The DataManager System

System Design Document

**Abstract**

This document describes the DataManager support Core System.

1. Overview of the DataManager System

Table 1 System Information

|  |  |
| --- | --- |
| **Name** | DataManager System |
| **Owner** | Jens Eliasson, jens.eliasson@ltu.se |

This System provides two Services:

* Historian (HS)
* Proxy (PS)

The first service is the Historian Service (HS), which provides features for storing and retrieving sensor data and generic files. The second service is the Proxy Service (PS), which provides features for mailbox-like behavior where low-power, e.g. sleepy devices, systems can push messages to during short periods of online time. Clients can afterwards fetch these messages at any time.

1. Use-cases

|  |
| --- |
| **Name of the Use-case** |
| **ID**: PS-1 |
| **Brief description**:  The Proxy service is used to cache data for a low-power IoT device during times of offline. |
| **Primary actors**:  Low-power (sleepy) device, a Proxy service and a client. |
| **Secondary actors**:  n/a |
| **Preconditions**:  Arrowhead Core services such as ServiceRegistry, Authorization and Orchestration must be started |
| **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**:  n/a |
| **Alternative flows**:  The Historian service can also be used for the same purpose, albeit with higher overhead and latency. |

|  |
| --- |
| **Name of the Use-case** |
| **ID**: ICN-1 |
| **Brief description**:  The Gatekeeper is tasked to start negotiating with another Cloud. . |
| **Primary actors**:  Orchestrator |
| **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 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 an 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 |
| Historian | To be used in a local cloud |
| Proxy | To be used in a local cloud |

# Consumed Services

|  |  |
| --- | --- |
| Service | Description |
| ServiceRegistry | To publish the Historian and Proxy services |

1. Security

The DataManager can run in secure or insecure mode. In insecure mode, any system can publish and fetch data to any resource. In secure mode, the communication is encrypted, only a system with a valid certificate can publish to the corresponding resource.

1. Internal Structure

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

* **DataManagerMain**: starts the HTTP or the HTTPS server based on the properties files and command line arguments, registers into the SR
* **DataManagerResource**: provides resources for the Proxy and Historian
* **DataManagerService**: provides functionality for the Historian service
* **ProxyService**: provides functionality for the Proxy service

# Usage

Start the module as a Java executable. The following command line arguments are available:

* “-tls”: starts the Core System in secure (HTTPS) mode, using the certificates which were set in the app.properties file.
* “-daemon” (Linux only!): starts the module in daemon mode, kill signals will prompt a normal shutdown, and the core system will de-register its services from the Service Registry.
* “-d”: starts the module in debug mode, which means every incoming REST request (URL + payload) and the corresponding response will be printed to the console output.

1. References
2. Revision history

# Amendments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Date | Version | Subject of Amendments | Author |
| 1 | 2018-09-21 | G4.0 | First draft | Jens Eliasson |

# Quality Assurance

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