Generic System-of-Systems Description (GSoSD)

The Arrowhead Core Microsystems

**Abstract**

This document outlines the 5th generation of the Arrowhead Core microsystems, which offer functionality we expect to be needed for the vast majority of Arrowhead use cases. It provides a high-level architectural description of what problems the Core microsystems solve, how these microsystems can interact, as well as how they may be applied to practical use cases.

The primary purpose of the document is to present the functionality offered by the Arrowhead Core microsystems. Its architectural aspects are defined in the abstract, by which we mean that no specific implementations or technologies are endorsed or mandated, respectively.

Table of Contents

[1. System-of-Systems Overview 3](#_Toc126329573)

[2. Microsystems 4](#_Toc126329574)

[2.1. The Core Microsystems 4](#_Toc126329575)

[2.1.1. Microservice Registry 4](#_Toc126329576)

[2.1.2. Authorisation Microsystem 5](#_Toc126329577)

[2.1.3. Orchestration Microsystem 5](#_Toc126329578)

[2.2. Local Cloud Bootstrapping 6](#_Toc126329579)

[2.3. The Microsystem-Microservice Matrix 7](#_Toc126329580)

[3. Use-cases 8](#_Toc126329581)

[3.1. Microservice Discovery 8](#_Toc126329582)

[3.2. Authorisation 9](#_Toc126329583)

[3.3. Orchestration 10](#_Toc126329584)

[3.4. Authorisation and Orchestration 11](#_Toc126329585)

[4. Control and Data Planes 12](#_Toc126329586)

[5. Release Notes 13](#_Toc126329587)

[6. Non-Functional Requirements 14](#_Toc126329588)

[6.1. Caching 14](#_Toc126329589)

[6.2. Independence 14](#_Toc126329590)

[7. References 15](#_Toc126329591)

[8. Revision History 15](#_Toc126329592)

[8.1. Amendments 15](#_Toc126329593)

[8.2. Quality Assurance 15](#_Toc126329594)

1. System-of-Systems Overview

The Eclipse Arrowhead project exists to produce solutions and specifications that support industrial automation systems that can adapt quickly and automatically to changing circumstances and business requirements. This adaptability is enabled through *Service-Oriented Architecture* (SOA), which entails dividing automation systems into smaller independent *microsystems* that communicate by sending messages to, or *consuming*, the *microservices* offered by the other microsystems. Each microsystem is expected to be designed around the principles of

1. **loose coupling**, which means that microsystems have narrow, well-defined application areas and depend on as few microservices as possible,
2. **late binding**, which entails providing as much functionality as possible given what needed microservices, configuration data and other details are currently available, as well as
3. **lookup**, which means that microsystems, when possible, look up needed data by consuming microservices instead of relying on the data being in given configuration files.

While perhaps not immediately apparent, these principles help facilitate the design of automation systems that are cheap and easy to implement, upgrade and extend, are highly resilient to failures and attacks, and are highly transparent about their constituents and activities.

Designing an industrial automation system as a system-of-systems in this manner has advantages, as we have already covered, but it also introduces the complexity inherent to all distributed applications. The Core microsystems of Eclipse Arrowhead exist to help manage this complexity by facilitating

1. **discovery**, or helping microsystems independently find the microservices they need,
2. **authorisation**, or regulating how microsystems are allowed to consume what microservices, as well as
3. **orchestration,** or centrally managing what microsystems consume what microservices.

Diagram

Description automatically generatedEach of these concerns is addressed by its own microservice, provided by its own microsystem, as outlined in the following component diagram:

In the next section, Section 2, we consider what these microservices and microsystems do in more detail. In Section 3, we describe various use cases in which some or all of them are used. In Section 4, we discuss how the Core microsystems make up the *control plane* of a distributed application and why that matters. In Section 5, we outline important differences between versions 4 and 5 of Arrowhead. In Section 6, we present key non-functional requirements. In Section 7, we consider key security requirements. Finally, in Sections 8 and 9 we list references and versions of this document.

