HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

caru

CloudEvents Router

Thesis

Student Research Project Thesis

University of Applied Sciences Rapperswil (HSR)
University of Applied Sciences of Eastern Switzerland (FHO)
Department of Computer Science

Fall Term 2019/2020

Authors	Linus Basig, Fabrizio Lazzaretti
Advisor	Prof. Dr. Olaf Zimmermann
Project Partner	CARU AG

Printing Date	December 19, 2019

Abstract

CARU is an AgeTech startup that builds an internet-connected device to help the elderly to live independently for longer. CARU's software architecture is heavily event-driven. The events are structured according to the CloudEvents specification of the Cloud Native Computing Foundation. The vision of CARU is a unified event plane where events can flow between all connected systems. To route the events between the systems on the device, in the cloud, and potentially even from third parties, this thesis introduces the CloudEvents Router. The router uses the fields defined by the CloudEvents specification to decide where to forward the events to.

As a starting point, we conducted extensive market research and found one solution with great potential: Apache Camel. Unfortunately, Camel did not meet all the resource and portability requirements. To create a solution that fulfills all these requirements, we designed, prototyped, and implemented our solution.

The primary result is an open-source CloudEvents Router proof of concept implementation in the Rust programming language. To achieve the desired portability, we based the architecture on an event-driven Microkernel, which supports the platform-specific implementation of individual components. The second deliverable is a report about our experience of using the Rust programming language for the first time. After a frustrating beginning, we started to enjoy using it because of the safety guarantees of its ownership model, the exceptionally helpful compiler, and the fantastic tooling.

Management Summary

Context and Problem

CARU AG is an AgeTech startup with the mission to help the elderly to live independently for longer. Its internet-connected device allows calling for help by voice command and offers data-driven functionalities powered by its many sensors. The different software components at CARU communicate mostly over events; whenever something notable happens, an event is published. Other components react to these events and create new events themselves. Most events are structured according to the CloudEvents specification of the Cloud Native Computing Foundation. By using the same event structure everywhere, CARU wants to create a unified event plane and simplify the communication between the different software systems. These include the ones on the device, in the cloud, and potentially even from third parties Always broadcasting each event to all systems would be inefficient and cause security risks.

Methods and Results

We address this shortcoming by introducing the CloudEvents Router. The CloudEvents Router is responsible for deciding which event should be shared with which system. This decision is based on the meta-data defined by the CloudEvents specification.

As a starting point, we conducted extensive market research to assess already existing products with the potential to solve this problem. We found one solution with great potential; unfortunately, the solution was incompatible with the resource limitations of CARU's device. We then design, implemented and open-sourced a CloudEvents Router proof of concept in the Rust programming language.

The second deliverable was a report on our experience using the Rust programming language for the first time. Rust has many features and concepts to ensure that the code written in it is safe and contains fewer bugs than other performance-oriented programming languages. These features and concepts cause a steep learning curve in the beginning. Fortunately, the exceptionally helpful tooling provided by the Rust Team enables a quick path to efficiency.

Future Work

With our thesis, we could show what the design and implementation of an extendable and portable CloudEvents Router could look like. The CloudEvents Route lays a solid foundation for CARU to bring their vision of a unified event plane into reality. To advance the vision, we propose three topics to consider for future work:

- Invest in the improvement of the Rust CloudEvents Software Development Kit to make the use of CloudEvents even more efficient.
- Extend the CloudEvents Router to preserve advanced delivery guarantees of the transport mechanisms it uses to receive and send CloudEvents.
- Extend the CloudEvents Router to integrate with monitoring solutions from the Cloud Native Computing Foundation.

Acknowledgments

First of all, we would like to thank our advisor, Prof. Dr. Olaf Zimmermann for his exceeding support and guidance.

We are also thankful for Reto Aebersold and Thomas Helbling from CARU for sponsoring this thesis and their invaluable input.

Lastly, we are incredibly grateful for our patient friends, colleages and partners, especially Hannah Li Hägi and Gloria Shi, for their feedback and proofreading.

Contents

1. Introduction

The Serverless Working Group of the Cloud Native Computing Foundation (CNCF) has proposed CloudEvents, a specification to describe events. This specification seeks to ease event declaration and delivery across services, platforms, and vendors. It is attracting lots of attention and contributions from major cloud providers and Software as a Service (SaaS) companies.[1]

The industry partner for this student project thesis is CARU AG. The company is an AgegTech startup with the mission to help the elderly live independently for longer. Its Internet of Things (IoT) device, the CARU Smart Sensor, allows calling for help by voice command or touch (Figure 1.1). In addition to this base feature, the device is equipped with a multitude of sensors that allow a variety of data-driven functionalities. To access these services, CARU offers a web-based application for relatives and professional caretakers.

Figure 1.1.: The CARU Smart Sensor placed in the home of a senior citizen[2]

The software architecture used by CARU is heavily event-driven. This is to say that the different components of their software landscape publish events when something notable happens. In turn, other components act on these events and publish new events themselves.

Figure 1.2 shows CARU's current system context. CARU is already using CloudEvents format for the events exchanged between its software components in the cloud. Simple Notification Service (SNS) provided by Amazon Web Services (AWS) is used to exchange the events in the cloud. Currently, the software on the CARU Smart Sensor does not use the CloudEvents format. The business logic on the CARU Smart Sensor is split into multiple services and runs in a single process (the CORTEX Process). The services exchange events over an in-process bus. When events need to be exchanged with other processes on the device, an inter-process communication mechanism (e.g. UNIX socket, Serial Interface) with a non-standardized format is used. When events are exchanged with the cloud, they are sent over Message Queuing Telemetry Transport (MQTT) to a MQTT broker provided by AWS in a non-standardized format. These events need to be converted to CloudEvents before they can be used in the cloud.

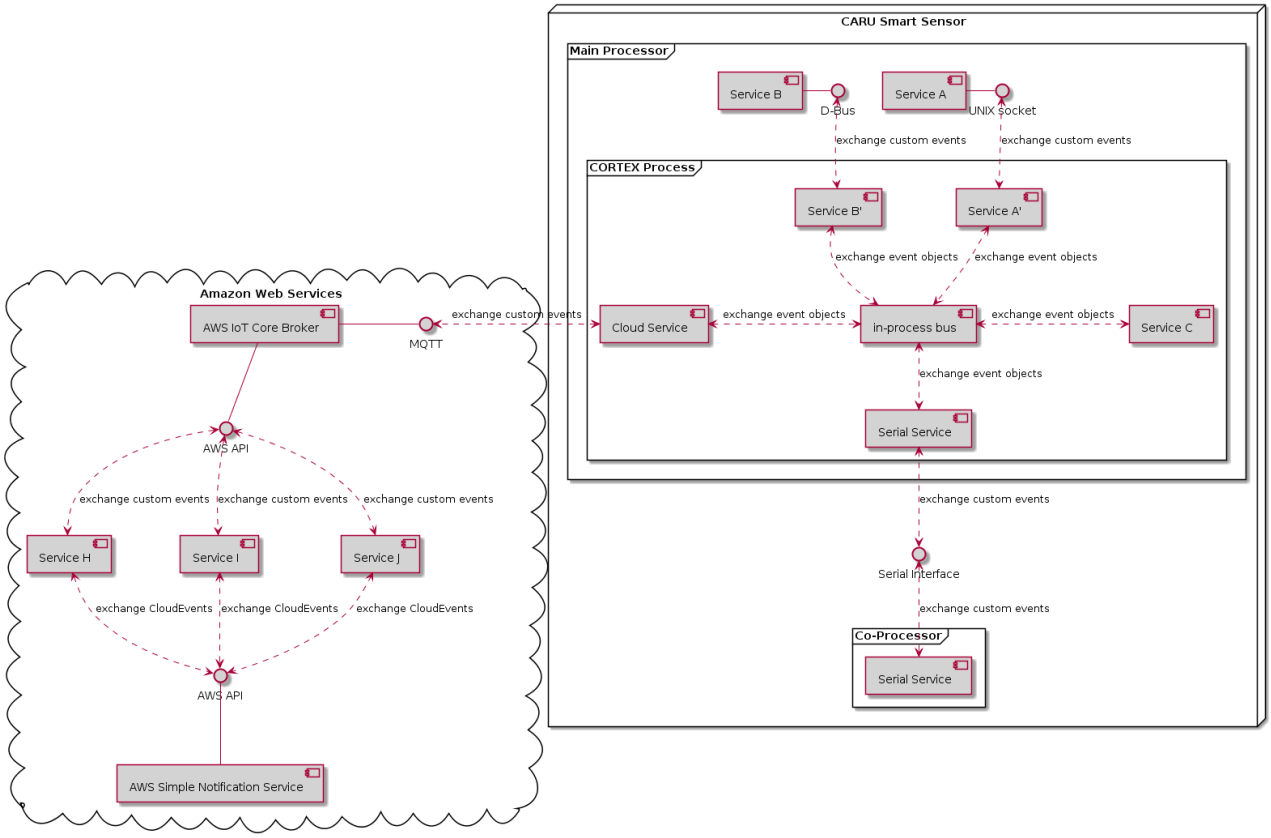

Figure 1.2.: CARU's Current System Context

1.1. Vision

CARU wants to create a unified event plane where events can flow between all connected systems. These include the ones on the device, in the cloud, and potentially even from third parties. To achieve this, CARU considers to embrace CloudEvents not only in the cloud but also on their devices.

However, always broadcasting each event to all systems would be inefficient and cause security risks. In this thesis the creation of a CloudEvents Router is proposed to circumvent these problems. It would be the router's task to route the events between the different systems based on the fields defined by the CloudEvents specification.[3]

Figure 1.3 shows CARU's envisioned system context with the unified event plane implemented. Whenever events are exchanged, they are formatted according to the CloudEvents specification. The CloudEvents Routers are placed in each system and connected over an appropriate transport protocol to its neighbours. Consequently, the services are then also connected to their CloudEvents Router over an appropriate transport protocol. In some cases, the appropriate transport protocols have not been determined yet and their definition is out of the scope of this thesis.

In addition to the unified event plane, CARU wants to explore other programming languages to complement its dominant use of Python. CARU proposed Rust as their preferred programming language for the implementation of the CloudEvents Router. The goal is to get a first impression of the fit into their existing ecosystem. The reported experience during this thesis will influence CARUs decision on whether or not they are going to further explore Rust for wider usage.

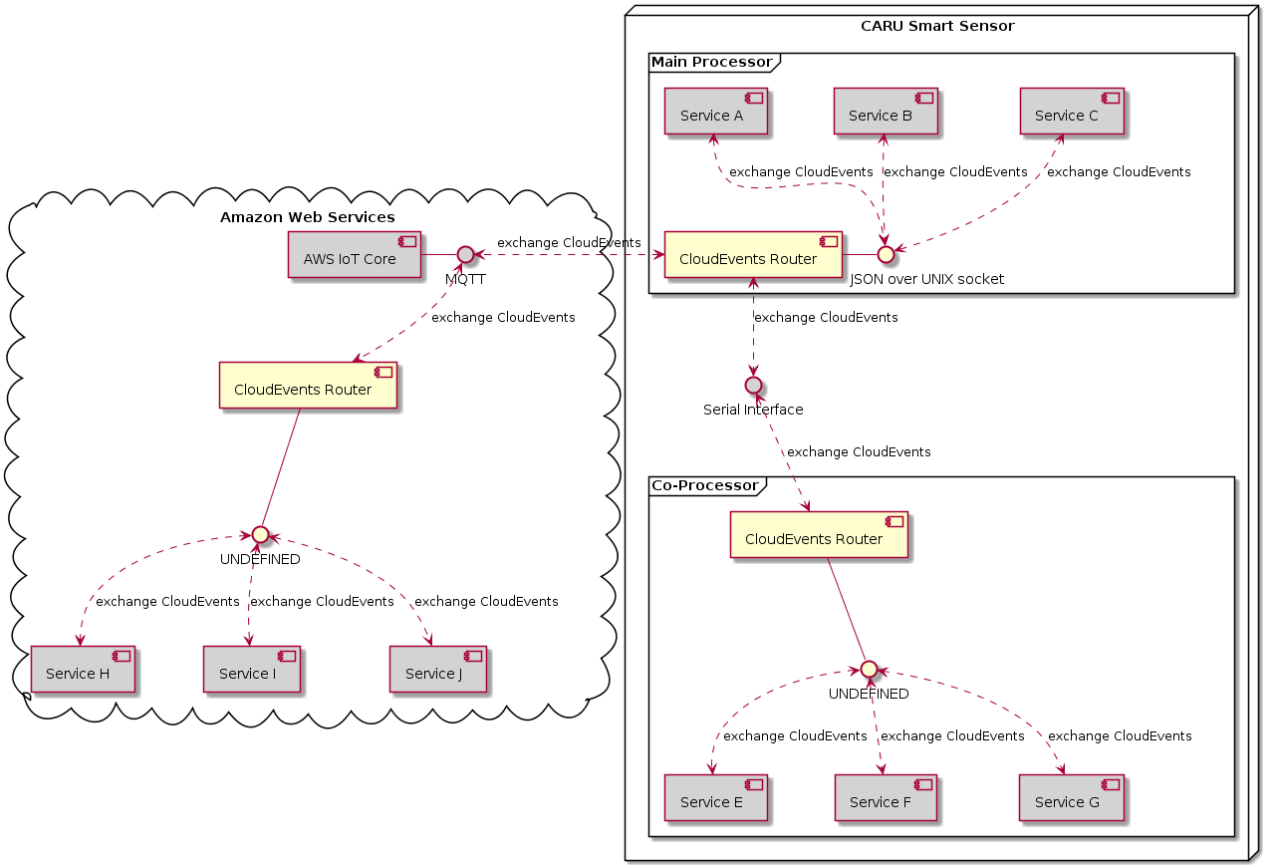

Figure 1.3.: CARU's Envisioned System Context

1.2. Structure

The following chapter (Market Analysis) will explore the market of already existing solutions capable of routing events. It compares the different solutions and elects the main competitor for our proposed solution.

