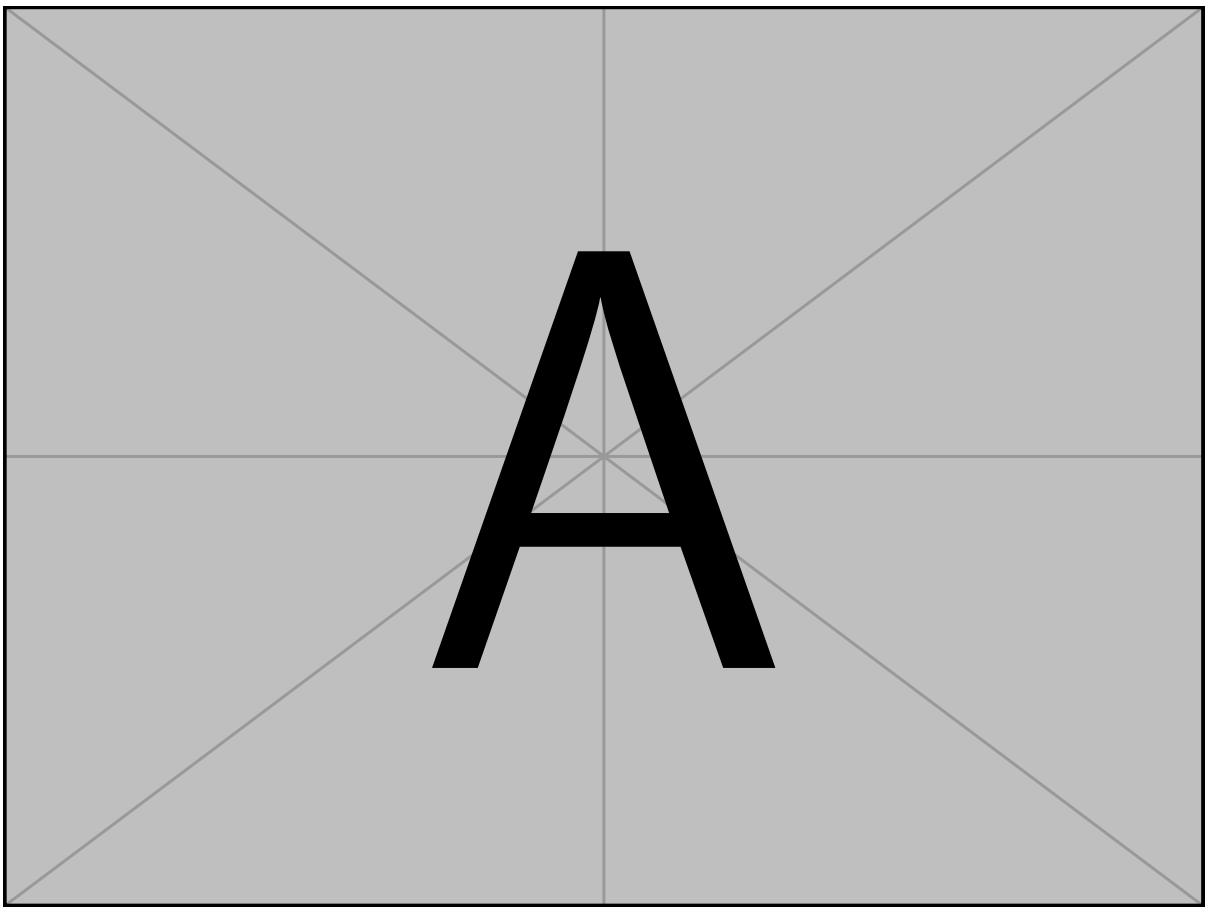
Embedded IoT for Eclipse Arrowhead



Albin Martinsson

Dept. of Computer Science and Electrical Engineering
Luleå University of Technology
Luleå, Sweden

Supervisor:

Jan van Deventer

To my dad Bengt-Göran Martinsson a special thanks for proof reading is required...

Abstract

The abstract is a mini thesis on its own. It should contain the briefest of motivation and problem description, what has been done, and summarize the results. The purpose is to give the reader a quick view of the content, and encourage the reader to read the rest of the thesis. It's also helpful to put the reader in the right frame of mind for the rest of the thesis. This section is typically anything between one paragraph for short research papers (of 8 pages) to a page for a full thesis

Contents

Acknowledgments

The research conducted in this thesis has been funded by the Arrowhead Tools project.

Chapter 1

Introduction

This thesis will take a look at the opportunity and possibility to connect your embedded IoT devices to a local Eclipse Arrowhead framework cloud. This project will use the STM32 B-L4S5I-IOT01A IoT discovery node as a development board, running the Mbed-OS 6.

1.1 Background

According to Artemis-IA the world is on the verge of a new industrial revolution, moving towards a more decentralized and software-oriented means of production.[1] This forth new, and in some sense planned, industrial revolution is called Industry 4.0 Lasi means.[2] Incorporating System of Systems, Cyber-Physical systems, and embedded software technologies will form the backbone and inherent part of every value chain then this new industrial revolution is complete Artemis-IA means.[1]

Cyber-Physical systems acts as a brigde between the data rich cybernetic world and the tecnology rich physical world Artemis-IA means. Artemis-IA adds that a differentating factor between traditonal embedded systems and Cyber-Physical systems are their scale, where traditonal embedded system have a more limited scale. Cyber-Physical system, on the other hand, has a much larger scale including interconnected embedded systems, human-, and socio technlogical systems as well Artemis-IA adds.[1]

For Europe to be able to compete with the rest of the worlds larger economies with iniatives, as for instance Chinas Made in China 2025, Europe needs to invest in the aforementioned technologies Artemis-IA adds. A shift away from propriatary solutions towards collaborative solutions is also needed means Artemis-IA in their report Embedded intelligence.[1]

1.2 Motivation

The need for a European incentive promoting Industry 4.0 is clear. The Arrowhead Tools project aims for digitalization and automation solutions for the European industries according to their website.[3] The Arrowhead Tools project uses the open source Eclipse Arrowhead framework, further contributing to the collaborative solutions needed for the European economie defined in the previous section.

