The Gatekeeper System

System Design Document

**Abstract**

This document describes the Gatekeeper automation support Core System.

1. Overview of the Gatekeeper System

Table 1 System Information

|  |  |
| --- | --- |
| **Name** | Gatekeeper System |
| **Owner** | Csaba Hegedűs hegeduscs@aitia.ai |

This System provides two Core Services:

* Global Service Discovery (GSD)
* Inter-Cloud Negotiations (ICN)

These Services are part of the inter-Cloud orchestration process and peferably these Services are not available for Application Systems.

*Note:* *Self-orchestrating[[1]](#footnote-1) Application Systems are currently not in scope, however they are not excluded from the Arrowhead framework. They are supported at high-level design, implementations have not been published yet. Therefore, the internal Services of the Gatekeepers are not public for Application Systems, hence not documented as such.*

The first is the Global Service Discovery (GSD) process, which aims at locating adequate service offerings in neighboring Clouds. The second is the Inter-Cloud Negotiations (ICN) process, in which mutual trust is established between two Clouds and the actual connection between endpoints is then built up. Working together with the Orchestrators of both Clouds, at the end a servicing instace can be created.

These Services each have two sets of interfaces:

* Provided intra-Cloud to the Orchestrator
* Provided inter-Cloud among the Gatekeepers

Currently, there is a hardwired list of other Local Clouds that are known to Gatekeeper. This ”neighborhood” is stored in the MySQL database of the Gatekeeper module, in the ”neighborhood” table.

1. Use-cases

|  |
| --- |
| **Name of the Use-case** |
| **ID**: GSD-1 |
| **Brief description**:  The Gatekeeper is tasked to find a Service in other Local Clouds. |
| **Primary actors**:  Orhestrator |
| **Secondary actors**:  Other Gatekeepers |
| **Preconditions**:  Orchestration process was started by an Application System. |
| **Main flow**:   1. The Orchestrator consumes the GSD Initialization Service of its local Gatekeeper. 2. The Gatekeeper locates other Gatekeepers in an arbitrary method (e.g. hardwired list, inter-Gatekeeper discovery protocol, etc.). 3. The Gatekeeper queries the other Gatekeepers by consuming their GSD Poll Service. 4. These Gatekeepers verify whether they could theoretically facilitate this request and answer with in a “yes/no” manner. 5. The requester Gatekeeper collects these answers and respond via the GSD Initialization Service to its Orchestrator. |
| **Postconditions**:  The Orchestrator has a list of other Local Clouds that can provide the Service we are looking for. |
| **Alternative flows**:  In theory, the GSD Initialization Service could be used by Application Systems themselves. |

|  |
| --- |
| **Name of the Use-case** |
| **ID**: ICN-1 |
| **Brief description**:  The Gatekeeper is tasked to start negotiating with another Cloud. . |
| **Primary actors**:  Orhestrator |
| **Secondary actors**:  Other Gatekeeper and other Orchestrator from the second Cloud. |
| **Preconditions**:  Orchestration process was started by an Application System. The GSD process has ended the the requester Orchestrator has chosen a partnering Cloud -where it wants to connect to. |
| **Main flow**:   1. The Orchestrator consumes the ICN Initialization Service of its local Gatekeeper. 2. The Gatekeeper locates the other Gatekeepers and consumes its ICN Proposal service. 3. The secondary Gatekeeper validates the AuthorizationControl and requests Orchestration from its own Orchestrator. 4. The secondary Orhestrator responds to the secondary Gatekeeper with a Service instance that can produce the Service. 5. The secondary Gatekeeper responds to the primary, requester Gatekeeper. 6. Additional administrative tasks are executed (e.g. configuration of the Gateway modules, from M3 release and up). 7. The primary, requester Orchestrator is receiving the response via the ICN initialization service. |
| **Postconditions**: |
| **Alternative flows**: |

The inter-Cloud orchestration process is shown on Figure 1. This Fig. details how this process is initiated by the Orchestrator of Cloud 1 and how Cloud 2 is contacted. At the end, Orchestrator 2 creates orchestration and this configuration is passed back to the original requester Consumer.

This Fig. includes what messages are sent, information shared between the entities.



**Figure 1. Inter-Cloud orchestration in details**

2. Application services

# Produced Services

|  |  |
| --- | --- |
| Service | Description |
| Global Service Discovery Init | Submitted by the local Orchestrator |
| Global Service Discovery Poll | Submitted by a Gatekeeper to others in the neighborhood |
| Inter-Cloud Negotiations Init | Submitted by the local Orchestrator |
| Inter-Cloud Negotiations Proposal | Submitted by a Gatekeeper to another one. |

# Consumed Services

|  |  |
| --- | --- |
| Service | Description |
| ServiceDiscovery | Only available for Core Systems. |
| AuthorizationControlM2 | Only available for the Gatekeeper. |
| OrchestrationM2 Service | Service Request Form is posted with special settings. |
| Gateway initialization service. | TBD. Not available as per M2 release. |

1. Security

The Gatekeeper is the only externally available Core System. Hence, its security is of essence. The intra-Cloud interfaces of its Services should only be accessed by its local Orchestrator.

For the GSD Poll inter-Cloud service, no security is required.

For the ICN Proposal inter-Cloud service, secure connection is required, since orchestration is done through it resulting in Application System behaviour changes.

*Note: In the current reference implementation, unauthorized access is not rejected, only logged for debugging purposes.*

1. Internal Structure

This module is a simple Java jar executable. It uses the config folder contents, where the configuration files are there. The code includes the following classes:

* GatekeeperMain: starts the HTTP and/or the HTTPS servers
* GatekeeperResource: offers the Services that are dedicated towards other Gatekeepers or its own local Orchestrator module
* AccessControlFilter: implements CN-based access control when the HTTPS (secure) server is started (e.g. the GSD-Init Service

# Usage

Start the module with the following command line arguments:

* ”-m <mode>”: selects whether simple HTTP (”insecure”) or HTTPS (”secure”) or both servers are needed (”both”)
* ”-d”: starts the module in daemon mode

1. References
2. Revision history

# Amendments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Date | Version | Subject of Amendments | Author |
| 1 | 2016-02-29 | 0.1 | First draft | Csaba Hegedűs |
| 2 | 2016-09-14 | M2 | Updated and corrected | Csaba Hegedűs |

# Quality Assurance

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Date | Version | Approved by |
| 1 |  |  |  |
| 2 |  |  |  |

1. Self-orchestrating Application Systems implement all Core Service interfaces and capable of running an orchestration process on their own – without the use of the central Orchestrator. [↑](#footnote-ref-1)