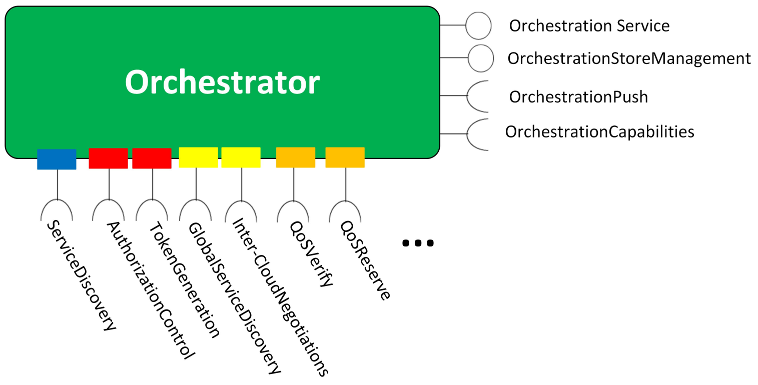

Figure 1: Overview on the Orchestrator System

3. TokenGeneration Service from the Authorization System,
4. GlobalServiceDiscovery from the Gatekeeper,
5. Inter-CloudNegotiations from the Gatekeeper,
6. QoSVerify from the QoS Manager,
7. QoSReserve from the QoS Manager,
8. Logging services from other supporting Systems, e.g. Historian,
9. and any other service from core systems that are necessary to settle during orchestration.

The Orchestrator mainly consumes services from other Core Systems in order to fulfill its primary functionality: provide connection targets for Application Systems in a secure and resource managed manner – hence build an SoS.

During this *orchestration process* the Orchestrator either facilitates a service request from an Application System or processes a system-of-systems (SoS) level choreography push from the Plant Description Engine ("Choreographer"). For the latter case, the Orchestrator System consumes the OrchestrationPush from affected Application Systems in order to deliver a renewed set of connection rules to them.

Within the Orchestrator, there is a database which captures design time bindings between Application Systems, the *Orchestration Store*. Operators of the Cloud and other System-of-Systems designer tools ("SoS Choreographers") are allowed to modify the rules stored in the Orchestration Store, other generic Application Systems are not.

The ServiceDiscovery Service is used to publish the Orchestration Service in the Service Registry. This Service is also used to query the Service Registry and fetch (metadata) information on other Application Systems.

The Services of the Authorization System can be used to verify access control and implement other security-related administration tasks.

The Services of the Gatekeeper can be utilized when inter-Cloud collaboration, servicing is required.

The Services of the QoS management System can be used to manage device, network and service-level Quality of Service agreements and configurations.

2 Use-cases

For the Orchestrator System, the primary scenario is to provide Application Systems with orchestration information upon request ("*service request*"). The outcome ("*orchestration response*") include orchestration rules that will tell the Application System what service provider(s) it should connect to and how.

An alternative, secondary version of this scenario involves the same information, however, provided by a connection initialized by the Orchestrator, rather than the Application Service itself (”orchestration push”). This is used to relay changes made in the Orchestration Store to the Application Systems (”changes information exchange setup within the SoS”).

Another scenario is when the Orchestration Store (that stores design time orchestration-related information) of the Orchestrator is being configured via an HMI or via the Plant Description Engine (SoS Choreographer) by the operators of the Local Cloud.

Table 1: Use case 1

Service Request From Application System	
ID:	ORCH-PULL
Brief description:	An Application System requests a Service.
Primary actors:	Service Consumer System
Secondary actors:	- the other Core System instances of the Local Cloud - the Core Systems instance of another Local Cloud (in case of inter-Cloud orchestration)
Preconditions:	-
Main flow:	1- The Application System requests orchestration. 2- The Orchestrator System begins the orchestration process with the other Core Systems. 3- The Orchestrator System responds to the Application System based on the request.
Postconditions:	-

Table 2: Use case 2

Orchestration information pushed to Application System	
ID:	ORCH-PUSH
Brief description:	The Orchestrator pushes new information on Application Systems.
Primary actors:	Orchestrator
Secondary actors:	the other Core Systems instances of the Local Cloud
Preconditions:	There was a change made in the Orchestration Store.
Main flow:	1- The Orchestrator detects a change in the Orchestration Store., 2- The Orchestrator begins the orchestration process with the other Core Systems for every change in the Store. 3- The orchestrator pushes new connection rules to the Application Systems based on the new Store entry.
Postconditions:	-