The Eclipse Arrowhead framework contains a lot of examples in an array of different languages and technologies mainly java, the language the framework is developed in. Python, C#, and C++ have client libraries and example code developed for them, which can be found on the project's GitHub page.[4]

However, there is no client library or code example for a specific piece of hardware to make the connection to a local Arrowhead cloud fast and easy, showcasing the capabilities of this project. A project with the clear intent to showcase both the capabilities, and possibilities of Cyber-Physical systems and the Eclipse Arrowhead framework as well is therefore needed.

This thesis will examine the possibilities of having a ready-made example to compile and run on a specific hardware plattform, that enables connection to a local Arrowhead cloud as a proof of concept. Much like the 'Getting started with' examples from Amazon Web Services and Microsoft Azure.[5, 6]

1.3 Problem definition

This project aims to investigate the possibilities, benefits, and limitations of using the Eclipse Arrowhead framework on embedded devices in contrast to commercially available solutions such as Amazon's Amazon Web Services and Microsoft Azure.

1.4 Equality and ethics

The ability to own and control your data is becoming rarer and rarer these days with giant corporations establishing their cloud services. You as a consumer always take a risk when pushing sensitive data to a cloud owned by someone else, the right to own your data should not have to be infringed upon. The Eclipse Arrowhead framework and the use of local clouds move the storage of your data from giant corporations to your own.

1.5 Sustainability

The use of small embedded devices instead of monolithic machines used by the industry today provides a much-needed decrease in energy consumption for larger industries.

The use of IoT devices would also on a greater scale enable preventive maintenance of components, reducing both the cost and materials required for maintenance later on.

1.6 Delimitations

1.6.1 Security

This thesis does not cover a solution to the numerous security risks and issues associated with IoT devices.

1.6.2 Core systems

This thesis will also only cover the three core systems of the Eclipse Arrowhead framework, which are the service registry, authorization, and orchestrator. The STM32 B-L4S5I-IOT01A IoT discovery node will not host the Arrowhead framework on the board itself, since the Arrowhead framework is too resource-heavy for such a small device. The board will instead connect to a local Arrowhead cloud hosted by another computer in the same network.

1.6.3 Intercloud connection

It will also only cover connection within the same local Arrowhead cloud, intracloud, as opposed to using multiple clouds, intercloud. Intercloud connection requires two more Arrowhead systems, gateway, and gatekeeper, to operate and the configuration of those systems is beyond the scope of this thesis.

1.7 Thesis structure

In chapter 2 related work is presented, a literature review of IoT, Industry 4.0, security, and the Eclipse Arrowhead framework is conducted. In chapter 3 theory is covered, describing what tecnolgies and scientific methods were used in this thesis. Chapter 4 covers implementation, describing how the different systems used in this thesis uses are designed from a software engineering perspective. An overview of how the systems interact with each other and how they are implementated. In chapter 5 an evaluation of the experiment conducted will be performed. Chapter 7 presents the conclusion of the work done in this thesis. The chapter also describes how to further investigate the questions raised in this thesis. In chapter 8 there is a list of references used in this thesis.

Chapter 2

Related work

2.1 Internet of things

Cluster of European research projects regarding the Internet of Things:

> 'Things' are active participants in buissness, information and social processes where they are enabled to interact and communicate aming themselves and with the enviorment by exhchanging data and information sensed about the envoirment while reacting autonomously to real/physical world events and influencing it by runnign processes that trigger actions and create services with or without direct human intervention.[7]

Gubbi et. al. defines the Internet of Things as:

> Interconnection of sensing and actuating devices providing the ability to share inforamtion across platforms through a unified framework, developing a common operating picture for enabling innovative applications. This is achieved by seamless large scale sensing, data analytics and information representation using cutting egde ubiquitous sensing and cloud computing.[7]

2.2 Industry 4.0

Lasi argues that the term industry 4.0 was coined beforehand as a planned fourth industrial revolution.[2] The use of internet of things devices, IoT devices from now on, and cyber-physical systems, CPS from now on is what defines the fourth industrial revolution Vadiya means.[8] See figure x for a short historic overview of previous industrial revolutions.

According to Vadiya industry 4.0 promotes the connection of sensors and devices both to the internet and to other sensors or devices.[8]

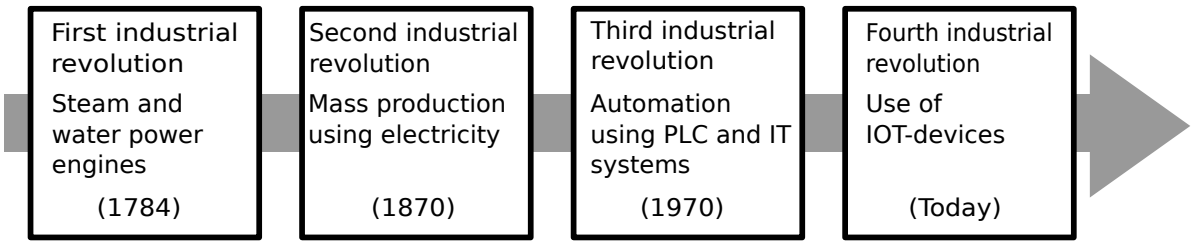


Figure 2.1: Historic overview of previous industrial revolutions

Hozdić states that a sensor is a device capable of providing an appropriate output in response to a measured value. One key feature of an intelligent sensor is that to increase the level of information processing it processes the information at a logical level Hozdić argues. An Intelligent sensor is capable of executing actions based on the measured value in contrast to regular sensors, making them easier to set up and use means Hozdić.[9]

