System Architecture Specification

(TINF19C, SWE I Practise Project 2020/2021)

Project: Service Registry

Customer: Rentschler & Holder
Rotebühlplatz 41
70178 Stuttgart

Supplier: Team 4 (Daniel Baumann, Tim Diehl, Goran Erdeljan, Serdar Ilhan, Benedict Wetzel)

Version	Date	Author	Comment
0.1	07.09.2020		created

Table of Contents

1. Introduction

The goal of the project is to add service discovery functionalities to the existing Oi4-Service-Registry, developed by the OI4-Alliance. To be added features are the registration of devices, which are announced via DNS-SD but also to take the services, which are already registered at the OI4-Service-Registry and announce them to the network using the DNS-SD mechanism. These features shall be implemented in an application running in a Docker-Container. The project shall also contain a Docker-Application for testing the functionalities of the system.

2. System Overview

2.1 System Environment

The Project shall be run on any server or computer in a Industry 4.0 environment. In this environment there may be any number of devices, which offer services over a network. These services shall either announce themselves using the DNS-SD mechanism or be directly connected to the OI4-MessageBus running on the MQTT-Broker.

2.2 Software Environment

All parts of the project shall run in Docker-Containers. The software will be implemented using NodeJS. Therefore the Docker-Containers will be based on the 'node'-Docker-Image. It is also required that the OI4-Service-Registry is running on the same Network. To communicate with the registry the network shall also contain a MQTT-Broker like 'Mosquitto'.

2.3 Hardware Environment

The Project shall be run on any server or computer running docker. Preferably the system shall run a distribution of Linux.

2.4 Quality Goals

2.4.1 Maintainability

As the system is split into multiple Docker-Container each one of them can be easily replaced. The application shall be highly configurable and be documented in an extensive user guide, making it easy to adapt and maintain.

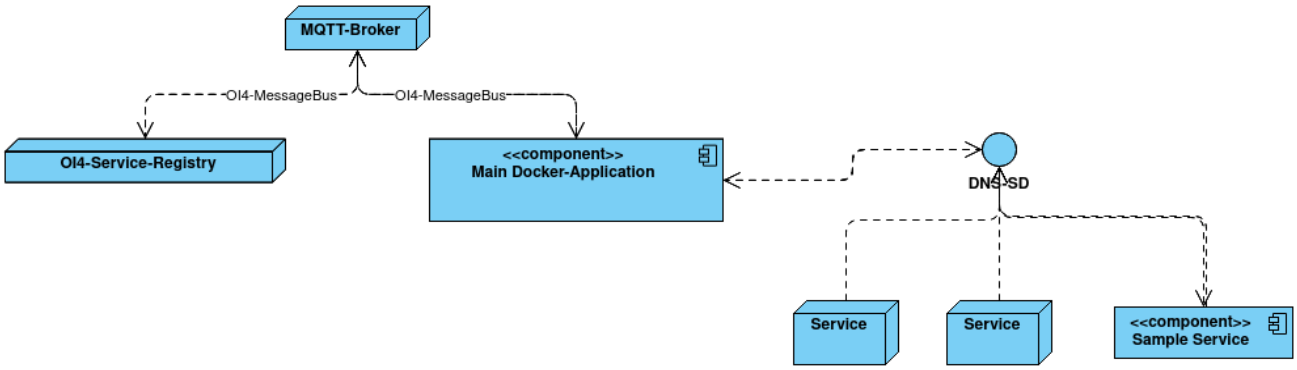
2.4.2 Efficiency

The system will work in real-time. Therefore it doesn't require persisting data between sessions. This opens up the possibility of a very lightweight approach and allows the system to work without any dedicated storage, except for the program itself.

2.4.3 Portability

Utilising the Containering provided by Docker shall make the system easily portable to any computer capable of running Docker-Containers.