In chapter Methods the tools and techniques used to find our solution are discussed.

Then the Results and Discussion chapter presents our solution and the most important aspects of it. It also summarizes the comparison of our solution with its main competitor established in the Market Analysis chapter.

Finally, Conclusion will summarize our learnings and proposes the next steps for the development of the CloudEvents Router.

Additionally, we provide the architecture documentation and a Rust experience report in separate documents.

2. Market Analysis

CARU wishes to get an overview of the existing solutions that have the potential to solve the problem stated in the previous chapter.[4]

First, we conducted a high-level comparison of a wide selection of possible solutions.

In a second step, we took an in-depth look at the most promising solutions and select one as the reference point for our implementation.

2.1. Competing Products

The products in the following subchapters were selected as potential solutions for the stated problem.

The list is not exhaustive. Especially enterprise service busses and offerings from public cloud providers other than AWS (e.g. Microsoft Azure, Google Cloud Platform) fit the stated problem space but are excluded because they are not interesting for CARU.

The products are primarily sorted by their native CloudEvents support and secondarily alphabetically.

2.1.1. CloudEvent Router and Gateway

The CloudEvent Router and Gateway is a simple proof of concept of a gateway that is routing CloudEvents between different sources and sinks. It implements HyperText Transfer Protocol (HTTP) and Kafka sources and sinks. [5]

It is a proof of concept created by a single developer (Matthias Wessendorf). It is not versioned and has no known productive users. It is distributed under the Apache 2 license. [5]

2.1.2. Knative Eventing v0.9

Knative Eventing is a set of loosely coupled services that route CloudEvents from producers to interested consumers. It is tightly integrated into Kubernetes.[6]

Knative Eventing is part of Knative, which is driven by companies like Google, IBM, Red Hat, and Pivotal. Knative is a Function as a Service (FaaS) platform on top of Kubernetes. Knative Eventing is one way to invoke functions deployed with Knative.[7, 8]

Knative is currently pre-1.0 and therefore not considered production-ready.[8]

Knative Eventing and Knative are distributed under the Apache 2 license.[9, 10]

2.1.3. Pacifica Dispatcher v0.2.3

Pacifica Dispatcher is a set of Python modules to receive and handle CloudEvents.[11]

Pacifica Dispatcher is part of Pacifica, which is an open-source scientific data management platform for harvesting, validating, and distributing data and metadata. Pacifica version 1.0 was released but no information about its users could be found.[12]

Pacifica is developed mainly by David Brown and Michael Hofmockel.[13]

Pacifica is distributed under the GNU 2 license.[14]

2.1.4. Serverless Event Gateway v0.9.1

The Serverless Event Gateway is a set of components that route CloudEvents to functions deployed by the Serverless Framework. The Serverless Framework is a FaaS abstraction layer. It enables developers to write code once and deploy it to multiple FaaS providers.[15]

The Serverless Event Gateway is developed by Serverless Inc. and not yet production-ready.[15]

The Serverless Event Gateway is distributed under the Apache 2 license.[16]

2.1.5. Amazon Simple Notification Service

Amazon SNS is a fully managed publish/subscribe messaging system. It allows the filtering based on meta-data sent with the messages (it has no native CloudEvents support).[17]

Amazon SNS is one of the core products of AWS. NASA and Futbol Club Barcelona are mentioned as example customers.[18, 19]

2.1.6. Apache Camel v2.24.2

Apache Camel is a Java framework to integrate various systems producing and consuming data. It comes with hundreds of connectors and supports around 50 data formats (but has no native CloudEvents support).[20]

Apache Camel is over 10 years old and there are many projects and products which use Apache Camel internally or integrate themself with it. One recognizable example of a company that uses Apache Camel is Netflix.[21]

There are also many consulting firms offering support for Apache Camel. This is an indication for us that Apache Camel has many active users.[22]

The development of Apache Camel is overseen by the Apache Software Foundation and it is distributed under the Apache 2 license.[20, 23]

2.1.7. Crossbar.io v19.10.1

Crossbar.io is a networking platform that allows applications the exchange of messages over the Web Application Messaging Protocol (WAMP). It was built with IoT in mind and supports many programming languages (but has no native CloudEvents support).[24, 25]

Crossbar.io has some lesser-known production users like Genesi or RECORD.evolution and is a Siemens MindSphere gold partner. Crossbar.io GmbH, the company behind Crossbar.io, is an industry partner of project PRODISYS. Project PRODISYS is a research project in the area of industrial IoT and is sponsored by the German Federal Ministry of Education and Research.[26–28]

Crossbar.io is distributed under the GNU 3 license.[29]

2.1.8. D-Bus v1.12

D-Bus is a message bus for interprocess communication. It supports one-to-one and one-to-many message passing via the D-Bus protocol. It works on Linux and UNIX systems. It uses Extensible Markup Language (XML) files for configuration.[30–32]

D-Bus is part of the freedesktop.org suite of tools to build interoperable open-source graphical and desktop systems. Many linux desktop systems like GNOME, KDE or XFCE are built on top of the freedesktop.org suite. Most applications built for these desktops use D-Bus to interact with the desktop and other applications. There are also many low level services, like NetworkManager or ModemManager, which expose their Application Programming Interface (API) over D-Bus.[33–36]

D-Bus can be used under the Academic Free License v2.1 or GNU 2.[37]

2.1.9. Node-RED v1.0.1

Node-RED is a Node.js tool for wiring hardware devices, APIs, and online services together. It was built with IoT in mind and offers thousands of components (but has no native CloudEvents support).[38]

Node-RED is used in production setups. It is a part of the IBM Watson IoT platform's developer experience. There are also hardware products that embedded Node-RED.[39–42]

The development of Node-RED is overseen by the OpenJS Foundation and it is distributed under the Apache 2 license.[38, 43]

2.1.10. RabbitMQ v3.8

RabbitMQ is an open-source message broker. It initially only supported Advanced Message Queuing Protocol (AMQP). Today, it also supports Simple Text Oriented Messaging Protocol (STOMP), MQTT, HTTP and WebSockets to receive and send messages.[44, 45]

RabbitMQ is over 10 years old and has many users: T-Mobile, AT&T, VMware, Mozilla, OpenStack, and the government of India are only a few examples.[44, 46]

RabbitMQ is maintained by Pivotal Software, a big software consulting company, and is distributed under the Mozilla Public License (MPL)[47] 1.1 license.[44, 48]

2.2. High-Level Comparison

The goal of the high-level comparison is to identify the top performers. Rather than finding the ”best” solution by simply adding up the points, the listing is meant to serve as a guide during the more in-depth comparison

We will do a high-level comparison of the solutions using two different contexts: CARU intends to use CloudEvents in the cloud and on their devices.

The two environments have different requirements and constraints. We will have a look at routing in the cloud in subsection 2.2.1 and in subsection 2.2.2 at routing on the device. The next section contains the criteria we used to compare the solutions.

Criteria

The following criteria will be used to compare the solutions:

XOR-criteria only one option at the time is applicable for each solution

SUM-criteria multiple options can be applicable simultaneously and their points get summed up

Name	Type	Description
CloudEvents Support	XOR	The product should support the CloudEvents standard. **3 points** Native CloudEvents support **2 points** No native support but CloudEvents can be routed with existing configuration options **1 points** No native support but CloudEvents can be routed with additional coding effort
Cloud Connector Support	SUM	The product should support multiple general and cloud-specific protocol connectors. **1 point** Native MQTT v3.1.1 support **1 point** Native HTTP support **1 point** Native Amazon SQS support **1 point** Implementation of own connector possible
Device Connector Support	SUM	The product should support multiple general and device-specific protocol connectors. **2 points** Native MQTT v3.1.1 support **2 points** Native UNIX socket support **1 point** Native HTTP support **1 point** Implementation of own connector possible

Routing Options	SUM	The product should support the definition of routing rules. **1 point** Contend-based routing rules (CloudEvents data field) **1 point** Header-based routing rules (any CloudEvents field) **1 point** Event-name-based routing rules (CloudEvents type field)
Cloud Compatibility	SUM	The product should be easily integrated into the existing cloud environment. **1 point** Official Docker container available **1 point** How to deploy to AWS documented **1 point** How to run in Linux documented
Device Compatibility	SUM	The product should be easily integrated into the existing device environment. **1 point** How to run in Linux documented **1 point** How to integrate into FreeRTOS documented
Documentation Quality	XOR	The product should be well documented. **4 points** Excellent quality with many examples **3 points** High perceived quality **2 points** Acceptable perceived quality **1 points** Any documentation
Project Maturity	XOR	The product should be mature. **3 points** High maturity (5+ years of active development) **2 points** Mature (3-5 year of active development) **1 point** Newcomer (1-3 year of active development)
Project Vitality	XOR	The product should have active maintainers. **3 points** More than 50 closed issues in the last month **2 points** 25 - 50 closed issues in the last month **1 point** 5 - 25 closed issues in the last month

Community	XOR	The product should have an active community. **3 points** More than 500 Stack Overflow questions with accepted answers **2 points** More than 100 Stack Overflow questions with accepted answers **1 point** Any Stack Overflow questions with accepted answers
License	XOR	The product should be open-source. **3 points** Open-source license without a copyleft clause **1 points** Open-source license with a copyleft clause

Table 2.1.: Criteria List for the High-Level Comparison

2.2.1. Routing in the Cloud

Product Name	CloudEvents Support	Cloud Connector Support	Routing Options	Cloud Compatibility	Documentation Quality	Project Maturity	Project Vitality	Community	License	Total[1]
CloudEvent Router and Gateway [5]	3	2	1 [2]	0	1	0	0 [3]	0	3	10
Knative Eventing [6, 50]	3	3 [4]	2 [5]	2 [6]	3	1	2	1 [7]	3	20
Pacifica Dispatcher [11, 54, 55]	3	1	3	1	2 [8]	0	0	0	3	13
Serverless Event Gateway [15]	3	1	1	3	2	1	0	3 [9]	3	17
Amazon Simple Notification Service[17]	2 [10]	4	1 [11]	1	3	3	? [12]	3 [13]	0	17
Apache Camel [20]	1	4	3	2 [14]	3	3	1	3 [15]	3	23
Crossbar.io [24, 25, 60]	0	0	1 [16]	3	3	3	1	1	1	13
Node-RED [38, 61, 62]	0	3 [17]	3	3	3	3	1	3	3	22
RabbitMQ[44, 45]	0	2	2 [18]	3 [19]	4	3	1	3	1 [20]	19

Table 2.2.: High-Level Comparison for Routing in the Cloud

[1] the total sum should help to identity the top performering solutions not the "best" one.
[2] routing is based on event type [49]
[3] no issues at all
[4] HTTP, SQS and custom implementation[51]
[5] header-based and event-name-based routing rules [52]
[6] official docker container and possible to deploy to AWS Elasitc Kubernetes Service (EKS)
[7] 16 answered questions on 09.10.2019[53]
[8] example provided, but only specific use case, to use with Pacific products
[9] the serverless framework community is very active, impossible to filter the serverless eventing community
[10] CloudEvents fields need to be passed as message attributes when publishing[56]
[11] header-based routing only[57]
[12] issues are not public
[13] over 1000 answered questions on 09.10.2019 [58]

2.2.2. Routing on the Device

Product Name	CloudEvents Support	Device Connector Support	Routing Options	Device Compatibility	Documentation Quality	Project Maturity	Project Vitality	Community	License	Total[21]
CloudEvent Router and Gateway [5]	3	2	1 [22]	0	1	0	0 [23]	0	3	10
Pacifica Dispatcher [11, 54, 55]	3	1	3	1	2 [24]	0	0	0	3	13
Apache Camel [20]	1	4	3	1	3	3	1	3 [25]	3	22
Crossbar.io [24, 25, 60]	0	2 [26]	1 [27]	1	3	3	1	1	1	13
D-Bus[31, 32, 67]	0	2 [28]	1	1	3	3	1	2	1	14
Node-RED [38, 61, 62]	0	4 [29]	3	1	3	3	1	3	3	21
RabbitMQ[44, 45, 66]	0	3	2 [30]	1	4	3	1	3	1 [31]	18

Table 2.3.: High-Level Comparison for Routing on the device

[14] no Docker container as it is a framework
[15] over 5000 answered questions on 09.10.2019[59]
[16] only Event-name-based routing rules
[17] SQS packages are available, but only from third party → not native
[18] header-based routing is possible with routing_key filtering[63] or topic exchanges[64]; Event-name-based routing could be realized with different exchanges
[19] multiple official deployment solutions exist for AWS[65, 66]
[20] licensed under MPL which is a copyleft license[47]
[21] the total sum should help to identity the top performering solutions not the "best" one.
[22] routing is based on event type [49]
[23] no issues at all
[24] example provided, but only specific use case, to use with Pacific products
[25] over 5000 answered questions on 09.10.2019[59]
[26] it allows WAMP over UNIX socket [60]
[27] only Event-name-based routing rules
[28] dbus work with UNIX socket and Transmission Control Protocol (TCP)
[29] UNIX socket packages are available, but only from third party → not native
[30] header-based routing is possible with routing_key filtering[63] or topic exchanges[64]; Event-name-based routing could be realized with different exchanges
[31] licensed under MPL which is a copyleft license[47]

2.3. In-Depth Comparison

The goal of the in-depth comparison is to find the best performing solution from the high-level comparison. In a later step, we compare the winner of the in-depth comparison with our own solution.