1. Microsystems

The Core microsystems of the Eclipse Arrowhead project are meant to address concerns expected to be typical to most settings where SOA is applied. We describe them here in the abstract. The implementations of these systems are meant to be described in separate, non-generic documents we refer to as *System-of-Systems Descriptions* (SoSDs).

|  |  |
| --- | --- |
| Microsystem | Microservice |
| Microservice Registry | Microservice Discovery |
|  | Microservice Registry Management |
| Authorisation Microsystem | Authorisation |
|  | Authorisation Management |
| Orchestration Microsystem | Orchestration Management |
|  | Orchestration Pull |
|  | Orchestration Push |

The Core microsystems, and the microservices they provide, are outlined in the below table:

In addition to the Core microsystems, several *support* microsystems are described in the document *GSoSD Arrowhead Support Microsystems 5.0*. The purposes of the respective Core microsystems are presented in the following subsection, after which we consider how sets of microsystems are bootstrapped. After that we present a matrix of both core and support microsystems provided, or planned to be provided, by the Eclipse Arrowhead project.

# The Core Microsystems

## Microservice Registry

One of the main principles of SOA is *lookup*, which means that whenever possible, data required for whatever reason should be looked up by consuming a microservice. One kind of data that is virtually always required to facilitate a system-of-systems is the network information, such as IP addresses and port numbers, needed by microsystems to be able to communicate with each other.

The Microservice Registry holds a table that maps the identifiers of individual microservices to the information generally required to communicate with their providers. Individual microsystems may elect to register, update or deregister their own microservices in a Microservice Registry, and so make it possible for other microsystems to look them up. When using a Microservice Registry, microsystems are only required to know, or be able to otherwise determine, the network information of the Microservice Registry. Only the names or identifiers of consumed microservices need to be known in advance.

Registering, updating, deregistering and querying the Microservice Registry is performed through its *Microservice Discovery* microservice. In addition, the microsystem provides the *Microservice Registry Management* microservice, which allows for microservices to be registered, updated or deregistered in bulk. The latter microservice is meant to be useful primarily for administration.

## Authorisation Microsystem

When a system-of-systems applies any kind of access control, it becomes relevant for every microservice provider to know what microsystems are allowed to consume its microservices—and with what limitations. The Authorisation Microsystem holds a table that maps the identifier of each microsystem to the identifiers of the individual microservices that microsystem is permitted to consume, as well as additional details regarding any limits or tokens related to that consumption. Each entry in this table can be referred to as an *access control rule*. Whenever a microsystem attempts to consume a microservice, the provider of that microservice may consult the Authorisation Microsystem to make sure that the consumer is permitted to do.

Using an Authorisation Microsystem means that individual microservice providers do not need to know in advance what other microsystems are allowed to consume them.

Checking whether an attempted microservice consumption is allowed is performed through the *Authorisation* microservice of the Authorisation Microsystem. In addition, the microsystem also provides the *Authorisation Management* microservice, which allows for access control rules to be added, updated, removed and queried. The latter microservice is meant to be useful for administrative purposes.

## Orchestration Microsystem

Using only the Microservice Registry means that network information does not have to be manually supplied to the individual microsystems of a given system-of-systems. However, each microsystem must still be manually configured, or use its own search procedure, to determine what exact microservice instances to consume. This becomes an issue when the same type of microservice is provided by many microsystems at the same time, but there are other differences to each provided microservice. If, for example, a given microservice instance reports a temperature for a specific location, what microservice instance is consumed affects what data is received by the consumer.

The Orchestration Microsystem holds a table that maps each relevant microsystem to the specific microservice instances it *should* consume. Additionally, in contexts where tokens are used to enable access control, the Orchestration Microsystem also stores these tokens together with the identifiers of the microservices to be consumed.

Microservice consumers get lists of microservices to consume, or *orchestration rules*, by using the *Orchestration Pull* microservice of the Orchestration Microsystem. Consumers may also use the *Orchestration Push* microservice to be notified of any changes to their orchestration rules. In addition, the Orchestration Microsystem also provides the *Orchestration Management* microservice, which allows for orchestration rules to be added, updated, removed and queried. The latter microservice is meant to be useful for administrative purposes.

# Local Cloud Bootstrapping

When a set of microsystems, or a *local cloud*, starts up, none of its constituent microsystems will have the addresses or credentials necessary to consume the microservices it needs. For that data to become available for the microsystems to look up, the Core microsystems must first become ready to provide their microservices. We refer to the process of the Core microsystems becoming ready as the *bootstrapping process*. What is required for the bootstrapping of a particular local cloud to be successful depends on which of the Core microsystems are being used. However, one requirement is constant: every microsystem and every microservice must have a unique identifier.

If only the Microservice Registry is used, each microsystem must be preconfigured with the network address of the Microservice Registry—or be able to determine it through some other well-defined mechanism. Every microsystem must know how to decide what specific microservice instances to consume, something we consider more in Section 3.1.