Hozdić defines a cyber-physical system, CPS, as a new generation of a system that integrates both physical and computer abilities. A cyber-physical system consists of two parts, one cybernetic and one physical. The cybernetic part of the system can be viewed as a summation of logic and sensor units while the physical part of the system can be viewed as a summation of the actuator units Hozdić adds. Xu et. al. states that cyber-physical systems are a key part of Industry 4.0. In contrast to the simple embedded systems of today will be exceeded due to advances in CPS that enable enhanced capability, scalability, adaptability, resiliency, safety, usability, and security.[10] Hozdić argues that the CPS can share and receive information from intelligent sensors that connect to digital networks is what enables and form an internet of things.[9]

2.3 Arrowhead framework

A local cloud is defined as a self-contained network with at least the three mandatory systems deployed, more on those in a later paragraph. Delsing et.al. also argues that the three mandatory core systems running a local cloud also need at least one application system deployed.[11]

Two terms have to be introduced to further understand what the Eclipse Arrowhead framework aims to accomplish, services and systems. Delsing et. al. defines a system as what is providing or consuming a service. Furthermore, a service is defined as what is used to convey information between a provider and a consumer Delsing et. al. argues.[11]

The Eclipse Arrowhead framework, Arrowhead from now on, consists of three mandatory core systems according to Delsing et. al. To fully operate a local cloud as defined in the previous section it must, according to Delsing, contain:

- Service registry system.

- Authorization system.

- Orchestration system.[11]

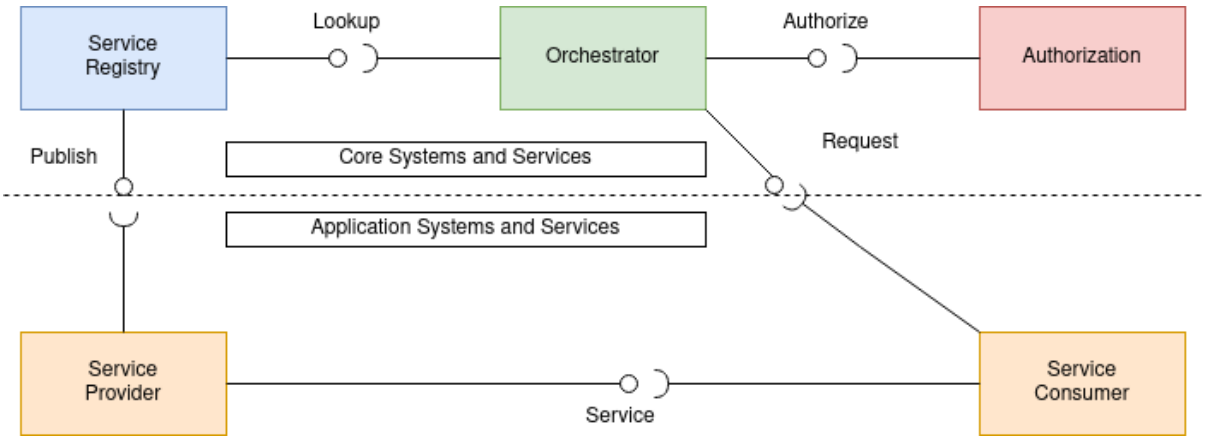


Figure 2.2: The core systems of the Eclipse Arrowhead framework

More about the dynamics betweeen the consumer and provider system will follow in the theory section.