2.3.1. Apache Camel

For a product overview please refer to subsection 2.1.6.

Apache Camel is implemented in Java, which is a strongly-typed language that gets compiled to Java bytecode. This bytecode is then interpreted and executed by the platform-specific Java Virtual Machine (JVM) implementation.

Apache Camel can receive, process and forward data. Apache Camel uses Java code to describe where the data is coming from, what should be done with the data (e.g. transform, filter) and where the data should go afterwards. This code gets compiled and deployed. [68] The core package has only a handful of dependencies (details about the dependency resolution can be found in Appendix I)

Apache Camel works with Java Development Kit (JDK) 1.5 and up and has only three external dependencies. (see Listing 6)

Installation and First Tutorial

To get a better understanding of Apache Camel, we followed the official "WALK THROUGH AN EXAMPLE CODE" tutorial.[69, 70]

The installation was straight forward: After Java and Maven were installed the example can be compiled with a simple command.

The tutorial code is clear and easy to follow.

The code for the tutorial can be found in the Appendix F.

2.3.2. Node-RED

For a product overview please refer to subsection 2.1.9.

Node-RED is implemented in JavaScript, which is a loosely-typed and interpreted language. Therefore, it can be deployed to any platform that supports the Node.js runtime. The core package has already hundreds of dependencies (dependency resolution is described in Appendix J).

Node-RED can receive, process (transform and filter) and forward data as well. Node-RED offers a graphical interface to configure the data flow. The configuration can be exported (JavaScript Object Notation (JSON) format) and deployed.

Node-RED works with Node.js version 8 and 10 (recommended).[71]

Installation and First Tutorial

To get a better understanding of Node-RED, we followed the official "Creating your first flow" tutorial.[72]

The installation was straight forward as well: After Docker was installed, the example can be run with a simple command.

It was possible to follow the instructions, but the inclusion of example images throughout the tutorial would have improved ease of use.

The code for the tutorial can be found in the Appendix G.

2.3.3. Knative Eventing

For a product overview please refer to subsection 2.1.2.

Knative Eventing is implemented in Go, which is a strongly-typed and compiled language. The documentation claims that it is a set of loosely coupled services that can be developed and deployed independently across a variety of platforms. But there are no instructions available on how to use it with any other platform than Kubernetes.[6, 50, 73]

Knative Eventing receives CloudEvents over one of its sources, routes them and forwards them to the interested Knative Functions. [51, 52, 74]

Installation and First Tutorial

To get a better understanding of Knative Eventing, we followed the official "Getting Started" tutorial.[75]

The installation guide has many steps and the first and second attempt at installation and following the tutorial step by step was unsuccessful (for details see Appendix H).

There are many outdated code examples in the documentation and in the example repository which are not working with the current version of Knative Eventing.[52, 76]

Without existing knowledge of Kubernetes we probably would not have been able to successfully install it by following the installation guide and examples.

2.3.4. Summary

Despite the good first impression, Knative Eventing did not perform as expected. Because it requires a Kubernetes cluster to run, and CARU does not have one, nor do they plan to use one in the near future, we can not recommend it.

Node-RED and Apache Camel are both mature projects and the first steps with them were very pleasant. We believe that both are capable of solving the stated problem. However, there are some aspects which lead us to prefer Apache Camel over Node-RED.

The target audience of the two projects seems to be different. Apache Camel is built around Enterprise Integration Patterns and therefore targets commercial and enterprise users. Although some commercial products are integrating Node-RED, the product itself is, with its configure by UI approach, optimized to be accessible by a broad spectrum of users. This is not a bad thing by itself, but it attracts lots of hobbyist programmers. Additionally, Node-RED only provides a small set of core adapters and all the others are contributions from outside of the project (with widely varying quality). This makes it

more difficult to ensure high and consistent quality. The more professional target audience and the tighter controlled component library lead us to favor Apache Camel. [20, 42, 77–79]

Another significant difference between the two projects is the used programming language. The type-safety of Java and the leaner dependency tree provided another advantage of Apache Camel.

The last point in favor of Apache Camel is the way configuration is handled. With its configuration as code approach, it is extremely easy to version it with established engineering practices (e.g. Git). In theory, the JSON export of the Node-RED configuration can also be versioned with Git, but it is very noisy because it also contains data required for its presentation in the graphical editor.

2.4. Conclusion

Despite the good impression Apache Camel made in the high-level and low-level comparisons, it can not deliver on all requirements. It does not support CloudEvents natively and running in a JVM creates significant overhead.

The extensibility of Apache Camel would allow the support of CloudEvents by implementing a type converter.[80]

There is a version of Apache Camel which runs on Quarkus. Quarkus is a Java platform offering fast boot times and low memory footprint. This would make it easier to run Apache Camel on the resource-constrained CARU device. The downside is that the feature-set of Apache Camel Quarkus is very limited compared to the full version of Apache Camel.[81, 82]

We decided to implement a Rust-based CloudEvents router. Our hypothesis is that we are able to build a leaner solution with a better problem fit and performance in resource-restricted environments. Apache Camel will be revisited in the performance verification section.

3. Methods

In this chapter, we describe the process of establishing our solution and give a summary of what we did in each step.

We loosely based our process on the Open Unified Process. The Open Unified Process defines four phases: inception, elaboration, construction, and transition. The inception phase is about finding out what to build. The elaboration phase is about understanding the requirements in detail and defining the software architecture. The construction phase is about the implementation of the solution. The transition phase is about validating and releasing the project results.[83]

The two most important phases for us were the elaboration and the construction phase. In the elaboration phase, we quickly iterated between the creation of comprehensive architecture documentation and the prototyping of certain aspects. After the elaboration phase ended, we implemented the designed solution while applying pair-programming.

3.1. Architecture Documentation

One central piece of our process was the creation of the architecture documentation based on the arc42 template from Gernot Starke and Peter Hruschka. The arc42 template provides a practical and pragmatic structure which guides through the process of creating a comprehensive software architecture documentation.[84]

In the following subsections, we will give a short introduction to each chapter suggested by the arc42 template. Additionally, we summarize our approach and share the insights we gained while writing them. The full architecture documentation with all the details can be found in Appendix B.

3.1.1. Introduction and Goals

The first chapter described by arc42 is about gathering the requirements and driving forces that are relevant for the solution. The template distinguishes between functional (business goals) and non-functional requirements (quality goals). There is also room for a list of stakeholders in this section.[85]

In order to find the relevant functional requirements, we collected a variety of use cases from the customer. Use cases are descriptions of situations in which the solution would provide value to the user. After collecting these use cases, we prioritized and reduced them with the customer to a list of 6 functional requirements (FR-1 - FR-6). Because of the limited time and resource available for this project, we had to limit the scope and could not consider all of the use cases. Table 3.1 shows the list of functional requirements.

No	Requirement
FR-1	The router must understand the CloudEvents specification version ~~0.3~~ 1.0.[1]
FR-2	The router must be able to run on a modern Linux distribution
FR-3	A simple way to define routing rules based on the CloudEvents attributes must be designed
FR-4	The router should offer a UNIX socket interface to send and receive CloudEvents
FR-5	The router should offer a MQTT version 3.1.1 interface to send and receive CloudEvents
FR-6	The router should provide a simple interface to access debugging information

Table 3.1.: Functional Requirements

The gathering of the non-functional requirements was comparably easy. We could derive them from the project proposal and problem definition document. Table 3.2 shows the list of non-functional requirements.

No	Goal	Scenario
NFR-1	Modularity	It should be easy to implement additional ports and configuration loaders.
NFR-2	Cross Platform	It should be easy to deploy the router to different environments especially to the cloud and embedded Linux platforms.
NFR-3	Open Source	Easy findable and followable documentation to setup the project and run it.

Table 3.2.: Nonfunctional Requirements

[1] CloudEvents specification version 1.0 was released on the 24. of October and it was decided with the customer to support it.

3.1.2. Architecture Constrains

The second chapter described by arc42 is about given facts that constrain the solution. The template proposes the distinction between technical, organizational, and political constraints.[86]

Fortunately, there were only a few constraints that we could derive from the project proposal and the problem definition document. The most notable constraints are the choice of the programming language Rust, the code versioning system, namely GitHub, and Apache 2, the license for the created software.

3.1.3. System Scope and Context

The third chapter described by arc42 is about neighboring systems and their interaction with the system in development. The template proposes the distinction between business (domain-specific inputs and outputs) and technical (channels, protocols, hardware) context.[87]

Our solution is a piece of integration software; hence, the business and technical context are the same. It is all about channels, protocols, and hardware.

We could derive the required information from the requirements and the use cases from the first chapter of the arc42 document. Usually, human users play an essential role in the system context, but in our case, the only users of our system are other systems. The resulting context diagram is shown in Figure 1.3 (in chapter 1: Introduction).

3.1.4. Solution Strategy

The fourth chapter described by arc42 is about the ”fundamental decisions and solution strategies that shape the system's architecture”. This chapter includes technology decisions and high-level architectural patterns and how they help to achieve critical non-functional requirements (quality goals). There is also space for organizational decisions like the development process.[88]

Our most notable solution strategies are the use of the Content-Based Router pattern, the Microkernel pattern, prototyping, and pair-programming. The Microkernel helps to addresses the non-functional requirements ”Modularity” and ”Cross Platform” (see Table 3.2). Prototyping and pair-programming are described in more detail later in this chapter.

3.1.5. Building Block View

The fifth chapter described by arc42 is about ”the static decomposition of the system into building blocks as well as their dependencies”.[89]

This chapter is probably the arc42 chapter, which has undergone the most iterations during the length of the project. We started with a quite simple design, but as soon as we began working on the prototype, the decomposition grew and grew. The evolution of the building blocks are shown in Appendix E.

This was mainly due to our classical, object-oriented approach. The problem was that in the beginning, it looked like Rust fully supports this approach. However, as we got to know the language better, we realized that using a classical object-oriented approach with heavy use of inheritance does not work as well with Rust as with other languages. More details can be found in the Rust experience report in Appendix C.

We changed our approach and used a more functional programming style, which reduced the complexity noticeably. In the end, our high-level decomposition was even more straightforward than our initial draft. Figure 4.1 shows the final building block diagram (see chapter 4).

3.1.6. Runtime View

The sixth chapter described by arc42 is about the ”concrete behavior and interactions of the system’s building blocks”. The goal is to show how the building blocks interact to achieve the functional and non-functional requirements.[90]

We focused on five core scenarios and illustrated them with sequence diagrams. Our five core scenarios are the bootstrapping process, the initialization of the Microkernel, the initial configuration, the routing of an event, and the configuration update. Figure 3.1 shows all the runtime scenarios.

Figure 3.1.: Runtime Scenarios

3.1.7. Deployment View

The seventh chapter described by arc42 is about ”the technical infrastructure used to execute your system”. The chapter should list the technical infrastructure and the mapping of the building blocks onto these infrastructure elements.[91]

The creation of an exact deployment concept is out of the scope of this project. Instead, we collected the specifications of the platforms used at CARU. We kept them in mind while developing the solution and made sure it will be runnable on them.

3.1.8. Crosscutting Concepts

The eighth chapter described by arc42 is about the ”overall, principal regulations and solution ideas that are relevant in multiple parts of your system.”. This chapter can include domain models, patterns and implementation rules.[92]

In this chapter we summarized the Microkernel Pattern from the Pattern-Oriented Software Architecture book and defined some implementation rules.

3.1.9. Architecture Decisions

The ninth chapter described by arc42 is about ”important, expensive, large scale or risky architecture decisions including rationals”. The goal is to make these decisions comprehensible and retractable for all stakeholders.[94]

After comparing the Architecture Decision Record (ARD), Markdown Architectural Decision Records (MADR) and Y-Statement templates, we decided to use the MADR template to document our architectural decisions. We chose MADR because of its dedicated section about the comparison of the considered options, the excellent documentation, and the open license (CC0). Because we used LaTeX for our architecture documentation, we adapted the MADR template for the use with LaTeX instead of Markdown.[95–97]

We created architecture decision records for our decisions to use

- MADR to structure our architectural decisions
- Rust as the programming language
- the Microkernel pattern to achieve portability
- a threading model for parallelism
- the broker pattern with channels for in-process communication

3.1.10. Quality Requirements

The tenth chapter described by arc42 is about ”all quality requirements as quality tree with scenarios”. This chapter adds more details to the quality goals defined in the Introduction and Goals chapter. It can contain additional quality goals with lower priority than the ones listed in the Introduction and Goals chapter.[98]

Because of the limited scope and time of the project, we decided to focus on landing zones instead of a quality tree with scenarios. Landing zones add more specific details to the non-functional requirements from the first architecture documentation chapter. For example, we split the non-functional requirement NFR-2 ”Cross Platform” into a landing zone with support for Ubuntu Linux as the minimum, support for embedded Linux as target and FreeRTOS as outstanding result. Table 3.3 lists all our landing zones with their minimum, target, and outstanding result.