If also the Authorisation Microsystem is used, all other microsystems must also be preregistered in its table, which maps microsystem identifiers to the identifiers of the microservices each respective microsystem is permitted to consume.

If also the Orchestration Microsystem is used, the need for having each microsystem be able to independently determine what microservice instances to consume goes away. However, what microsystems are to consume what microservices must instead be registered in the table of the Orchestration Microsystem.

# The Microsystem-Microservice Matrix

En bild som visar bord

Automatiskt genererad beskrivningIn order to keep track of what services are produced and consumed by different microsystems, we present a microsystem-microservice matrix below. This matrix outlines the Core microsystems in terms of what microservices they provide and consume.

The matrix is meant to eventually contain references to both the abstract definition of all microsystems, as well as to the available implementations of those microsystems. The abstract definitions are provided in documents we refer to as Service Descriptions (SDs), while the interfacing details of their implementations are documented in Interface Design Descriptions (IDDs). The full matrix can be found in Appendix 1.

1. Use-cases

To make it more apparent how the Arrowhead Core microsystems can be used in practical scenarios, we here present four use case descriptions. Each description centres around a sequence diagram illustrating how microsystems consume each other’s microservices.

# Microservice Discovery

Diagram

Description automatically generatedThe first, and most basic, of our use cases shows how some microsystem **B** can consume the microservice **a**, provided by **A**, by looking up its address in a Microservice Registry instance. No security mechanisms are in place. The only precondition is that **A** and **B** both know of the network address of the Microservice Registry instance. The use case is depicted below.

The use case consists of three message exchanges, which are as follows:

1. **A** registers its microservice **a** with the Microservice Registry, which accepts the registration.
2. Later, **B** attempts to lookup the microservice **a** via the Microservice registry, which responds with the network address, and other details, of **a**.
3. **B** then proceeds to consume **a** by sending a request to **A**, which responds with some status and payload of relevance to the microservice.

As we assumed that only one instance of the **a** microservice was registered in the Microservice Registry, the response in exchange 2 only contained one address. If, however, more than once such microservice instance would have been registered, **B** would have received the network addresses of all of them. **B** would then have been forced to determine by itself which instance would be most appropriate to consume, perhaps by looking at other details also present in the received registrations.

Alternatively, **B** could have provided the instance identifier of a particular microservice, instead of a microservice type name, in its query request to the Microservice Registry. Since those identifiers must be unique, the query result could then only contain at most one result. It would, however, also mean that relevant microservice instance identifiers must be known in advance, and that the specific instance is online when **B** needs to consume it.

Another approach is to centralise decisions about microservice consumption using an Orchestration Microsystem, as shown later in Section 3.3.

# Authorisation

Diagram

Description automatically generatedIn this, our second, use case, we add access control by introducing an Authorisation Microsystem. Whenever an attempt is made to consume a microservice, its providing microsystem validates the request by consulting the Authorisation Microsystem before responding to it. We assume that all microservices were registered in the Microservice Registry before the use case begins. We also assume that the Authorisation Microsystem has already been provided with an appropriate set of access control rules. The use case is depicted below:

The use case consists of the following message exchanges:

1. **A** looks up **b** via the Microservice Registry, which responds with the network address of **b**.
   1. However, to make sure A is authorised to discover microservices, the Microservice Registry consults the Authorisation Microsystem before responding, which gives its approval.
2. **A** attempts to consume **b** by sending a request to **B**, which later responds.
   1. Before responding, however, **B** must ascertain that **A** is permitted to consume its **b**. Since **B** does not yet know the network address of the Authorisation Microsystem, it queries it and subsequently receives the address.
      1. In turn, the Microservice Registry checks if B is permitted to consume its Microservice Discovery microservice, which it proves to be.
   2. **B** then proceeds to check with the Authorisation Microsystem if **A** is permitted to consume its instance of **b**, which reports that it is.

Using authorisation in this manner means that a list of access control rules must be prepared and provided to an Authorisation Microsystem in advance. As such rules must contain the unique identifiers associated with the various microsystems and microservices in the given local cloud, also those identifiers must be known in advance. To be able to guarantee that every microsystem is associated with the expected set of identifiers, it must be preconfigured with its own set of identifiers. It is possible to avoid this need for preconfiguration by using a separate microsystem that dynamically creates access control rules according to given strategies. Such a microsystem is, however, beyond the scope of this document.