2.4 Amazon Web Services

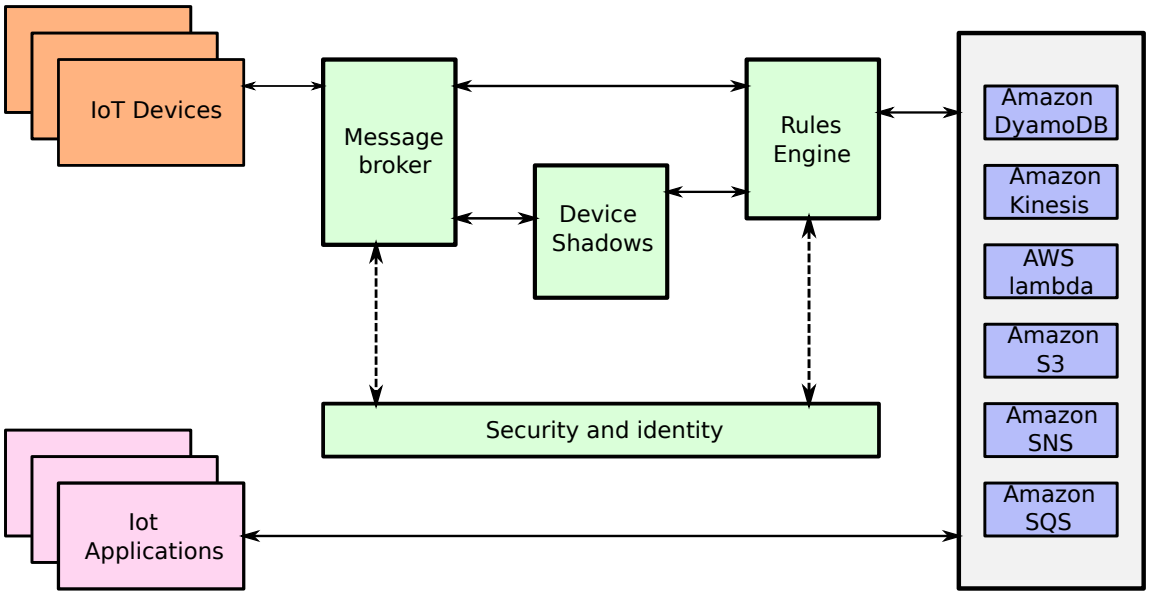


Figure 2.3: The core systems of the Amazon Web Services

2.5 Security

Meneghello et. al. argues that the increasing number of IoT devices and the pervasive nature of new smart home or healthcare devices can pose a real threat to the users' integrity. Sensitive information Meneghello et. al. defines as a video recording of the user's home, the user's location, access to buildings, health monitoring, and industrial processes.[12]

Meneghello et. al. states the security requirements of an IoT system can be divided into three different operational levels, namely the information, access, and functional level. The information level should guarantee that the integrity, anonymity, confidentiality, and privacy of the system are preserved. Meaning that messages should not be altered during transmission, the identity of the data source and the clients' private information remains hidden and that data cannot be read by third parties Meneghello et. al. argues. The access level provides a guarantee that only legitimate users are allowed access to the network and the devices associated with that network. It also guarantees that users within the network only use resources they are allowed to use Meneghello et. al. states. The functional level should guarantee the continued functionality of a network even in

the case of malfunction or a malicious attack Meneghello et. al. adds.[12]

Zhang also argues that privacy is a big concern with IoT devices and suggests two solutions data collection policy and data anonymization. A policy that describes how the data is collected from the devices would restrict the flow of data, therefore ensuring privacy preservation can be ensured Zhang states. Data anonymization means that the private information sent by the IoT devices is either encrypted or that the relation of the data and its owner is concealed according to Zhang.[13]

Meneghello et. al. argues that one of the main aspects of security within IoT is to ensure that the data sent is the data received and that the data has not been tampered with or read during transmission. The most important operation to guarantee that is the use of encryption, which converts the message sent in plain text to an encrypted message only readable with a decryption key Meneghello et. al. states. Meneghello et. al. states that there are two mechanisms for encryption, symmetric and asymetric. Symmetric encryption is when the same key is used for encryption and decryption, so it has to be shared with both the sender and receiver. Asymmetric encryption on the other hand only shares the public key and the sender and receiver have their own private keys Meneghello et. al. means.[12]

Hassija et. al. states the importance of end-to-end encryption and the challenges it poses for IoT systems. End-to-end encryption is required to ensure the confidentiality of the data, the application should not let anyone except the intended recipient read the messages sent Hassija adds.[14]

Noor, Meneghello, and Zhang state the importance of authentification in IoT systems.[15, 12, 13] Noor adds that 60% of all IoT systems use authentification to grant access to the user.[15] Zhang argues that public key cryptosystem provides more security compared to symmetric encryption schemes, but has the drawback of having high computational overhead.[13]

Noor argues that conventional cryptographic primitive is unsuitable for IoT devices due to their lack of computational power and limited battery life and memory capacity.[15] With IoT devices lacking capabilities as background Noor, Meneghello, and Zhang all agree that a push for lightweight cryptography is required to ensure the security of these devices.[15, 12, 13]

2.6 Communication

MQTT, Message Queue Telemetry Transport, is a lightweight messaging invented by IBM suitable for IoT Wukkadada means.[16] MQTT is a publish/subscribe protocol that requires a minimal footprint and bandwidth to connect an IoT device according to HIVEMQ.[17] MQTT consists of an MQTT broker and MQTT clients, where the broker is responsible for sending messages between the sender and its recipients.[16] A client on the other hand publishes a message to the broker that other clients can subscribe to

HIVEMQ adds.[17]

HTTP, HyperText Transfer Protocol, is a request/response protocol consisting of clients and servers that communicate by exchanging individual messages. The clients are responsible for the requests and the servers are responsible for the response Mozilla developer network clarifies. [18] In contrast to the lightweight MQTT protocol with low overhead and bandwidth, HTTP can cause serious bandwidth issues Wukkadada adds. [16] The biggest benefits to using HTTP are that it supports the RESTful Web architecture and that it is a globally accepted web messaging standard Naik suggests. [19]

Naik argues that HTTP exceeds MQTT in message size, message overload, power consumption, resource requirements, bandwidth, and latency. All things that are considered negative for a protocol. On the other hand, HTTP exceeds MQTT in interoperability, standardization, security, and provisioning Naik adds.[19] All things that are considered positive for a protocol. Shariatzadeh argues that that HTTP may be expansive for many IoT devices but it can be beneficial due to the interoperability since it is developed for the web originally.[20] Wukkadada also points out the lower power consumption of the MQTT protocol but adds that the more verbose HTTP protocol can be easier for developers to understand. Wukkada drives home the point of choosing MQTT for IoT devices.[16]

Chapter 3

Theory

3.1 MYSQL

3.2 Swagger UI

3.3 Arrowhead

A local cloud is defined as a self-contained network with at least the three mandatory systems deployed, more on those in a later paragraph. Delsing et.al. also argues that the three mandatory core systems running a local cloud also need at least one application system deployed.[11]

Two terms have to be introduced to further understand what the Eclipse Arrowhead framework aims to accomplish, services and systems. Delsing et. al. defines a system as what is providing or consuming a service. Furthermore, a service is defined as what is used to convey information between a provider and a consumer Delsing et. al. argues.[11]

The Eclipse Arrowhead framework, consists of three mandatory core systems according to Delsing et. al To fully operate a local cloud as defined in the previous section it must, according to Delsing, contain:

- Service registry system.
- Authorization system.
- Orchestration system.[11]

The service registry system is responsibly for enabling discovery and registring services Delsing et. al. states. According to the Eclipse Arrowhead projects own GitHub page the service registry system provides the database which stores the offered services in the local cloud.[21] The Github page also states the three main objectives of the service registry system are:

- To allow the application system to register available services to the database.
- Remove or update available services from the database.
- Allow application system to use the lookup functionality of the registry.