No	NFR	Description	minimum	target	outstanding
LZ-1	NFR-2	Supported operating systems	Ubuntu Linux	Embedded Linux	FreeRTOS
LZ-2	NFR-3	Easy findable and followable documentation to setup the project and run it.	README document exists	Documentation exist and contains clear setup and run documentation.	Multiple tutorials for different adapter setups are provided.
LZ-3	-	Events processed on a normal computer	5/s [2]	500/s [3]	1500/s [4]
LZ-4	-	Average event size	1 KByte	64 KByte [5]	64 KByte

Table 3.3.: Landing Zones

3.1.11. Risks and Technical Debt

The eleventh chapter described by arc42 is ”a list of identified technical risks or technical debts, ordered by priority”. The list can also include suggestions for measures to minimize, mitigate or avoid risks or reduce technical debts.[100]

Here we listed the two known short comings of our solution: single unit of mitigation and limited Quality of Service (QoS) support. Both topics are discussed in details in chapter 4.

[2] Current number of events processed per second on the CARU Smart Sensor
[3] Estimated number of events processed in the cloud by the end of the year
[4] Expected number of events processed in the cloud per second for 1000 devices
[5] Maximal size the CloudEvents specification allows [99]

3.2. Prototyping

While creating the architecture documentation, we often tried out different aspects of the architecture and the programming language by creating small prototypes.

We created eleven working prototypes and multiple non-compiling or non-runnable prototypes. Some of them are very small like the hello world[6], the Rust documentation generation[7] or the deserialize of JSON with Rust[8] prototype. Others, like the Microkernel[9], package structure[10] or multi-threading[11] prototype, implement almost the full functionality, but with radically different approaches.

The creation of the prototypes was crucial for our improved understanding of the programming language and the ability to take advantage of its unique features.

3.3. Pair-Programming

Pair-programming is a style of programming where two programmers working collaboratively on the same code, design or test at the same time. The desired goal behind this strategy is to have better code quality and better know-how distribution.[101, 102]

After we created the design of our solution, we started its implementation. To take maximal advantage of the new knowledge each of us gained while prototyping, we decided to do pair-programming.

The implementation of the CloudEvents Router's core was accomplished by pair-programming. One was the pilot at the keyboard, while the other kept a high-level overview and track of the task at hand. The roles were switched after each finished task.

After we created the core of the CloudEvents Router, the remaining tasks were divided and worked on individually. These tasks were mostly simple implementation tasks without architectural relevance. To keep a common understanding of the code base, each of the crafted changes were reviewed by the other team member.

[6] https://github.com/ce-rust/architecture-prototype/tree/master/hello-world
[7] https://github.com/ce-rust/architecture-prototype/tree/master/code-doc
[8] https://github.com/ce-rust/architecture-prototype/tree/master/deserialize-json
[9] https://github.com/ce-rust/architecture-prototype/tree/master/microkernel-adapters
[10] https://github.com/ce-rust/architecture-prototype/tree/master/multi-creates
[11] https://github.com/ce-rust/architecture-prototype/tree/master/multithreading-mikrokernel-adapters-arc
https://github.com/ce-rust/architecture-prototype/tree/master/multithreading-mikrokernel-adapters-clean

4. Results and Discussion

This section goes through all the functional and non-functional requirements defined at the beginning of the project. We explain if and how we fulfilled them. Additionally, we discuss alternative approaches we would consider if we were to reencounter the same challenges.

All the functional requirements can be found in Table 3.1 and all the non-functional requirements can be found in Table 3.2. The landing zones for the non-functional requirements can be found in Table 3.3. In the beginning of each section, the full requirement description is repeated.

4.1. FR-1: CloudEvents Support

FR-1: *"The router must understand the CloudEvents specification version 1.0."*

The current implementation of the CloudEvents Router supports the CloudEvents specification version "0.2" and "1.0".

Initially we planned to use the existing "cloudevents" Rust library/crate. But unluckily, the library only implements the CloudEvents specification in version "0.2".[1]

We forked the code from the author of the library and added support for version "1.0" of the specification ourselves. It was challenging to change existing code, which only supports a single version of the specification, to allow for support of multiple versions of the specification. The main difficulty was to introduce a version independent CloudEvents abstraction while honoring the design decisions of the original author. In the end, we touched almost every line of his implementation in order to add support for both versions of the specification.[2]

Our goal was to contribute our changes back to the repository of the original author. Unfortunately, he did not react to our pull request until the finalization of this document was achieved.

This part of the project did show us how incredibly difficult it is to design a solid API for a library that can be used in a variety of projects. If we had to do this part again, we probably would not try to create a library for general use and only create an internal representation of CloudEvents for the use in our project. The creation of a decent CloudEvents library in Rust could be a topic for a student research project in itself.

4.2. FR-2: Modern Linux Support

FR-2: *"The router must be able to run on a modern Linux distribution."*

The current implementation of the CloudEvents Router supports modern Linux distributions. Our CI pipeline makes sure all our tests run successfully on Ubuntu "18.04", Arch "2019.12.05" and Debian Linux.[3] In general, the support of Rust for modern Linux distributions is quite good.[103]

[1] https://docs.rs/crate/cloudevents/0.1.1

[2] https://github.com/ce-rust/rust-cloudevents/tree/feat/cloudevents-v1-refactored

[3] Ubuntu and Arch Linux are tested with custom Docker images. Debian Linux is tested with the official Rust Docker image.

4.3. FR-3: Routing Rules Design

FR-3: *"A simple way to define routing rules based on the CloudEvents attributes must be designed."*

We currently have an implementation for a rule based routing component.[4] The rule based router component accepts rules described in a simple, JSON-based language. The language was inspired by the Amazon SNS Subscription Filter Policies.[104]

The routing rule language consists of two logic operation statements and four matching expressions. The logic keywords are `And` and `Or`. The matching keywords are:

- `Contains` for simple matches
- `Exact` for exact matches
- `StartsWith` for matches at the beginning
- `EndsWith` for matches at the end

Any matching and logic operation keywords can be combined with `And` or `Or` to build a tree.

Listing 1 shows an example routing rule which defines that all CloudEvents whose type field is `"events.cloudevents.example"` and source field starts with `"service::prefix"` should be forwarded to port `"output-port"`.

```
{
  "output-port": {
    "And": [
      {
        "Exact": ["Type", "events.cloudevents.example"],
        "StartsWith": ["Source", "service::prefix"]
      }
    ]
  }
}
```

Listing 1: Routing Rule Example

In addition to the rule based router component we have an implementation of a broadcast router.[5] The broadcast router forwards all incoming events to a list output ports without taking the content of the CloudEvents into account.

4.4. FR-4: UNIX Socket Support

FR-4: *"The router should offer a UNIX socket interface to send and receive CloudEvents."*

We currently have implementations for UNIX socket input and output port components.[6] Both implementations act as UNIX socket server. After a client has connected to the socket, the input port waits for the client to send CloudEvents serialized as JSON. The output port serializes the outgoing CloudEvents as JSON and writes them to the open socket. The newline character is used to separate events. When no client is connected to the output socket, back-pressure builds up until no new event can be accepted on any input port.

[4] https://github.com/ce-rust/cerk/tree/master/cerk_router_rule_based
[5] https://github.com/ce-rust/cerk/tree/master/cerk_router_broadcast
[6] https://github.com/ce-rust/cerk/tree/master/cerk_port_unix_socket

There is no implementation for a client mode. This means we cannot connect to existing UNIX sockets that were created by a different process. We prioritized the server mode because of the use cases described in the architecture documentation.

Because UNIX sockets are operating system resources, there is a chance of unprovoked errors. The current error handling is, with one exception, quite radical. The UNIX socket port will panic and shut down the whole CloudEvents Router. Only the case of a lost client connection is handled less radically. In that case, the UNIX socket port will wait for a new client to connect to the socket.

In theory, a UNIX socket server can accept multiple clients that connect to the same socket. We decided that our current implementation will only support one client at a time to keep the complexity at a minimum.

4.5. FR-5: MQTT Support

FR-5: *"The router should offer a MQTT version 3.1.1 interface to send and receive CloudEvents."*

We currently have an implementation for a MQTT input/output port component.[7] We use the Eclipse Paho MQTT Rust Client Library[8] to connect to the MQTT broker.[105]

The Eclipse Paho MQTT Rust Client Library is a Rust wrapper around the Eclipse Paho MQTT C Client Library. We decided to use the Paho library because it was by far the most mature MQTT client available in Rust. The downside of using a C wrapper library is that we encountered some issues when we tried to cross-compile our CloudEvents Router for the ARM architecture. The wrapper had no issues cross-compiling the Paho C library but failed to cross-compile OpenSSL, which is a dependency of the Paho C library. Fortunately, the Paho Rust library offers a feature flag that allowed the deactivation of the encryption of the transport layer. This means that currently, our implementation does not support connecting to a broker over an encrypted connection when building it for the ARM architecture.

MQTT has the concept of QoS. There are three levels of QoS: "at most once delivery" (0), "at least once delivery" (1) and "exactly once delivery" (2). Currently, the CloudEvents Router only supports QoS 0 because input and output ports are loosely coupled. To support QoS 1 and 2 an acknowledgment mechanism would need to be designed and implemented.[106]

4.6. FR-6: Debugging

FR-6: *"The router should provide a simple interface to access debugging information."*

The current implementation of the CloudEvents Router produces structured logs for debugging. We use the `log`[9] Rust library/crate to produce log messages. The `log` library only provides the logging API and needs an actual logging implementation.

In our examples we used the `env_logger`[10] logging implementation. This library allows the user to change the logging level after the binary was compiled by setting an environment variable. `RUST_LOG=info`, for example, sets the logging level to "info".

This would definitely be an area worth investing more resources in in subsequent work. The integration with CNCF monitoring projects like OpenTracing or Prometheus would be very beneficiary. OpenTracing is vendor-neutral instrumentation library for distributed tracing and Prometheus is a monitoring solution.[107, 108]

[7] https://github.com/ce-rust/cerk/tree/master/cerk_port_mqtt
[8] https://docs.rs/crate/paho-mqtt
[9] https://docs.rs/crate/log/0.4
[10] https://docs.rs/crate/env_logger/0.7

4.7. NFR-1: Modularity

NFR-1: *"It should be easy to implement additional ports and configuration loaders."*

We currently have implementations for two debug and two protocol ports.[11] All the ports are implemented as separate packages and only use the public API of the Microkernel. They are well documented and serve as examples for developers who want to implement their own ports.

We currently do not provide any configuration loader as a library. But we provide multiple example projects that implement their own, static configuration loaders.[12]

Listing 2 shows the static configuration loader of the "hello world" example project. The configuration loader waits for the `BrokerEvent::Init` event and then sends the configurations for the `"router"`, `DUMMY_SEQUENCE_GENERATOR` and `DUMMY_LOGGER_OUTPUT` component to the kernel for delivery.

```
fn static_config_loader_start(
  id: InternalServerId,
  inbox: BoxedReceiver,
  sender_to_kernel: BoxedSender,
) {
  info!("start static config loader with id {}", id);
  loop {
    match inbox.receive() {
      BrokerEvent::Init => {
        sender_to_kernel.send(BrokerEvent::ConfigUpdated(
          Config::Vec(vec![Config::String(String::from(DUMMY_LOGGER_OUTPUT))]),
          String::from("router"),
        ));
        sender_to_kernel.send(BrokerEvent::ConfigUpdated(
          Config::Null,
          String::from(DUMMY_SEQUENCE_GENERATOR),
        ));
        sender_to_kernel.send(BrokerEvent::ConfigUpdated(
          Config::Null,
          String::from(DUMMY_LOGGER_OUTPUT),
        ));
      }
      broker_event => warn!("event {} not implemented", broker_event),
    }
  }
}
```

Listing 2: The static config loader of the "hello world" example

4.8. NFR-2 and LZ-1: Cross Platform

NFR-2: *"It should be easy to deploy the router to different environments especially to the cloud and embedded Linux platforms."*

LZ-1: The landing zone is to have a minimal result with support for Ubuntu Linux, a targeted result with support of an Embedded Linux and an outstanding result with support of FreeRTOS.

[11] https://github.com/ce-rust/cerk#ports
[12] https://github.com/ce-rust/cerk/tree/master/examples/src

The current implementation of the CloudEvents Router runs on the "x86_64" and "ARMv7" processor architecture. "ARMv7" based processors are often built into embedded devices and run Linux. It may run on more platforms that are supported by the Rust toolchain.[103]

We used the Microkernel Pattern described by Buschmann, Meunier, Rohnert, Sommerlad and Stal in their book Pattern-Oriented Software Architecture as the basis for our cross platform efforts.

"The Microkernel architectural pattern applies to software systems that must be able to adapt to changing system requirements. It seperates a minimal functional core from extended functionality and customer-specific parts. The Microkernel also serves as a socket for plugging in these extensions and coordinating their collaboration."[93]

The Microkernel pattern defines five kinds of participating components: internal servers, external servers, adapters, clients and the Microkernel itself. It is the Microkernels task to provide the core mechanisms, offer communication facilities and manage the resources. Internal servers implement additional services and encapsulate some system specifics. External servers provide programming interfaces to its clients. Both, internal and external servers, collaborate with the Microkernel component. In addition to these internal components, the pattern also defines clients and adapters. Clients are separate applications that consume services from the Microkernel. An adapter is code that is part of the client, but is provided by the creators of the Microkernel. Nowadays, adapters are often called Software Development Kits (SKD). SDKs help the developers to interact with external server's APIs by adding an abstraction layer to it.[93]

The only significant difference between our interpretation of the Microkernel Pattern and the description from the authors is that we merged the internal and external servers. We call both of them internal servers because, in our context, we did not see any differences in how we interact with them. Treating them the same but naming them differently seemed unnecessary to us. We also do not provide any adapters/Software Development Kit (SDK)s at the moment. Figure 4.1 shows the building blocks of the Microkernel with their responsibilities.