# Orchestration

As we mentioned towards the end of the first use case in Section 3.1, using only a Microservice Registry means that we must either rely on (1) microsystems being able to choose what specific microservice instances to consume, or (2) being able to preconfigure each microsystem with the identifiers of those specific microservice instances. If we, on the other hand, introduce an Orchestration microsystem, we can avoid either by having the individual microsystems lookup what microservices we *want* them to consume.

Chart, box and whisker chart

Description automatically generatedWe assume that all microservices were registered in the Microservice Registry before the use case begins, as well as that the Orchestration Microsystem has been provided with an appropriate set of orchestration rules, which describe what microsystems should consume what microservices. We also assume no security mechanisms are in place. The use case is depicted below:

The use case consists of the following message exchanges:

1. **A** looks up the network address of the Orchestration Pull microservice by consuming the Microservice Discovery microservice of the Microservice Registry, which responds with the requested address.
2. **A** queries the Orchestration Microsystem for the network addresses of the microservices it should consume, which responds with the network address of **b**, provided by **B**.
   1. Since the Orchestration Microsystem does not hold any actual network addresses, it consults the Microservice Registry for the network address of **b**, which it also gets.
3. **A** consumes **b** of **B** by sending some request, which **B** responds to.

In this particular use case, orchestration information was *pulled* by **A**, by which we mean that A requested the information from the Orchestration Microsystem on its own accord. The Orchestration Microsystem also supports *pushing* orchestration rules, however, which means that rather than requesting it directly, **A** could have subscribed to changes to the orchestration rules relevant to itself. Pushing orchestration rules in this manner means that changes to the rules are more likely to end up at their respective microsystems. They are also likely going to get the information quicker than if they had been requesting their rules from the Orchestration Microsystem at regular intervals.

# Authorisation and Orchestration

While we already covered authorisation and orchestration separately in Sections 3.2 and 3.3, we have one important reason to consider them being used at the same time. The Orchestration microsystem can create and distribute access tokens, which can be used to increase security by having the credentials actually passed between microsystems expire faster.

Diagram

Description automatically generatedWe assume that all microservices were registered in the Microservice Registry before the use case begins. We also assume that **A** has already looked up the network address of the Orchestration Microsystem, as well as that the Orchestration Microsystem has been provided with an appropriate set of orchestration rules. We also assume that the Authorisation Microsystem has already been provided with an appropriate set of access control rules. The use case is depicted below:

The use case consists of the following message exchanges:

1. **A** requests its set of orchestration rules from the Orchestration Microsystem, which later responds with the network address of **b** and the access token **A** needs to consume **b**.
   1. First, however, the Orchestration Microsystem checks if **A** is authorised to request its orchestration rules, which it proves to be.
   2. Then the Orchestration Microsystem determines the network address of **b** by consulting the Microservice Discovery microservice of the Microservice Registry.
      1. The Microservice Registry checks if the Orchestration Microsystem is allowed to consume the Microservice Discovery microservice, which it is.
   3. Before responding A, the Orchestration Microsystem also requests the Authorisation Microsystem to create an access token for A, which it does.
2. **A** consumes **b** and includes the access token it received in the request. **B** responds.
   1. Before responding, **B** uses the access token to check if **A** is authorised to consume **b**, which it proves to be.
3. Control and Data Planes

**Work in progress.**

A section for describing the usage of the AH Core services to set up and control the information exchange and separation from the actual data transmission.

1. Release Notes

**Work in progress.**

This section describes the differences between current version and previous versions of the architecture.

1. Non-Functional Requirements

**Work in progress.**

# Caching

In order to increase autonomy and robustness caching of AH Core service content is advised.

# Independence

Individual microsystems must proactively and reactively ensure that the best possible effort is exerted for it to be able to fulfil its intended function.

1. References

**Work in progress.**

1. Revision History

# Amendments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Date | Version | Subject of Amendments | Author |
| 1 | 2023-01-12 | 0.1 | Initial draft. | Per Olofsson |
| 2 | 2023-02-03 | 0.2 | Refined and extended Sections 1, 2 and 3. Removed unneeded sections. Cleaned up the other. | Emanuel Palm |

# Quality Assurance

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Date | Version | Approved by |
| 1 | YYYY-MM-DD | 1.0 | Nnnnn Nnnnnnn |
| 2 |  |  |  |