The Authorization system contains two databases for keeping track of which system can consume services from which other systems, depending on whether or not the Application system are in the same cloud or not according to the projects Github page. The GitHub documentation also states that if the authorization happens within the same cloud it is called intra-cloud authorization and if it happens across two local clouds it is called inter-cloud authorization.[21]

The Orchestration system is responsible for pairing and finding service providers and consumers Delsing et. al. declares. Delsing et. al. continues to state that the orchestrator also stores the orchestration requirements and the resulting orchestration rules.[11] The project's documentation argues that the main objective of the orchestrator system is to find an appropriate provider for the requesting consumer system.[21]

The documentation also states that there are two types of orchestration, store orchestration, and dynamic orchestration. Store orchestration uses the database orchestration store to find predefined orchestration rules. Dynamic orchestration on the other hand searches the entire local cloud, or even other clouds, to find the matching provider.[21]

3.4 ARM Mbed

3.5 MBed-http

Chapter 4

Implementation

The system consists of three major parts, the provider on the stm32 board, a Python flask app as a consumer, and the Eclipse Arrowhead framework. The main objective of the Eclipse Arrowhead framework is to connect the consumer and the provider in a safe and structured way. The provider is built with C/C++ using ARMs' mbed os, and mainly the mbed-http library.

4.1 System architecture

4.1.1 Services

The provider offers two services to the consumer. The first one is sending the temperature from the LPS22HB temperature and pressure sensor. The service URI of this service is /temperature and when performing a GET request the temperature reading will be returned as an integer. A sequence diagram visualizing how the temperature service is implemented.

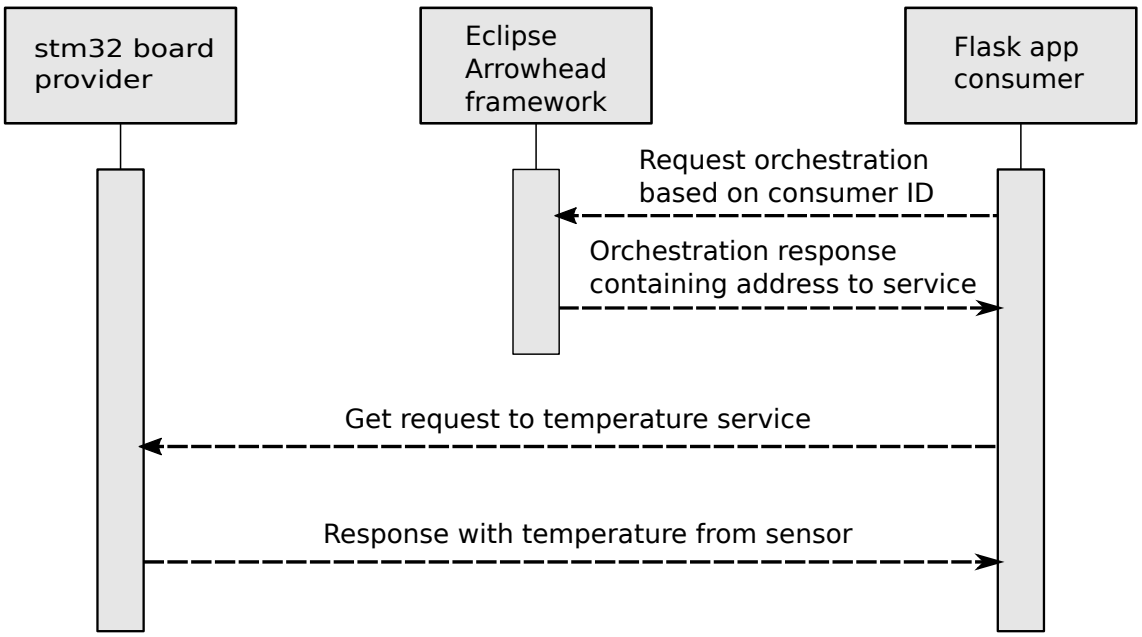


Figure 4.1: Sequence diagram of the process of connecting the consumer and provider through the temperature service.

The second one is turning on and off a LED based on the state of that LED. The service URI of this server is /LED and the desired state of the LED, ON or OFF, is specified in the payload of that call. It returns the new state of the LED and perform that action on the LED. A sequence diagram visualizing how the LED service is implemented.

4.1.2 Sequence of execution

Since the core systems is dependent on each other, the order of execution of the queries to the database matters a lot. To have the board act as a part of an Arrowhead system and registering that to the local cloud the following order of execution has to be used.

- Register provider.
- Register consumer.
- Register a service definition.
- Add intracloud authorization rules.
- Create an orchestration store entry.
- Recieve orchestration information based on consumer ID.

A sequence diagram visualizing the order of execution in the core systems.