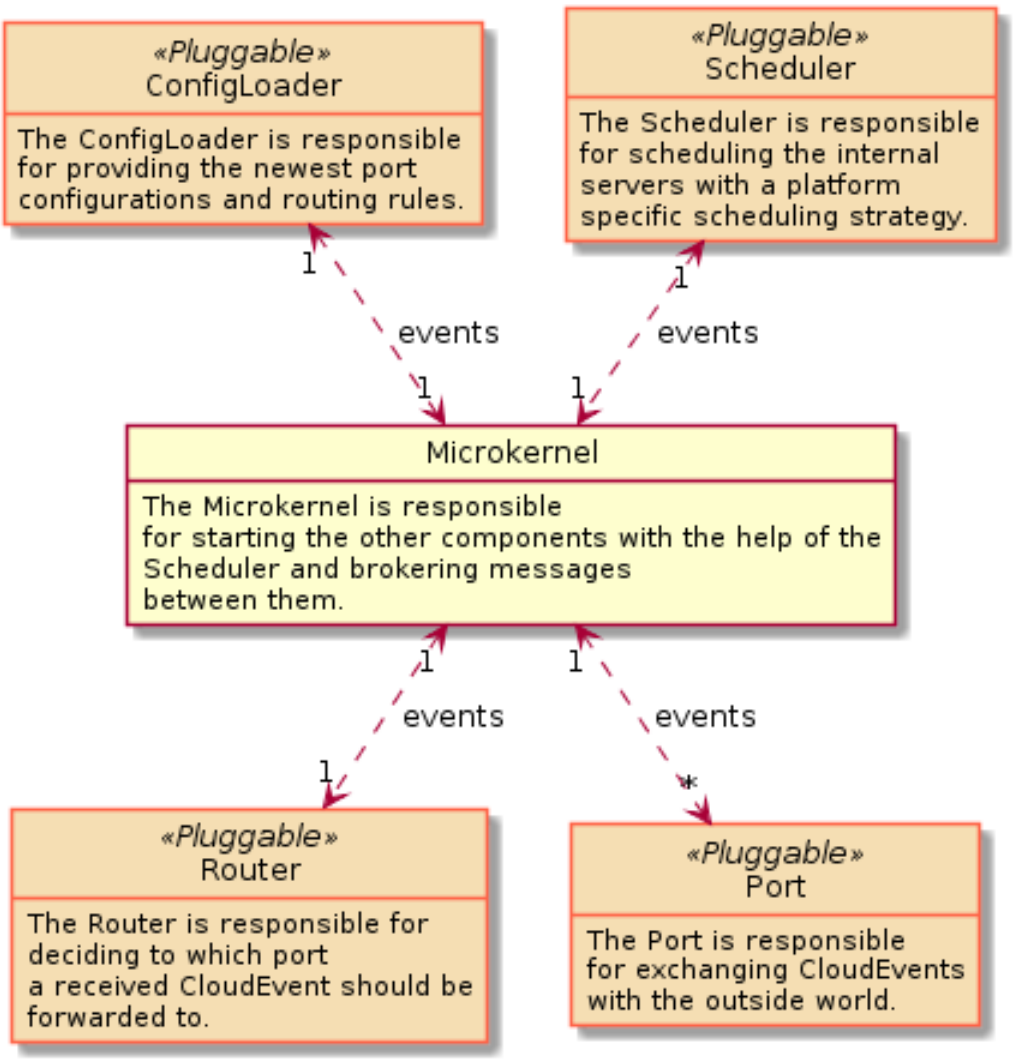

Figure 4.1.: Building Blocks of the Microkernel with their Responsibilities

Each of the pluggable components, except the Scheduler, is a function that has a `fn(id: InternalServerId, inbox: Receiver, sender_to_kernel: Sender);` signature. `id` is the unique identifier assigned to the internal server and is used to refer to each other.

Rust has the concept of channels for sending data between threads. Channels consist of two parts: the `Sender` and the `Receiver`. The ownership of each part is given to different threads and all the data sent with the `Sender` can be received by the `Receiver`.[109]

We took this concept and defined an interface based on it. This interface must then be implemented by Scheduler for the specific platform it targets. Each internal server receives a Receiver (`inbox`) and a Sender (`sender_to_kernel`). The `inbox` is used to receive messages from the Microkernel and the `sender_to_kernel` is used to send messages to the Microkernel.

The outstanding result for the cross platform landing zone is the support for FreeRTOS. During the prototyping phase we did explored the integration with FreeRTOS by utilizing the "freertos_rs"[13] Rust library. We decided, together with CARU, that Rust's support for bare-metal environments, and FreeRTOS specifically, is not mature enough to be used for a Scheduler implementation. Our Microkernel design should allow the easy implementation of a FreeRTOS Scheduler as soon as the Rust tooling for FreeRTOS is stable enough.

[13] https://docs.rs/freertos_rs/0.3.0

4.9. NFR-3 and LZ-2: Open Source

NFR-3: *"Easy findable and followable documentation to setup the project and run it."*

LZ-2: The landing zone regarding documentation defines that the minimal result is achieved when there is a simple README file. The targeted result is to have extensive documentation. To have many detailed tutorials would be an outstanding result.

Our implementation of the CloudEvents Router is open-sourced under the Apache 2 license. It can be found on GitHub: https://github.com/ce-rust/cerk

We created an README file which provides an overview over the project and instructions to get started with it.[14]

In addition to the README we used the fantastic documentation tooling of Rust and annotated our code. With a simple command a beautiful documentation website gets created from the code annotations.[110]

There are five example projects with different CloudEvents Router configurations.[15] Each of them has its own README with instructions to run it.

4.10. LZ-3 and LZ-4: Performance Tests

LZ-3: The landing zone defines the targeted throughput of the router. A minimum of 5 events per second is required, 500 per second is the goal and 1500 per second is outstanding.

LZ-4: The landing zone regarding message size defines only two goals: A minimum of 1 KB per message and a target of 64 KB. 64 KB is the maximal size allowed by the CloudEvents specification.[1]

At the end of the construction phase we tested the performance of our router and compared it to Apache Camel, the winner of the market analysis. To conduct the performance tests, we set up a project containing multiple Docker containers.

Our setup consists of an MQTT broker, our CloudEvents Router implementation and a simple router based on Apache Camel.

A Script produces, with help of the Eclipse Mosquitto command-line tool, a high load of CloudEvents and sends them to a topic on the MQTT broker. One of the routers subscribes to the topic, routes the events and sends them back to a different topic. The script measures how long it took to receive all sent messages. This test is performed for each implementation in isolation.

We overachieved the outstanding throughput with over 2500 events per second. We also outperformed Apache Camel, which achieved around 1700 events per second. In a real scenario, Apache Camel would probably have a slightly worse throughput because the test compared our fully featured router with a minimal implementation of router with Apache Camel. Apache Camel was only receiving events from a MQTT broker and sent it back over a different topic without taking their content into account. Our router also deserialized and serialized the events them from and to JSON.

We also repeated the same test while restricting the resource available to the routers. There, our router demonstrated that he still performs reasonably, with only 20 MB of memory, which is not even enough to start the Apache Camel router.

We conducted only a minimal performance test with quite simple scenarios. We can imagine many other scenarios worth testing. For example how the routers perform while applying complex routing rules or how the performance characteristics from different port types are.

[14] https://github.com/ce-rust/cerk/blob/master/README.md
[15] https://github.com/ce-rust/cerk/tree/master/examples/src

The detailed performance report is attached in Appendix K and the source code for the performance test can be found in a separate repository.[16]

[16] https://github.com/ce-rust/performance-test

5. Conclusion

In the conclusion we summarize the work of this thesis and evaluate our results.

After that, we will have an outlook on topics worth exploring in future work.

5.1. Learnings

The main goals of this thesis are the creation of the CloudEvents Router and the report about our experience with Rust.

5.1.1. Market Analysis

The market analysis was broad at the beginning and went into depth with a second comparison of the best fitting procucts for our partner, CARU.

The first comparison showed, that the CloudEvents standard is still rather novel and that none of the products that provide native CloudEvents support have reached the second round of our comparison.

The second comparison was much more detailed. It includes a walk through the setup and the first tutorial of each product. Sadly, Knative Eventing was a let down to us. The product that has many big names like Google, IBM and Red Hat behind it. It was not easy to install and is, according to the tutorials, not as loosely coupled and cross-platform-available as officially described. However, the other two products that came on top at the in-depth comparison, Apache Camel and Node-RED, both made a solid impression. It was a tough and tight race between the two and the decision to choose Apache Camel over Node-RED was not an easy one.

Nerveless, after the market analysis we decided to still implement our own event router in Rust. None of the products had a perfect fit for the use case of CARU.

5.1.2. The CloudEvents Router

Prior to the implementation of the router, a lot of thought was put into the architecture design. This was as important for us as for the customer.

To create this architecturally sound solution, we iterated between creating comprehensive architecture documentation and prototyping different aspects of the problem.

The chosen architecture is designed to allow platform-independence: the cloud, embedded Linux and also FreeRTOS, a real-time operating system, were our targets. In order to accomplish this, we decided to use the Microkernel pattern as our architectural base. With this approach multiple run-times could be implemented for the same product.

The design also allowed us to have well-separated ports for different interfaces which could be used to send and receive the CloudEvents. The ports could also be implemented later through third parties.

While the concept of the Microkernel did not change during the design and prototype phase, the coding style and the interaction between the components did a lot.

The core components were all implemented while pair-programming, to get out the most of our prototype know-how and to provide the best possible code quality.

During the implementation of additional ports for our router, the kernel design proved to be sound. Adding additional ports was possible without editing the other components. However, one slight detail lead to some discussions: Does the router need to be a separate component?

At the end we open-sourced our solution under the product name CERK. The acronym stands for CloudEvents Router with a Microkernel.

Next to the product itself, we also extended an already existing CloudEvents library for Rust, on which our Rust router depends on.

5.1.3. The Rust Programming Language

Using Rust for the first time was challenging at first. After a frustrating start, we began enjoying using it. Rust brings all the advantages of resource efficient languages plus some unique safety features. However, the latter makes it harder to get started with Rust. Nevertheless, these new concepts are also the reason for the steep learning curve. Fortunately, the exceptionally helpful error messages provided by the compiler make it easy to find one's mistakes. Finally, the tools provided by the Rust Team is thoughtfully integrated with each other. The creators of Rust brought together the best practices from many other programming languages and made them the default.

We could really only advice CARU to try out Rust them self and consider using it for new components on the device.

5.2. Next Steps

The CloudEvents Router is still a prove of concept and is in the current state not production ready.

At the moment, no advanced error mitigation concept is implemented. The entire router is one "Units of Mitigation"; when there is an unhandled error in any of the components, the whole router crashes.

We recommend the implementation of units of mitigation per adapter as described in the pattern "Units of Mitigation" in the book Patterns for Fault Tolerant Software by Robert Hanmer [111]. With this concept a single erroneous port could be stopped and consequently restarted, without the need of restarting the entire router.

Another reliability goal that is still missing is a QoS concept. A method to define QoS option like the MQTT protocol does with the requirement "at least one delivery" would make sens for the router.[106] This could be implemented stateless with the ability that an input port would wait with a acknowledgment of a message until the output port would send the message successful.

Additionally more ports, blog posts and tutorials on how to deploy the router would be great to create an open-source community around the product.

List of Figures

List of Tables

List of Listings

Glossary

There are some overlapping glossary entries between the documents of this thesis. Each document contains only the glossary entries which are referred in that document.