3. Architectural Concept

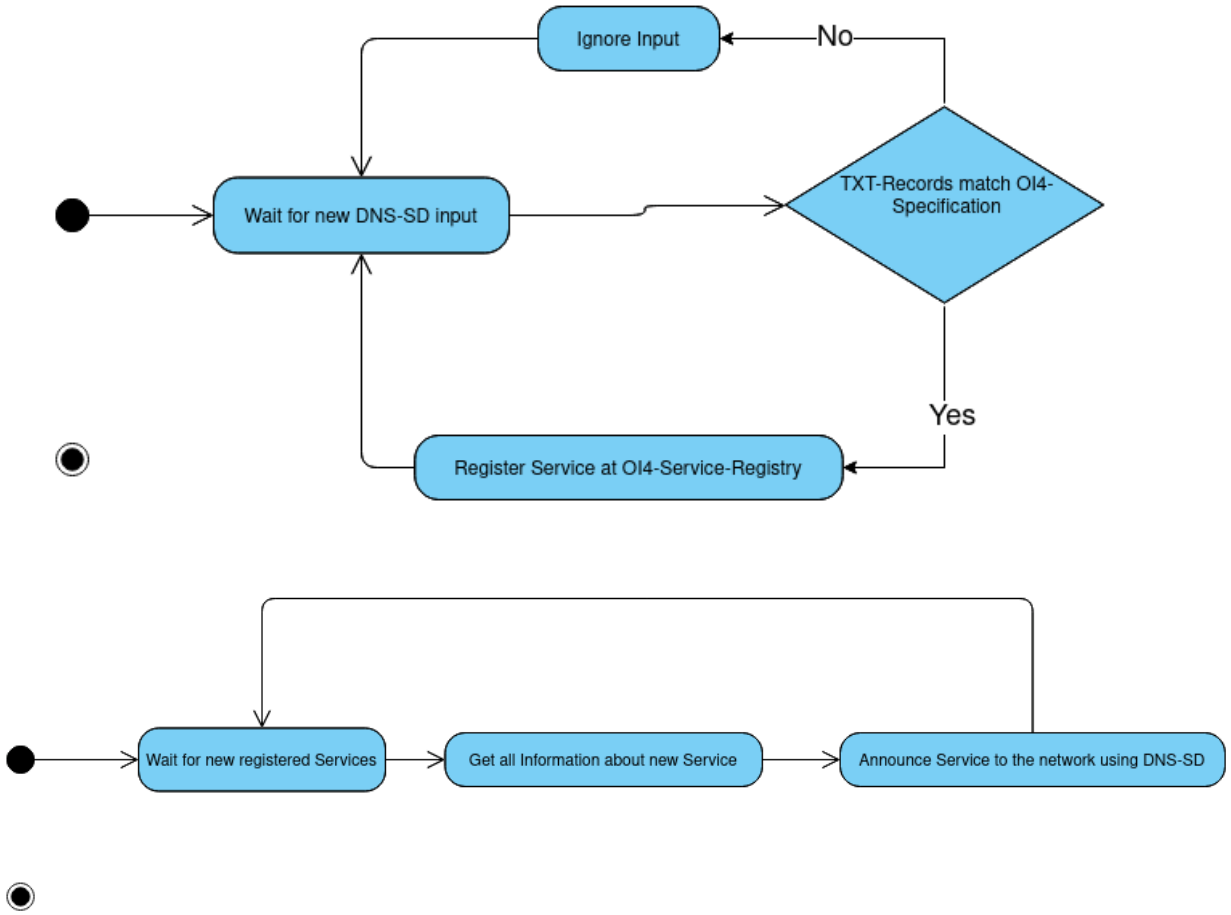


The project consists of two main parts: the Docker-Container connecting the DNS-SD mechanism with the OI4-Service-Registry and the Docker-Container, which tests and showcases the functionalities of the system. To realize a working system, there need to be other applications running in the environment. These are the OI4-Service-Registry developed by the OI4-Alliance and a MQTT-Broker on which the communication with the OI4-Service-Registry takes place. The communication using the MQTT-Protocoll is specified by the OI4-Alliance.

Conceptually the main Docker-Container is just another MQTT-Client. The OI4-Service-Registry also acts as a MQTT-Client. Both communicate with each other over the MQTT-Broker. In most cases the MQTT-Broker of choice will be Eclipse-Mosquitto, as it is available as a Docker-Image and therefore easy to deploy. All Communication over MQTT must be conform with the specifications published by the OI4-Alliance. In the context of the systems specified by the OI4-Alliance this way of communicating is called the OI4-MessageBus. Services can use the MessageBus to be registered at the OI4-Service-Registry. The Test Application shall also be able to connect to the MessageBus, if it is configured to register itself at the OI4-Service-Registry.

The DNS-SD mechanism on the other hand is used to announce important information about a service in the network. It is not limited to services, which are part of the OI4-ecosystem. In this use-case it is not only used to announce the services registered at the Service-Registry, but if configured to, also collect information about not yet registered services, which may announce themselves to the network and register them at the OI4-Service-Registry.

3.1 Architectural Model



The diagrams above show the two main activities the main Docker-Application performs.

The main Docker-Application consists of multiple functions, which can be categorized, depending on the feature they are part of. The features they implement are: Receiving DNS-SD entries and forwarding them to the OI4-Registry, Checking conformity of data before sending it to the OI4-Registry, Announcing Services registered at the OI4-Service-Registry via DNS-SD. As the Application is written in NodeJS it will mostly be an event-based software-architecture reacting to new DNS-SD entries or new services registered at the OI4-Service-Registry.