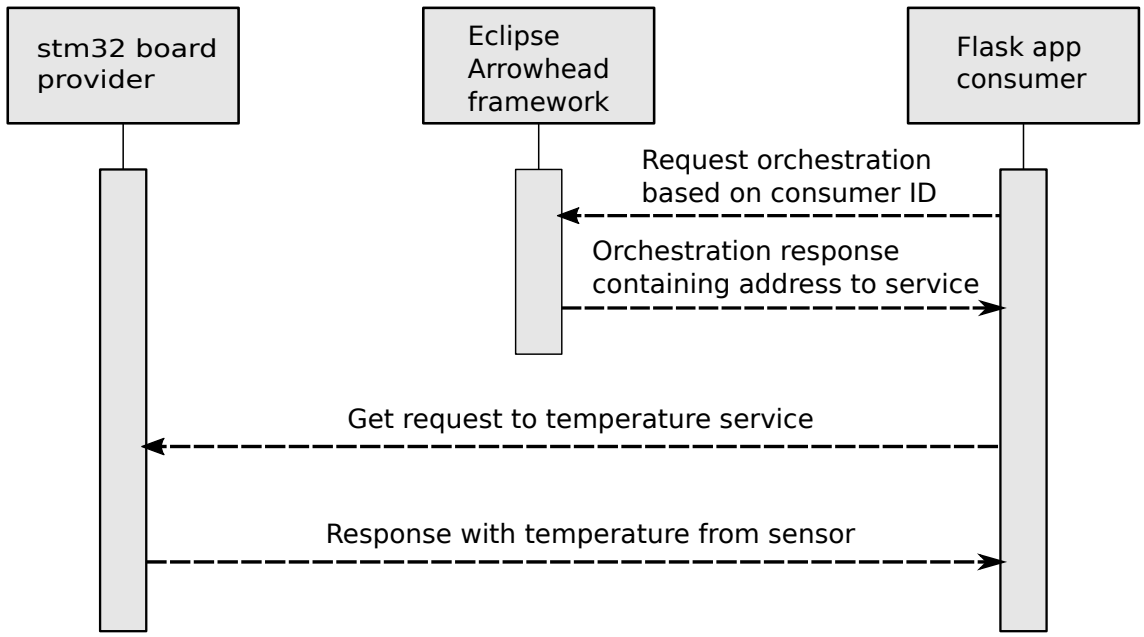


Figure 4.2: Sequence diagram of the process of connecting the consumer and provider through the LED service.

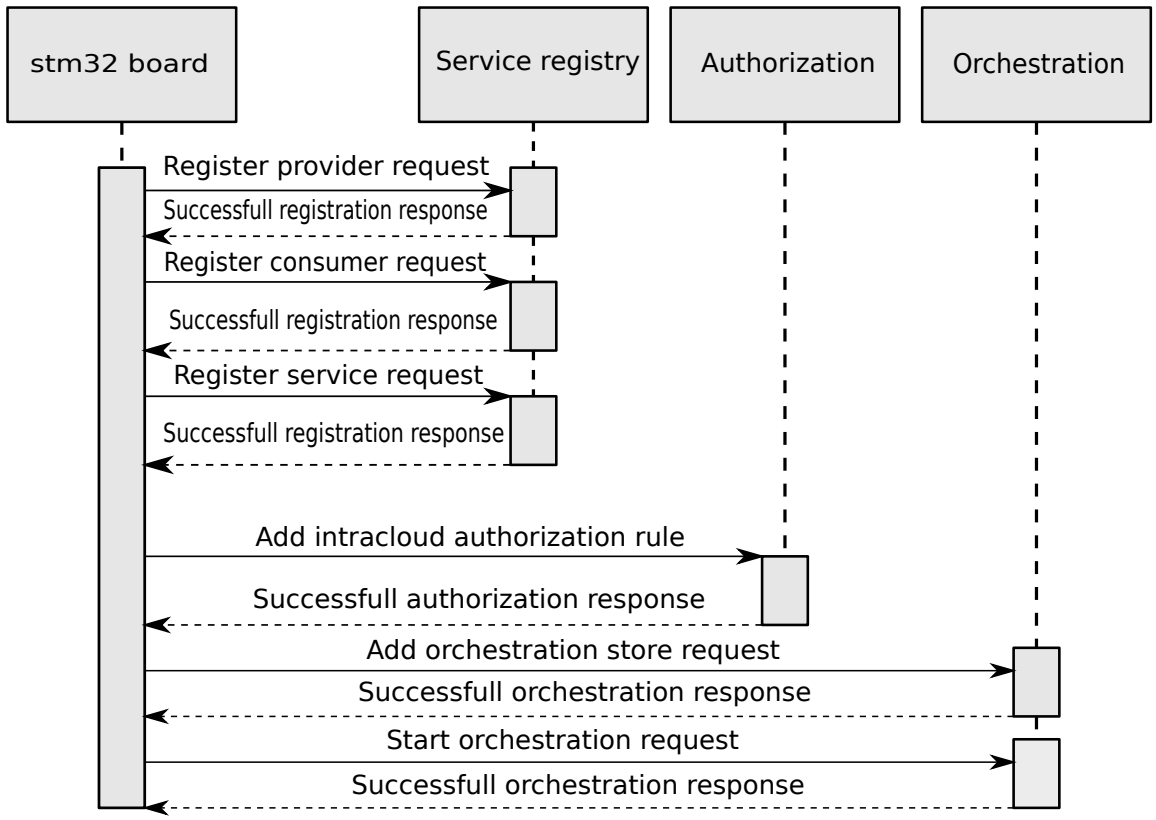


Figure 4.3: Sequence diagram of the process of using the Eclipse Arrowhead framework.