"ownership model" The "ownership model" is the most unique feature of Rust. In Rust memory is allocated and freed explicit, but in contrast to C or other low level programming languages without the need of an garbage collector, Rust enforces a set of rules on this actions, to ensure memory safety."[112]. 54, see Rust

LaTeX "LaTeX, which is pronounced «Lah-tech» or «Lay-tech» (to rhyme with «blech» or «Bertolt Brecht»), is a document preparation system for high-quality typesetting. It is most often used for medium-to-large technical or scientific documents but it can be used for almost any form of publishing."[113]
LaTeXis distributed under the LaTeX project public license. It is a free software license.[114]. 31, 51, 141, 142, 179

Academic Free License Academic Free License is a open-source license provided by the Open Source Initiative (OSI). It is not a copyleft license.[115]. 19, 54, see copyleft, OSI & open-source

AgegTech "AgegTech is about digitally-enabling the Longevity economy". The products could be divided into "4 categories of digital-enablement in AgeTech: services purchased by older people; services purchased on behalf of older people; services traded between older and younger people; and services delivered to future older people."[116]. 13

AMQP "AMQP, the Advanced Message Queuing Protocol, is an open standard for interoperable messaging at the wire protocol level. Message brokers that implement AMQP can communicate with each other and exchange messages without the need for adapters or bridges. An AMQP message broker can provide first-class native language bindings for multiple programming languages; so AMQP-based messaging is a good choice for cross-platform compatibility across the Enterprise. "[117]
It is standardized by OASIS.[118]. 57

Apache 2 Apache 2 is an open-source license from the Apache Software Foundation. The license allow modifications and redistribution of the software. It has no copyleft.[119]. 17–19, 29, 41, 54, see copyleft & open-source

API Is a interface to communicate from one system to an other[120]. 57

arc42 arc42 is a template for documentation and communication of a system or software. It focus on lean and agile development approaches.[84] The template is available in available in different format, one of them is LaTeX.[121] It is open-source licensed under the Creative Commons Attribution-ShareAlike 4.0 International License.[122]. 28–32, see LaTeX

AWS Amazon Web Services (AWS) is a cloud platform provided by Amazon[123]. 57

Buildkite Buildkite is a service for coordinating buildpipelines which get executed on own infrastructure. They provide a build agent that can easily be deployed to AWS and other platforms. Buildkite is well integrated with GitHub.[124]. 142, 143, see AWS, CI & GitHub

bytecode "Bytecode is program code that has been compiled from source code into low-level code designed for a software interpreter. It may be executed by a virtual machine (such as a JVM) or further compiled into machine code, which is recognized by the processor."[125]. 24, see Java & JVM

C C is a programming language developed by Dennis Ritchie. C++ is a superset of C. Later it was standardized by AMSI and ISO.[126]. 37, 51–53, 55, see C++

C++ C++ is a general-purpose programming language designed by Bjarne Stroustrup. Later it was standardized by AMSI and ISO.[126]. 51

C++ Bindings C++ Bindings are wrappers for a programming language to use C++ function in it.[127]. see C++

Cargo Cargo is the default package manager for Rust. It is really simple, compiles code, installs packages, runs tests, formats source files and generates the documentation. Cargo can be extended with additional subcommands by installing packages via Cargo itself.. 54, 142, see Rust

CARU Smart Sensor The CARU Smart Sensor is a product placed in the living environment of an elderly person to collect various room parameters, learn the behaviors of the inhabitant and alarm relatives or caretakers in case of deviations. Its user experience is optimized for elderly people.[2]
The device has two central processing units (CPUs) with two different operating systems on it.. 13, 32, see FreeRTOS & Linux

CC0 "No Rights Reserved" License[128]. 31

CI "Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early."[129]. 57

Cloud Native "Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach."[130]. 52, see CNCF

CloudEvents The CloudEvents standard is a standard from the CNCF. The goal of the standard is to "describe event data in common formats to provide interoperability across services, platforms and systems."[1] The standard reached version 1.0 on the 24th of October 2019.. 9, 10, 13–15, 17–23, 25, 26, 28, 32, 33, 35–37, 39, 41, 43, 44, 52, 53, 181, 182, see CNCF

CNCF The Cloud Native Computing Foundation is part of the Linux Foundation. The goal of the foundation is to empower the usage of the cloud by providing standards and open-source projects, the use the term Cloud Native.[130] The foundation host project like Kubernetes[131] and Prometheus[132]. Apple, Microsoft, Google and Amazone Web Services are only a couple of the most famous partners.[133]. 57, see Cloud Native

copyleft Copyleft is a method for making a program including all versions and modifications free.[134]. 22, 23, 51–53

Docker "Docker is an open-source project for automating the deployment of applications as portable, self-sufficient containers that can run on the cloud or on-premises."[135]. 21, 23, 25, 35, 41, 157, 161, 182

DSL A custom language for a specific purpose. "It provides ways to work with exactly those abstractions you need in the respective context."[136]. 57

Eclipse Mosquitto Eclipse Mosquitto is an open-source MQTT message broker with an additional command line tool to publish and subscribe from and to MQTT queues.[137]. 41, 181

EKS Amazon Elastic Kubernetes Service is a managed Kuberentes cluster as a service provided by Amazon.[138]. 57, see Kubernetes

FaaS "Function as a service (FaaS) is a cloud computing model that enables users to develop applications and deploy functionalities without maintaining a server, increasing process efficiency. The concept behind FaaS is serverless computing and architecture, meaning the developer does not have to take server operations into consideration, as they are hosted externally. This is typically utilized when creating microservices such as web applications, data processors, chatbots and IT automation."[139]. 57

FreeRTOS FreeRTOS is a Real Time Operating System (RTOS) written in C distrubuted under the MIT license. The kernel binary is around 6KB to 12KB.[140]. 21, 32, 38, 40, 43, see RTOS

Git Git is an open-source distributed version control system. The tool was created in 2005 by the Linux community.[141]. 26, 52, 141, 142

GitHub GitHub is a source code management and collaboration platform based on Git. It is a well known platform for open-source code and was acquired in 2018 by Microsoft.[142]. 10, 29, 41, 51, 55, 141–143, 182, see Git

GNU 2 The GNU 2 or GNU General Public License v2 is is a copyleft license of the Free Software Foundation, the latest version is version 3.[143]. 18, 19, 53, 54, see GNU 3, copyleft & open-source

GNU 3 The GNU 3 or GNU General Public License v3 is a copyleft license of the Free Software Foundation, the latest version is version 3.[144]. 19, 52, 54, see GNU 2, copyleft & open-source

Go Go is a statically typed general purpose programming language.[145] The language was developed by Google an is now open-sourced under the BSD-style license.[146]. 25

HTTP "The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers"[147]. 57

Industry 4.0 "The term "Industry 4.0" refers to the transformation of the manufacturing sector by digital technologies, such as the internet of things (IoT), artificial intelligence (AI), machine learning, robotics, 3-D printing, visualization, virtual/augmented reality and analytics."[148]. 54, see IoT

IoT "The internet of Things, or "IoT" for short, is about extending the power of the internet beyond computers and smartphones to a whole range of other things, processes and environments. Those "connected" things are used to gather information, send information back, or both."[149]. 53, 57

Java "Java is a high-level programming language developed by Sun Microsystems."[150] Java code is compiled to java bytecode and executed with a JVM. 18, 24, 26, 54, 57, 142, 182, see JVM & bytecode

JavaScript "JavaScript is a programming language commonly used in web development. It was originally developed by Netscape as a means to add dynamic and interactive elements to websites."[151] It is a scripting language based on the ECMAScript standard.[152]. 24, 53, 57

JDK The Java Develompen Kit is a combination of the Java runtime, the compile and other tooling that is needed for the development of Java applications.[153]. 57

JSON "JSON is a syntax for serializing objects, arrays, numbers, strings, booleans, and null. It is based upon JavaScript syntax but is distinct from it: some JavaScript is not JSON."[154]. 57

JVM The Java Virtual Machine is a virtual machine running a java program. It implements a concrete commands for the abstract java specification. It executes the java bytecode and manages the memory.[155]. 57, see Java

Kafka Apache Kafka is a stream processing software and a message broker. Kafka provides queuing and publish-subscribe.[156]. 17

Knative Knative is a FaaS platform on top of Kubernetes.. 17, see Kubernetes & FaaS

Knative Eventing Knative Eventing is a set of loosely coupled services that route CloudEvents from producers to interested consumers. It is tightly integrated into Kubernetes.[6]. 17, 22, 25, 161, see Knative & Kubernetes

Kubernetes Kubernetes is an open-source container orchestration platform hosted by CNCF.[157]. 17, 25, 52, 53

Linux Linux is a operating system written in C with a microkernel architecture and licenced under the GNU 2 license.[126, 158, 159]. 9, 19, 21, 28, 32, 35, 39, 43, 52, 55

Markdown "Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convert it to structurally valid XHTML (or HTML)."[160]. 31

Maven Maven is a software project management tool. It could download and install packages and help with the compilation.[161]. 24, 151, 167, 182

Microkernel "The Microkernel architectural pattern applies to software systems that must be able to adapt to changing system requirements. It seperates a minimal functional core from extended functionality and customer-specific parts. The Microkernel also serves as a socket for plugging in these extensions and coordinating their collaboration."[93]. 29–31, 33, 38–40, 43, 44, 145

MIT license The MIT License is a open-source license originally created by the Massachusetts Institute of Technology and documented by the OSI.[162]. 52, 54, see OSI & open-source

MQTT MQTT is a lightweight publish/subscribe messaging transport protocol for machine to machine communication.[106, 163]. 57

MVP A product is at the state of a Minimum Viable Product when it gives the user an additional value, but as little extra feature as possible to get a good feedback loop for additional features[164]. 57

Node.js Node.js is an open-source JavaScript runtime build on top of the Google Chrome V8 JavaScript engine.[165]. 19, 24, 169, see JavaScript

Open Unified Process "an agile and Unified Process that contains the minimal set of practices that help teams to be more effective in developing software. OpenUP embraces a pragmatic, agile philosophy that focuses on the collaborative nature of software development. It is a tools-agnostic, low-ceremony process that can be used as is or extended to address a broad variety of project types."[83]. 27

open-source "Generally, Open Source software is software that can be freely accessed, used, changed, and shared (in modified or unmodified form) by anyone. Open source software is made by many people, and distributed under licenses that comply with the Open Source Definition.
The internationally recognized Open Source Definition provides ten criteria that must be met for any software license, and the software distributed under that license, to be labeled "Open Source software.""[166]
GNU 2, GNU 3, Apache 2, Academic Free License and MIT license are all open-source licenses which have been approved by OSI. 19, 22, 28, 44, 51–53, 141, see OSI

OSI OSI is a California public non-profit corporation with the goal of educate about open source.[167]. 58, see open-source

pair-programming Pair-programming is a style of programming where two programmers working collaboratively on the same code, design or test at the same time. The desired goal behind this strategy is to have better code quality and better know-how distribution.[101, 102]. 29, 33, 44

PlantUML PlantUML is a tool to generate different Unified Modeling Language (UML)) and non UML diagrams. The diagrams are generated by writing with the PlantUML Language. The tool is written in Java and has various output formats, one of the im JPGE.[168]. 142, see UML

PRODISYS PRODISYS is a Project sponsored by the German Federal Ministry of Education and Research[28] to make a pilot for a better planning and coordination in the industry with new features of the Industry 4.0[28]. 19

pull request "Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch."[169]. 141, 142, see Git & GitHub

Python Python is a high-level general purpose programming language; It is used as scripting language, for Web application and many other things[170]. 14, 18

QoS Quality of Service is defined in the term of telecommunication as "Totality of characteristics of a telecommunications service that bear on its ability to satisfy stated and implied needs of the user of the service."[171]. 58

RTOS Real Time Operating Systems are a type of operating systems where "The scheduler ... is designed to provide a predictable (normally described as deterministic) execution pattern. This is particularly of interest to embedded systems as embedded systems often have real time requirements. A real time requirements is one that specifies that the embedded system must respond to a certain event within a strictly defined time (the deadline). A guarantee to meet real time requirements can only be made if the behaviour of the operating system's scheduler can be predicted (and is therefore deterministic)."[172]. 58

Rust Rust is a low-level programming language with guaranteed thread and memory safety. It achieves that with its unique "ownership model". The goal is to build safe and reliable software.[112, 173]
Rust is used by Mozilla Firefox, Dropbox and many others.[174, 175]
In 2019, Rust has also been elected in the well-known survey of Stack Overflow for being the "most loved" programming language.[176]. 10, 14, 15, 26, 29, 31, 33, 35, 37, 39–41, 43, 44, 51, 52, 54, 142, 143, 145, 182, see "ownership model" & Stack Overflow

rustc rustc is the Rust compiler provided by the Rust Team. It can be used directly (`rustc`) or with Cargo (`cargo build`).[177]. 142, see Rust & Cargo

rustup rustup is a tool to install different versions of Rust, like stable, beta and nightly, and for different compilation targets.
It is the recommended way for developers to install Rust.[178]. 10, 142, see Rust & Cargo

SaaS Software as a service (SaaS) is a software distribution model in which a third-party provider hosts applications and makes them available to customers over the Internet.[179]. 58

SDK Software Development Kits are libraries which ease the access to APIs. 58

Serial Interface "A serial interface is a communication interface between two digital systems that transmits data as a series of voltage pulses down a wire."[180]. 13

Serverless Framework The Serverless Framework is a FaaS abstraction layer.. 18, 54, see FaaS

Siemens MindSphere "MindSphere is the cloud-based, open IoT operating system from Siemens that connects your products, plants, systems, and machines."[181]. 19

SNS Amazon Simple Notification Service (SNS) is a fully managed publish/subscribe messaging system. It allows the filtering based on meta-data sent with the messages.[17]. 58

SQS Amazon Simple Queue Service (SQS) is a fully managed message queuing service from Amazon with first-in, first-out (FIFO) support[182]. 58

Stack Overflow Stack Overflow is primary a Q&A forum for coding related problems.. 22, 54

STOMP A Simple Text Oriented Message Protocol, designed for asynchronous message passing[183]. 58

T-Metric T-Metric is a time tracking app with a functionally limited free plan for up to 5 people. It has integrations for various products, such as GitHub.[184]. 10, 141

TCP The Transmission Control Protocol (TCP) is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks.[185]. 58

UML UML is a standard for diagramming notation. It was created by Booch and Rumbaugh in 1994.[186]. 54, 58, 142

UNIX UNIX a general-purpose, mutli-user operation system. It is written in C and is the root of the Linux operating system.[126, 187]. 19

UNIX socket "The `AF_UNIX` socket family is used to communicate between processes on the same machine efficiently."[188] . 13, 20, 23, 28, 36, 37, 55, see UNIX