Table 3: Use case 3

Orchestration information pushed to Application System	
ID:	ORCH-PUSH
Brief description:	The Orchestrator pushes new information on Application Systems.
Primary actors:	Orchestrator
Secondary actors:	the other Core Systems instances of the Local Cloud
Preconditions:	There was a change made in the Orchestration Store.
Main flow:	1- The Orchestrator detects a change in the Orchestration Store., 2- The Orchestrator begins the orchestration process with the other Core Systems for every change in the Store. 3- The orchestrator pushes new connection rules to the Application Systems based on the new Store entry.
Postconditions:	-

3 Behaviour Diagrams

The orchestration process consists of negotiations with other Core Systems, as Fig.2 and 3 depict. To be elaborated.

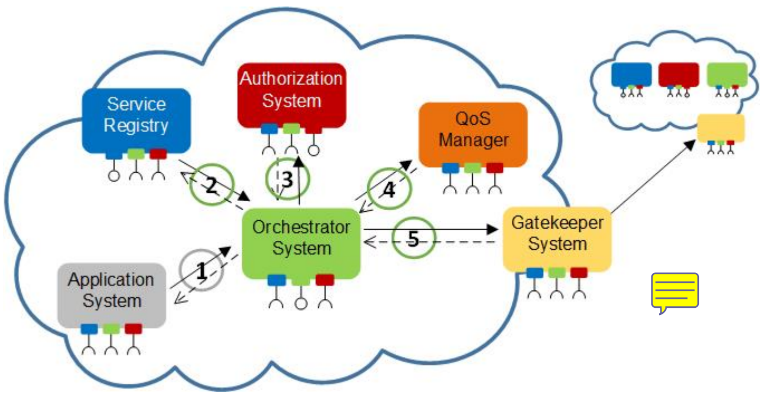

Figure 2: Interaction between Core Systems during the orchestration process

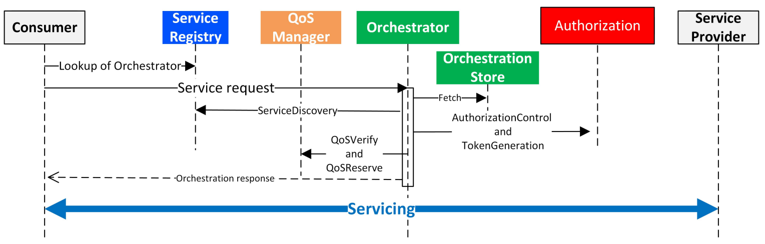

Figure 3: Interaction between Core Systems during the orchestration process

4 Application services

This section details the Services the Orchestrator provides and might consume.

4.1 Produced Services

Service Name	IDD path
Orchestration Service	-
OrchestrationManagement	-

4.2 Consumed Services

le 1 Pointers to IDD documents

Service Name	Functionality
ServiceDiscoveryM2	Fetch available and suitable Service Providers
AuthorizationControl	Check access rights
TokenGeneration	Generate access tokens
QoSVerify	Check if the requested QoS level can be met
QoSReserve	Reserve resource reservations if necessary
GSD-Init	Discover and look up Service in other Clouds
ICN-Init	Start negotiations with another Cloud

5 Security

As a mandatory Core System, the Orchestrator has to be protected at all costs. As this System is a trusted management layer entity – a decision maker – within its Local Cloud, its responses are executed by Application Systems. Therefore, the protection of the Orchestrator is of greate essence. In case it is compromised, intruders might be able to reconfigure the general operations of the network. CIAA analysis required for all deployment separately.

5.1 Security Objectives

The Orchestrator's Services have to be available all times.

The orchestration information provided by this Core System is unique and confidential to its recipient.

The message from the Orchestrator has to maintain its integrity during transmission.

5.2 Assets

The Orchestration Store poses the only persistent asset of this System. All other assets important in the orchestration process are related to other Core Systems.

5.3 Non-technical Security Requirements

Scalability of the Orchestrator's Services shall be exampled and set. This issue relates to the maximum practical size of a Local Cloud.

Error/event handling description is required. How do we detect and then handle when a servicing instance breaks up and requires re-orchestration (e.g. because Service Provider goes offline)? How do we propagate this information towards the QoS Manager and other Core Systems? Clear description of the usage for Event Handler and Historian is required.

6 References

7 Revision history

7.1 Amendments

No.	Date	Version	Subject of amendments	author
1	2017. 09. 19.	0.1	Initial	Csaba Hegedus

7.2 Quality Assurance

No.	Date	Version	Approved by