4.2 System components

Based on the architecture described in the previous section it is clear that the software components have two main tasks

- Send and recieve information using HTTP POST and GET methods.
- Constructing correct JSON strings to act as payload in the POST and GET methods.

4.2.1 HTTP post using the mbeb-http library

The first function performs a HTTP post with a constructed JSON body as payload. It does that with the help of the network interface object defined in the setup and sends that post to the appropriate URL.

```
1  std::string http_post_request_with_response(NetworkInterface* _net, std
   ::string url, std::string body)
2  {
3      HttpRequest *post_request = new HttpRequest(_net, HTTP_POST, url.
   c_str());
4      post_request->set_header("Content-Type", "application/json");
5      HttpResponse *post_response = post_request->send(body.c_str(),
   strlen(body.c_str()));
6      if (!post_response) {
7          printf("HttpRequest failed (error code %d)\n", post_request->
   get_error());
8          return std::to_string(post_request->get_error());
9      }
10     printf("\n----- HTTP POST response -----\n");
11     std::string response_body = post_response->get_body_as_string();
12     delete post_response;
13     return response_body;
14 }
```

4.2.2 HTTP get using the mbeb-http library

The second function is similar to the first one with the main difference that it performs an HTTP get with a JSON payload. It also uses the network interface to send it to the appropriate URL. It does that with the help of the network interface object defined in the setup and sends that post to the appropriate URL.

```
1 std::string http_get_request_with_response(NetworkInterface* _net, std::
   string url)
2 {
3     HttpRequest *get_request= new HttpRequest(_net, HTTP_GET, url.c_str());
4
5     HttpResponse *get_request_response = get_request->send();
6
7     if (!get_request_response) {
```

```
8           printf("HttpRequest failed (error code %d)\n", get_request->
    get_error());
9           return std::to_string(get_request->get_error());
10      }
11      printf("\n------ HTTP GET response ------\n");
12      std::string response_body = get_request_response->get_body_as_string();
13      delete get_request_response;
14      return response_body;
15  }
```

4.2.3 Constructing appropriate JSON strings

To use the POST and GET function defined in the previous section a correct JSON payload, or HTTP body, has to be created. To register a system, consumer, or provider, a body similar to the one defined underneath should be used.

```
1       std::string register_system_body = "{\"address\": \"192.168.0.101\", \"
    authenticationInfo\": \"\", \"port\": 1234, \"systemName\": \"
    system_name\"}";
```

The next operation is to register a service, and just as when registering a system, a correct JSON payload is required. The previously defined provider system is passed as a parameter here an interface, has to be defined as well.

```
1       std::string register_service_body = "{\"serviceDefinition\": \"
    service_definition\", \"providerSystem\": {\"systemName\": \"
    system_name\", \"address\": \"192.168.0.101\", \"port\": 1234, \"
    authenticationInfo\": \"\" }, \"interfaces\": [\"HTTP-INSECURE-JSON\"],
     \"serviceUri\": \"temperature\"}\r\n";
```

These three calls should all be made to the service registry core system.

To create intracloud rules provider, consumer, and service definition ids have to be passed as parameters. Two helper functions were implemented to achieve this. The first one parses the response from registering a system, finds the substring containing the systems id and returns that as a character pointer. The second one parses the response from registering a service, finds the substring containing the service definition id, and returns that as a character pointer. The field interfaceIds can be looked up in the table system_interface in the Arrowhead database and should correlate with the selected interface.

```
1  +----+---------------------+---------------------+---------------------+
2  | id | interface_name      | created_at          | updated_at          |
3  +----+---------------------+---------------------+---------------------+
4  |  1 | HTTP-SECURE-JSON    | 2021-02-03 11:56:35 | 2021-02-03 11:56:35 |
5  |  2 | HTTP-INSECURE-JSON  | 2021-02-03 11:56:35 | 2021-02-03 11:56:35 |
6  |  3 | HTTPS-SECURE-JSON   | 2021-02-16 17:45:57 | 2021-02-16 17:45:57 |
7  +----+---------------------+---------------------+---------------------+
```

The correct JSON can now be constructed and posted to the authorization core system.

```
std::string add_intracloud_authorization_body = "{\"consumerId\": " +
std::to_string(consumer_id) + ",\"interfaceIds\": [3], \"providerIds\":
 [" + std::to_string(provider_id) + "], \"serviceDefinitionIds\": [" +
std::to_string(service_id) + "]}\r\n";
```

To create an orchestration store rule the previously defined consumer and provider system and the consumer id have to be provided. Information about the operating cloud and interface has to be defined, and posted to the orchestrator core system.

```
std::string create_orchestration_store_body = "[{ \"serviceDefinitionName
\": \"service_definition\", \"consumerSystemId\": " + std::to_string(
consumer_id) + ", \"providerSystem\": { \"systemName\": \"system_name
\", \"address\": \"192.168.0.101\", \"port\": 1234, \"
authenticationInfo\": \"\"}, \"cloud\": { \"operator\": \"aitia\", \"
name\": \"testcloud2\" }, \"serviceInterfaceName\": \"HTTP-INSECURE-
JSON\", \"priority\": 1}]\r\n";
```

A get request is sent to the orchestrator with the id of the consumer as a parameter. The response from the orchestrator contains information about the address, port, and service URI of the provider the consumer wants to connect to. To parse the response from the orchestrator a helper function was created, it takes the response as a parameter and returns the address, port, and service URI as a string. If every step is successful, the consumer can connect to the provider.