WAMP "WAMP is a routed protocol that provides two messaging patterns: Publish & Subscribe and routed Remote Procedure Calls. It is intended to connect application components in distributed applications. WAMP uses WebSocket as its default transport, but can be transmitted via any other protocol that allows for ordered, reliable, bi-directional, and message-oriented communications."[189]. 58

WebSocket "The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code. The security model used for this is the origin-based security model commonly used by web browsers. The protocol consists of an opening handshake followed by basic message framing, layered over TCP. The goal of this technology is to provide a mechanism for browser-based applications that need two-way communication with servers that does not rely on opening multiple HTTP connections"[190]. 19

XML XML is a syntax for documents. "XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure."[191]. 58

Acronyms

ADR ARD (Architecture Decision Record). 31, Glossary: ADR

AMQP Advanced Message Queuing Protocol[118]. 19, Glossary: AMQP

AMSI American National Standards Institute. 51

API Application Programming Interface. 19, 35, 37–39, Glossary: API

AWS Amazon Web Services[123]. 13, 17, 18, 21–23, 51, 143, Glossary: AWS

CI Continuous Integration. 10, 35, 142, 143, Glossary: CI

CNCF . 13, 37, 52, 53, Glossary: CNCF

CPU central processing unit. 52, 182

CRD Custom Resource Definitions [192]. 162

DSL Domain Specific Language. Glossary: DSL

EKS Elasitc Kubernetes Service . 22, Glossary: EKS

FaaS Function as a Service. 17, 18, 53, 54, Glossary: FaaS

FR Functional Requirements. 28

GB Megabytes, equal to 1000 Megabytes. 182

HTTP HyperText Transfer Protocol. 17, 19, 20, 22, Glossary: HTTP

IDE Integrated Development Environment. 10, 143

IoT Internet of Things. 13, 19, Glossary: IoT

ISO International Organization for Standardization. 51

JDK Java Development Kit. 24, 151, 167, Glossary: JDK

JSON JavaScript Object Notation. 24, 26, 33, 36, 41, Glossary: JSON

JVM Java Virtual Machine. 24, 26, 51, 53, Glossary: JVM

KB Kilobytes, equal to 1000 Bytes. 41, 52, 182

MADR Markdown Architectural Decision Records[96]. 31

MB Megabytes, equal to 1000 Kilobytes. 41, 182

MPL Mozilla Public License. 19, 23

MQTT Message Queuing Telemetry Transport[106]. 13, 19, 20, 28, 37, 41, 44, 52, 181, 182, Glossary: MQTT

MVP Minimum Viable Product[164]. Glossary: MVP

NFR Non-Functional Requirements. 28, 29, 32

OSI Open Source Initiative. 51, 53, 54, Glossary: OSI

QoS Quality of Service. 32, 37, 44, 181, Glossary: QoS

RTOS Real Time Operating System. 52, Glossary: RTOS

SaaS Software as a Service. 13, Glossary: SaaS

SDK Software Development Kit. 39, Glossary: SDK

SNS Simple Notification Service . 13, 18, 36, Glossary: SNS

SQS Simple Queue Service[182] . 20, 22, 23, Glossary: SQS

STOMP Simple Text Oriented Messaging Protocol[183]. 19, Glossary: STOMP

TCP Transmission Control Protocol[185]. 23, Glossary: TCP

TLS Transport Layer Security. 162

UML Unified Modeling Language. 54, 142, Glossary: UML

WAMP Web Application Messaging Protocol[189]. 19, 23, Glossary: WAMP

XML Extensible Markup Language[191]. 19, Glossary: XML

Bibliography

[1] Cloudevents/spec at v1.0, GitHub, 2019.
[Online]. Available: https://github.com/cloudevents/spec/tree/v1.0.

[2] Caru smart sensor | notruf für senioren | #agetech – caru - wir machen lebensräume intelligent.
2019. [Online]. Available: https://www.caruhome.com/.

[3] T. Helbling, "Project thesis proposal - cloudevent router," Apr. 2019.

[4] O. Zimmermann, "Aufgabenstellung studienarbeit," Sep. 2019.

[5] S. W. Matthias Wessendorf, Cloudevent router and gateway, GitHub.
[Online]. Available: https://github.com/rh-event-flow/cloudevents-gateway.

[6] Knative eventing, GitHub. [Online]. Available: https://github.com/knative/eventing/.

[7] Knative steering committee.
[Online]. Available: https://knative.dev/contributing/steering-committee/.

[8] Knative documentation. [Online]. Available: https://knative.dev/v0.9-docs/.

[9] Knative serving license.
[Online]. Available: https://github.com/knative/serving/blob/master/LICENSE.

[10] Knative serving license.
[Online]. Available: https://github.com/knative/eventing/blob/master/LICENSE.

[11] Pacifica dispatcher. [Online]. Available: https://pacifica-dispatcher.readthedocs.io/en/v0.2.3/.

[12] Pacifica. [Online]. Available: https://github.com/pacifica/pacifica.

[13] Pacifica contributors. [Online]. Available: https://github.com/orgs/pacifica/people.

[14] Pacifica license.
[Online]. Available: https://github.com/pacifica/pacifica/blob/master/LICENSE.

[15] Serverless event gateway.
[Online]. Available: https://github.com/serverless/event-gateway/tree/0.9.1/.

[16] Serverless event gateway.
[Online]. Available: https://github.com/serverless/event-gateway/blob/0.9.1/LICENSE.

[17] Amazon simple notification service. [Online]. Available: https://aws.amazon.com/sns/.

[18] Aws partner story: Nasa.
[Online]. Available: https://aws.amazon.com/partners/success/nasa-image-library/.

[19] Aws case study: Futbol club barcelona.
[Online]. Available: https://aws.amazon.com/solutions/case-studies/futbol-club-barcelona/.

[20] Apache camel. [Online]. Available: https://camel.apache.org/.

[21] Apache camel user stories.
[Online]. Available: https://camel.apache.org/manual/latest/user-stories.html.

[22] Apache camel commercial offerings.
[Online]. Available: https://camel.apache.org/manual/latest/commercial-camel-offerings.html.

[23] Apache camel license.
[Online]. Available: https://github.com/apache/camel/blob/master/LICENSE.txt.

[24] Crossbar.io. [Online]. Available: https://crossbar.io/.

[25] Crossbar.io, Crossbar/crossbar: Crossbar.io - wamp application router, 2019. [Online]. Available: https://github.com/crossbario/crossbar.

[26] Crossbar.io users. [Online]. Available: https://crossbar.io/users/.

[27] Crossbar.io and mindsphere. [Online]. Available: https://crossbario.com/blog/Crossbar-and-Mindsphere-At-the-show/.

[28] Prodisys - fortiss, 2019. [Online]. Available: https://prodisys.fortiss.org/en/.

[29] Crossbar.io license. [Online]. Available: https://github.com/crossbario/crossbar/blob/master/crossbar/LICENSE.

[30] Introduction to d-bus, 2013. [Online]. Available: https://www.freedesktop.org/wiki/IntroductionToDBus/.

[31] Dbus, Dec. 2017. [Online]. Available: https://www.freedesktop.org/wiki/Software/dbus/.

[32] H. Pennington, A. Carlsson, A. Larsson, S. Herzberg, S. McVittie, and D. Zeuthen, D-bus specification, May 2019. [Online]. Available: https://dbus.freedesktop.org/doc/dbus-specification.html.

[33] Freedesktop.org, 2019. [Online]. Available: https://www.freedesktop.org/wiki/.

[34] Freedesktop.org users, 2019. [Online]. Available: https://wiki.debian.org/FreeDesktop.

[35] D-bus users, 2019. [Online]. Available: https://www.freedesktop.org/wiki/Software/DbusProjects/.

[36] Modemmanager, 2019. [Online]. Available: https://www.freedesktop.org/wiki/Software/ModemManager/.

[37] D-bus license, 2019. [Online]. Available: https://dbus.freedesktop.org/doc/COPYING.

[38] Node-red. [Online]. Available: https://nodered.org/.

[39] N. O'Leary, "Moving to the js foundation," Oct. 2016. [Online]. Available: https://nodered.org/blog/2016/10/17/js-foundation.

[40] Getting started with watson iot platform using node-red, Jun. 2016. [Online]. Available: https://developer.ibm.com/recipes/tutorials/getting-started-with-watson-iot-platform-using-node-red/.

[41] 2019 node-red community survey. [Online]. Available: https://nodered.org/about/community/survey/2019/.

[42] Netiot edge. [Online]. Available: https://www.netiot.com/edge/.

[43] Node-red license, 2019. [Online]. Available: https://github.com/node-red/node-red/blob/master/LICENSE.

[44] Messaging that just works — rabbitmq, 2019. [Online]. Available: https://www.rabbitmq.com/.

[45] Which protocols does rabbitmq support? 2019. [Online]. Available: https://www.rabbitmq.com/protocols.html.

[46] Products and success stories. [Online]. Available: https://www.amqp.org/about/examples.

[47] Mpl 2.0 faq. [Online]. Available: https://www.mozilla.org/en-US/MPL/2.0/FAQ/.

[48] Rabbitmq license. [Online]. Available: https://github.com/rabbitmq/rabbitmq-server/blob/master/LICENSE-MPL-RabbitMQ.

[49] Cloudevents-gateway/httpeventpublisher.java at master · rh-event-flow/cloudevents-gateway, 2018. [Online]. Available: https://github.com/rh-event-flow/cloudevents-gateway/blob/master/http-sink/src/main/java/io/streamzi/cloudevents/gateway/sink/eventbus/HttpEventPublisher.java.

[50] Knative eventing. [Online]. Available: https://knative.dev/v0.9-docs/eventing/.

[51] Knative sources. [Online]. Available: https://knative.dev/v0.9-docs/eventing/sources/.

[52] Broker and trigger | knative.
[Online]. Available: https://knative.dev/v0.9-docs/eventing/broker-trigger/.

[53] Stackoverflow knative.
[Online]. Available: https://stackoverflow.com/questions/tagged/knative/.

[54] Pacifica/pacifica-dispatcher at v0.2.3.
[Online]. Available: https://github.com/pacifica/pacifica-dispatcher.

[55] Pacifica/pacifica-dispatcher-example: Pacifica dispatcher example, 2019.
[Online]. Available: https://github.com/pacifica/pacifica-dispatcher-example.

[56] Amazon sns - message attributes.
[Online]. Available: https://docs.aws.amazon.com/sns/latest/dg/sns-message-attributes.html.

[57] Amazon sns - message filtering.
[Online]. Available: https://docs.aws.amazon.com/sns/latest/dg/sns-message-filtering.html.

[58] Stackoverflow knative.
[Online]. Available: https://stackoverflow.com/questions/tagged/amazon-sns/.

[59] Stackoverflow knative.
[Online]. Available: https://stackoverflow.com/questions/tagged/apache-camel/.

[60] Welcome to crossbar.io — crossbar.io application router 19.10.1 documentation, 2019.
[Online]. Available: https://crossbar.io/docs/.

[61] Documentation: Node-red. [Online]. Available: https://nodered.org/docs/.

[62] Node-red/node-red: Low-code programming for event-driven applications, 2019.
[Online]. Available: https://github.com/node-red/node-red.

[63] Rabbitmq tutorial - routing — rabbitmq.
[Online]. Available: https://www.rabbitmq.com/tutorials/tutorial-four-python.html.

[64] Rabbitmq tutorial - topics — rabbitmq.
[Online]. Available: https://www.rabbitmq.com/tutorials/tutorial-five-python.html.

[65] Rabbitmq/rabbitmq-autocluster: Rabbitmq peer discovery and cluster formation plugin, supports rabbitm
2018. [Online]. Available: https://github.com/rabbitmq/rabbitmq-autocluster.

[66] Rabbitmq/rabbitmq-server: Pen source multi-protocol messaging broker, 2019.
[Online]. Available: https://github.com/rabbitmq/rabbitmq-server.

[67] Dbus / dbus · gitlab, 2019.
[Online]. Available: https://gitlab.freedesktop.org/dbus/dbus/tree/dbus-1.12.

[68] Walk through another example - apache camel. [Online]. Available:
https://camel.apache.org/manual/latest/walk-through-another-example.html.

[69] Walk through an example code - apache camel.
[Online]. Available: https://camel.apache.org/manual/latest/walk-through-an-example.html.

[70] Camel/examples/camel-example-jms-file at master · apache/camel. [Online]. Available:
https://github.com/apache/camel/tree/master/examples/camel-example-jms-file.

[71] Supported node versions : Node-red.
[Online]. Available: https://nodered.org/docs/faq/node-versions.

[72] Creating your first flow : Node-red.
[Online]. Available: https://nodered.org/docs/tutorials/first-flow.

[73] Eventing/mission.md at master · knative/eventing, 2019.
[Online]. Available: https://github.com/knative/eventing/blob/master/docs/mission.md.

[74] Knative channels. [Online]. Available: https://knative.dev/v0.9-docs/eventing/channels/.

[75] Getting started | knative.
[Online]. Available: https://knative.dev/v0.9-docs/eventing/getting-started/.

[76] Container source example | knative.
[Online]. Available: https://knative.dev/v0.9-docs/eventing/samples/container-source/.

[77] Apache camel components code.
[Online]. Available: https://github.com/apache/camel/tree/master/components.

[78] Node-red node library. [Online]. Available: https://flows.nodered.org/search?type=node.

[79] Node-red core nodes. [Online]. Available: https://github.com/node-red/node-red/tree//1.0.0/packages/node_modules/%40node-red/nodes.