The Test-Application will also be implemented in NodeJS. It will have somewhat of a data layer consisting of information about the current configuration but also contain a list of current DNS-SD entries being announced over the network. There will also be a layer announcing the Test-Application either using DNS-SD or registering it directly via the OI4-MessageBus, depending on configuration. This and other options can be configured using the environment-variables of the Docker-Container. The information about current DNS-SD entries is updated based on events thrown by the node module 'mdns'. The web-interface then periodically reads these entries from the data-layer and displays them.

4. Subsystem Specifications

4.1 /MOD10/ DNS-SD to OI4-Service-Registry Interface

4.1.1 /MOD11/ DNS-SD listener

/MOD11/	DNS-SD listener
System requirements covered	LF10
Service	Listens to DNS-SD entries on the network and registers them at the OI4-Service-Registry
Interfaces	DNS-SD, OI4-MessageBus
External Data	-
Storage Location	-
Open points	-

4.1.2 /MOD12/ OI4-Service-Registry listener

/MOD12/	OI4-Service-Registry listener
System requirements covered	LF20
Service	Listens to changes in services registered at the OI4-Service-Registry and announces them to the network using DNS-SD
Interfaces	DNS-SD, OI4-MessageBus
External Data	-
Storage Location	-
Open points	-

4.1.3 /MOD13/ OI4-Conformity Validator

/MOD13/	OI4-Conformity Validator
System requirements covered	LF60
Service	This module shall be used, when registering services at the OI4-Service-Registry to ensure that the OI4-Specifications are met
Interfaces	DNS-SD, OI4-MessageBus
External Data	-
Storage Location	-
Open points	-

4.2 /MOD20/ Test Application

4.2.1 /MOD21/ Web-Interface

/MOD21/	Web-Interface
System requirements covered	LF50
Service	Show all services discovered using DNS-SD
Interfaces	Data-Layer, Web-Interface
External Data	-
Storage Location	-
Open points	-

4.2.2 /MOD22/ Announce Service via DNS-SD

/MOD22/	Announce Service via DNS-SD
System requirements covered	LF30
Service	Announce itself to the network using DNS-SD
Interfaces	DNS-SD
External Data	-
Storage Location	-
Open points	-

4.2.3 /MOD23/ Register itself at the OI4-Service-Registry

/MOD23/	Register itself at the OI4-Service-Registry
System requirements covered	LF40
Service	Register the Test-Application at the OI4-Service-Registry
Interfaces	OI4-MessageBus
External Data	-
Storage Location	-
Open points	-

4.2.4 /MOD24/ DNS-SD listener

/MOD24/	DNS-SD listener
System requirements covered	LF50
Service	Listen to DNS-SD entries on the network and syve them to the data-layer
Interfaces	Data-Layer, DNS-SD
External Data	-
Storage Location	-
Open points	-

5. Technical Concepts

5.1 Communication with other IT-Systems

The main Docker-Container shall communicate with the OI4-Service-Registry and use DNS-SD to receive information about previously unknown services. It shall also use DNS-SD to announce services, that are registered at the OI4-Service-Registry, to the network. The Docker-Container for testing purposes shall use DNS-SD to announce itself to the system and use DNS-SD to list all available services in the network.

5.2 Deployment

The Docker-Images used to create the Docker-Containers can be built from the source-code or downloaded via the 'docker-hub'.

5.3 Data Validation

The system shall validate whether the data it receives is conform with the specifications published by the OI4-Alliance and only use valid data.

5.4 Exception Handling

Exceptions shall be logged to the console. If the Exceptions are fatal, the system shall restart to maximize availability.

5.5 Logging

All main events shall be logged to the console. The output on the console could be directed to a file.

5.6 Configurability

Both Docker-Containers shall be able to be configured using environment-variables, these are easily set when starting a Docker-Container or when inside a 'docker-compose' file.

5.7 Internationalization

All documentation and commenting in the source-code shall be done in English, ensuring maximum internationalization.

5.8 Testability

The system will be tested in different environments covering a number of scenarios.

5.9 Availability

Running in a Docker-Container ensures that the application can be restarted instantly if it stops running. All other forms of unavailability will probably arise because of problems in other parts of the environment.

6. References

- SRS: https://github.com/GoranErdeljan/TINF19C-Team-4-Service-Registry/tree/master/PROJECT/SRS

- CRS: https://github.com/GoranErdeljan/TINF19C-Team-4-Service-Registry/tree/master/PROJECT/CRS