4.3 Error handling

From the sequence diagram above it can be seen that the success of each subsequent operation is dependent on the success of the previous one. Without properly defined and registered systems a service definition can not be registered for instance, if one command fails the chain of commands is broken. The commands will still execute but return an appropriate error code and error message describing what the error is. For instance, a provider system with the same name as a previous one in the database is trying to register itself to the service registry. This will throw an error stating that such a system already exists and not return the system id, which will cause the command to add intracloud rules to fail since the id of the provider system is required. The orchestrator can not be performed if the systems are not authorized correctly, causing the GET method to the orchestrator returning a blank response which in turn causes the consumer not being able to connect to a provider.

Chapter 5

Evaluation

5.1 Evaluation

Describe the test setup to verify that your problems from 1.3 have been solved. This can be done in different ways depending on focus of your problems. Some problems may purely objective, such as "improve the performance of X compared to Y". These are easy to evaluate since you simply need to compare the performance, and perhaps compare against a few more technologies that you have listed in Section 2 (related work). In other cases the problems may be very subjective, such as "Create a mobile app that can be used while driving, and which shows the most fuel efficient time to change gear". This problem will require a user-study in which several persons drive without the application, you calculate the fuel consumption, then they drive with the application and then you calculate the fuel consumption again. Then you collect the objective measurements (fuel consumption comparisons) and the subjective opinions from the users about whether the application was unobtrusive, usable, etc. (typically via a questionnaire)

Chapter 6

Discussion

6.1 Discussion

Discuss how you solved your problems from 1.3, and what the results were (from section 5). Describe alternative solutions, what you could have done differently, problems you encountered, how your results compare to other peoples' results, etc. Go through each problem individually, and then in the end add general remarks and discussion points which are outside the problems themselves but that you think may be valuable to share with the reader. This section can have several subsections.

Chapter 7

Conclusions and future work

7.1 Conclusions and future work

This section describes the outcome of your work and summarizes your efforts. It also outlines things that are left to do to reach a full solution, or to integrate your solution with something else.

7.2 Security

References

[1] Artemis-IA News Embedded Intelligence: Trends & Challenges book release. https://artemis-ia.eu/news/embedded-intelligence-tsrends-challenges-book-relea html, accessed 2021-04-21.

[2] Heiner Lasi, Peter Fettke, Hans Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business and Information Systems Engineering*, 6:239–242, 8 2014.

[3] Arrowhead tools. https://arrowhead.eu/arrowheadtools/, accessed 2021-04-22.

[4] Arrowhead consortia. https://github.com/arrowhead-f, accessed 2021-03-10.

[5] User Guide and Freertos Qualification. User manual Getting started with X-CUBE-AWS STM32Cube Expansion Package for Amazon Web Services ® IoT. (September):1–31, 2020.

[6] Getting started with the STMicroelectronics B-L475E-IOT01A / B-L4S5I-IOTOA1 Discovery https://docs.microsoft.com/en-us/samples/azure-rtos/getting-started/getting-started-with-the-stmicroelectronics-b-l475e-iot01a--b-l4s5i-iotoa1-dis accessed 2021-04-22.

[7] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29:1645–1660, 9 2013.

[8] Saurabh Vaidya, Prashant Ambad, and Santosh Bhosle. Industry 4.0 - a glimpse. volume 20, pages 233–238. Elsevier B.V., 1 2018.

[9] Elvis Hozdić. Smart factory for industry 4.0: A review, 2015.

[10] Li Da Xu, Eric L. Xu, and Ling Li. Industry 4.0: State of the art and future trends. *International Journal of Production Research*, 56:2941–2962, 2018.

[11] Delsing Jerker. Iot automation: Arrowhead framework - 1st edition - jerker delsing -. 2017.

[12] Francesca Meneghello, Matteo Calore, Daniel Zucchetto, Michele Polese, and Andrea Zanella. Iot: Internet of threats? a survey of practical security vulnerabilities in real iot devices. *IEEE Internet of Things Journal*, 6:8182–8201, 10 2019.

[13] Zhi Kai Zhang, Michael Cheng Yi Cho, Chia Wei Wang, Chia Wei Hsu, Chong Kuan Chen, and Shiuhpyng Shieh. Iot security: Ongoing challenges and research opportunities. pages 230–234. Institute of Electrical and Electronics Engineers Inc., 12 2014.

[14] Vikas Hassija, Vinay Chamola, Vikas Saxena, Divyansh Jain, Pranav Goyal, and Biplab Sikdar. A survey on iot security: Application areas, security threats, and solution architectures, 2019.

[15] Mardiana binti Mohamad Noor and Wan Haslina Hassan. Current research on internet of things (iot) security: A survey. *Computer Networks*, 148:283–294, 1 2019.

[16] Bharati Wukkadada, Kirti Wankhede, Ramith Nambiar, and Amala Nair. Comparison with http and mqtt in internet of things (iot). pages 249–253. Institute of Electrical and Electronics Engineers Inc., 12 2018.

[17] Getting Started with MQTT. `https://www.hivemq.com/blog/how-to-get-started-with-mqtt`, accessed 2021-03-18.

[18] An overview of HTTP. `https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview`, accessed 2021-03-10.

[19] Nitin Naik. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. Institute of Electrical and Electronics Engineers Inc., 10 2017.

[20] N Shariatzadeh, T Lundholm, L Lindberg, G Sivard Procedia Cirp, and undefined 2016. Integration of digital factory with smart factory based on internet of things. *Elsevier*.

[21] eclipse-arrowhead/core-java-spring. `https://github.com/eclipse-arrowhead/core-java-spring`, accessed 2021-03-10.