[80] Apache camel type converter.
[Online]. Available: https://camel.apache.org/manual/latest/type-converter.html#TypeConverter-WritingyourownTypeConverters.

[81] Apache camel extensions for quarkus.
[Online]. Available: https://camel.apache.org/camel-quarkus/latest/.

[82] Quarkus. [Online]. Available: https://quarkus.io.

[83] Open unified processe, 2007.
[Online]. Available: https://www.eclipse.org/epf/general/OpenUP.pdf.

[84] Arc42 - arc42. [Online]. Available: https://arc42.org/.

[85] G. Starke and P. Hruschka, Introduction and goals.
[Online]. Available: https://docs.arc42.org/section-1/.

[86] ——, Architecture constraints. [Online]. Available: https://docs.arc42.org/section-2/.

[87] ——, Context and scope. [Online]. Available: https://docs.arc42.org/section-3/.

[88] ——, Solution strategy. [Online]. Available: https://docs.arc42.org/section-4/.

[89] ——, Building block view. [Online]. Available: https://docs.arc42.org/section-5/.

[90] ——, Runtime view. [Online]. Available: https://docs.arc42.org/section-6/.

[91] ——, Deployment view. [Online]. Available: https://docs.arc42.org/section-7/.

[92] ——, Crosscutting concepts. [Online]. Available: https://docs.arc42.org/section-8/.

[93] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal,
Pattern-oriented software architecture. Jul. 12, 1996, 476 pp., ISBN: 0471958697.
[Online]. Available:
https://www.ebook.de/de/product/3239671/frank_buschmann_regine_meunier_hans_rohnert_peter_sommerlad_michael_stal_pattern_oriented_software_architecture.html.

[94] G. Starke and P. Hruschka, Architecture decisions.
[Online]. Available: https://docs.arc42.org/section-9/.

[95] M. Nygard, Documenting architecture decisions, Nov. 5, 2019. [Online]. Available:
http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions.

[96] Markdown architectural decision records, Nov. 5, 2019.
[Online]. Available: https://github.com/adr/madr/tree/2.1.2.

[97] U. Zdun, R. Capill, H. Tran, and O. Zimmermann, Sustainable architectural design decisions,
Mar. 9, 2014. [Online]. Available:
https://www.infoq.com/articles/sustainable-architectural-design-decisions/.

[98] G. Starke and P. Hruschka, Quality requirements.
[Online]. Available: https://docs.arc42.org/section-10/.

[99] Cloudevents specification v0.3, 2019.
[Online]. Available: https://github.com/cloudevents/spec/blob/v0.3/spec.md.

[100] ——, Risks and technical debt. [Online]. Available: https://docs.arc42.org/section-11/.

[101] L. A. Williams, "The collaborative software process,"
PhD thesis, The University of Utah, 2000.
[Online]. Available: http://collaboration.csc.ncsu.edu/laurie/Papers/dissertation.pdf.

[102] M. Fowler, "Pairprogrammingmisconceptions," 2006.
[Online]. Available: https://martinfowler.com/bliki/PairProgrammingMisconceptions.html.

[103] Rust platform support, 2019.
[Online]. Available: https://forge.rust-lang.org/release/platform-support.html.

[104] Amazon sns subscription filter policies. [Online]. Available:
https://docs.aws.amazon.com/sns/latest/dg/sns-subscription-filter-policies.html.

[105] Eclipse paho - mqtt and mqtt-sn software, Dec. 17, 2019.
[Online]. Available: https://www.eclipse.org/paho/.

[106] A. Banks and R. Gupta, Mqtt version 3.1.1 plus errata 01,
OASIS Standard Incorporating, Dec. 2015. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html.

[107] The opentracing project, 2019. [Online]. Available: https://opentracing.io/.

[108] Prometheus - monitoring system and time series database, 2019.
[Online]. Available: https://prometheus.io/.

[109] Multi-producer, single-consumer fifo queue communication primitives, 2019.
[Online]. Available: https://doc.rust-lang.org/std/sync/mpsc/index.html.

[110] Doc - the cargo book, 2019.
[Online]. Available: https://doc.rust-lang.org/cargo/commands/cargo-doc.html.

[111] R. Hanmer, Patterns for fault tolerant software. John Wiley & Sons, Jul. 12, 2013, 320 pp.
[Online]. Available: https://www.ebook.de/de/product/21164648/robert_hanmer_patterns_for_fault_tolerant_software.html.

[112] C. N. Steve Klabnik, The rust programming language.
https://doc.rust-lang.org/book/: Random House LCC US, Aug. 12, 2019, ISBN: 1718500440.
[Online]. Available: https://www.ebook.de/de/product/37149179/steve_klabnik_carol_nichols_the_rust_programming_language_covers_rust_2018.html.

[113] Introduction to latex. [Online]. Available: https://www.latex-project.org/about/.

[114] The latex project public license. [Online]. Available: https://www.latex-project.org/lppl/.

[115] Academic free license ("afl") v. 3.0 | open source initiative.
[Online]. Available: https://opensource.org/licenses/AFL-3.0.

[116] T. Woods, "'age-tech': The next frontier market for technology disruption," Forbes, 2019.
[Online]. Available: https://www.forbes.com/sites/tinawoods/2019/02/01/age-tech-the-next-frontier-market-for-technology-disruption/#567c4dea6c84.

[117] 1.3. amqp - advanced message queuing protocol red hat enterprise mrg 3 | red hat customer portal, 2019. [Online]. Available:
https://access.redhat.com/documentation/en-us/red_hat_enterprise_mrg/3/html/messaging_programming_reference/amqp____advanced_message_queuing_protocol.

[118] R. Godfrey, D. Ingham, and R. Schloming,
OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0, OASIS, Oct. 2012.
[Online]. Available: https://www.oasis-open.org/standards#amqpv1.0.

[119] Apache license, version 2.0, 2019.
[Online]. Available: https://www.apache.org/licenses/LICENSE-2.0.

[120] P. Eising, What exactly is an api? - perry eising - medium, Medium, Dec. 2017.
[Online]. Available: https://medium.com/@perrysetgo/what-exactly-is-an-api-69f36968a41f.

[121] Download arc42 - arc42. [Online]. Available: https://arc42.org/download.

[122] Arc42 license - arc42. [Online]. Available: https://arc42.org/license.

[123] What is aws. [Online]. Available: https://aws.amazon.com/what-is-aws/.

[124] Buildkite features, 2019. [Online]. Available: https://buildkite.com/features.

[125] Knative serving license. [Online]. Available: https://techterms.com/definition/bytecode.

[126] B. Stroustrup, Programming. Addison Wesley, May 15, 2014, 1312 pp., ISBN: 0321992784. [Online]. Available: https://www.ebook.de/de/product/21839507/bjarne_stroustrup_programming.html.

[127] H. Patel, Python - c++ bindings | hardik patel, Dec. 2017. [Online]. Available: https://www.hardikp.com/2017/12/30/python-cpp/.

[128] Creative commons - cc0, Nov. 5, 2019. [Online]. Available: https://creativecommons.org/share-your-work/public-domain/cc0.

[129] Continuous integration | thoughtworks, 2019. [Online]. Available: https://www.thoughtworks.com/continuous-integration.

[130] Cncf cloud native definition v1.0, 2019. [Online]. Available: https://github.com/cncf/toc/blob/master/DEFINITION.md.

[131] [Online]. Available: https://www.cncf.io/cncf-kubernetes-project-journey/.

[132] Cncf prometheus project journey - cloud native computing foundation, 2019. [Online]. Available: https://www.cncf.io/cncf-prometheus-project-journey/.

[133] Members - cloud native computing foundation. [Online]. Available: https://www.cncf.io/about/members/.

[134] What is copyleft? [Online]. Available: https://www.gnu.org/licenses/copyleft.en.html.

[135] What is docker? Aug. 2018. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/docker-defined.

[136] S. Efftinge, M. Volter, A. Haase, and B. Kolb, The pragmatic code generator programmer, Sep. 2006. [Online]. Available: https://www.theserverside.com/news/1365073/The-Pragmatic-Code-Generator-Programmer.

[137] Eclipse mosquitto - an open source mqtt broker. [Online]. Available: https://mosquitto.org/.

[138] Amazon elastic kubernetes service. [Online]. Available: https://aws.amazon.com/eks/.

[139] M. Rouse, "What is function as a service (faas)? - definition from whatis.com," TechTarget, 2018.

[140] Freertos - market leading rtos (real time operating system) for embedded systems with internet of things [Online]. Available: https://www.freertos.org/.

[141] S. Chacon and B. Straub, Pro git. Apress, 2014.

[142] K. Johnson, Github passes 100 million repositories, Nov. 2018. [Online]. Available: https://venturebeat.com/2018/11/08/github-passes-100-million-repositories/.

[143] Gnu general public license v2.0 - gnu project - free software foundation, Sep. 2017. [Online]. Available: https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html.

[144] The gnu general public license v3.0 - gnu project - free software foundation, Nov. 2016. [Online]. Available: https://www.gnu.org/licenses/gpl-3.0.html.

[145] C. Doxsey, An introduction to programming in go. 2012, ISBN: 978-1478355823. [Online]. Available: http://www.golang-book.com/books/intro.

[146] The go project - the go programming language, 2019. [Online]. Available: https://golang.org/project/.

[147] H. F. Nielsen, J. Mogul, L. M. Masinter, R. T. Fielding, J. Gettys, P. J. Leach, and T. Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, RFC 2616, Jun. 1999. DOI: 10.17487/RFC2616. [Online]. Available: https://rfc-editor.org/rfc/rfc2616.txt.

[148] P. Mukhopadhyay, "Sales transformations for "industry 4.0"," Forbes, Oct. 2019. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2019/10/16/sales-transformations-for-industry-4-0/#23aafb725a42.

[149] C. McClelland, "What is iot? - a simple explanation of the internet of things," IoT For All, May 2019. [Online]. Available: https://www.iotforall.com/what-is-iot-simple-explanation/.

[150] Apr. 2012. [Online]. Available: https://techterms.com/definition/java.

[151] Javascript definition, Aug. 2014. [Online]. Available: https://techterms.com/definition/javascript.

[152] Ecmascript® 2019 language specification, Ecma International. [Online]. Available: https://www.ecma-international.org/ecma-262/10.0/index.html.

[153] Jdk definition from pc magazine encyclopedia. [Online]. Available: https://www.pcmag.com/encyclopedia/term/45608/jdk.

[154] Json - javascript | mdn, Nov. 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON.

[155] B. Venners, Inside the java virtual machine. McGraw Hill, 1999, ISBN: 0-07-135093-4.

[156] What is apache kafka? [Online]. Available: https://aws.amazon.com/msk/what-is-kafka/.

[157] Cncf kubernetes project journey. [Online]. Available: https://www.cncf.io/cncf-kubernetes-project-journey/.

[158] Linux from foldoc, 2000. [Online]. Available: http://foldoc.org/linux.

[159] The linux kernel archives - faq. [Online]. Available: https://www.kernel.org/category/faq.html.

[160] J. Gruber, Daring fireball: Markdown. [Online]. Available: https://daringfireball.net/projects/markdown/.

[161] Maven – welcome to apache maven. [Online]. Available: https://maven.apache.org/.

[162] G. Haff, The mysterious history of the mit license | opensource.com, Apr. 2019. [Online]. Available: https://opensource.com/article/19/4/history-mit-license.

[163] Mqtt homepage. [Online]. Available: http://mqtt.org/.

[164] "Minimum viable product (mvp)," [Online]. Available: https://www.techopedia.com/definition/27809/minimum-viable-product-mvp.

[165] Introduction to node.js. [Online]. Available: https://nodejs.dev/.

[166] Frequently answered questions: What is "open source" software? [Online]. Available: https://opensource.org/faq#osd.

[167] About the open source initiative | open source initiative. [Online]. Available: https://opensource.org/about.

[168] Drawing uml with plantuml, plantuml language reference guide, 2019. [Online]. Available: http://plantuml.com/guide.

[169] About pull requests. [Online]. Available: https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests.

[170] D. Kuhlman, A python book: Beginning python, advanced python, and python exercises, 1.1a. Apr. 2012. [Online]. Available: https://web.archive.org/web/20120623165941/http://cutter.rexx.com/~dkuhlman/python_book_01.html.

[171] E.800 : Definitions of terms related to quality of service,
International Telecommunication Union (ITU), Sep. 2008.
[Online]. Available: https://www.itu.int/rec/T-REC-E.800-200809-I/en.

[172] Why rtos and what is rtos? [Online]. Available: https://www.freertos.org/about-RTOS.html.

[173] D. Bryant, "A quantum leap for the web," Medium, 2016. [Online]. Available:
https://medium.com/mozilla-tech/a-quantum-leap-for-the-web-a3b7174b3c12.

[174] Rust. [Online]. Available: https://research.mozilla.org/rust/.

[175] Production users. [Online]. Available: https://www.rust-lang.org/production/users.

[176] "Stack overflow developer survey 2019," Tech. Rep., 2019.
[Online]. Available: https://insights.stackoverflow.com/survey/2019.

[177] The rustc book. [Online]. Available: https://doc.rust-lang.org/rustc/what-is-rustc.html.

[178] [Online]. Available: https://www.rust-lang.org/tools/install.

[179] M. Rouse, "What is software as a service (saas)? - definition from whatis.com,"
TechTarget, Feb. 2019.

[180] What is a serial interface? [Online]. Available:
https://www3.nd.edu/~lemmon/courses/ee224/web-manual/web-manual/lab9/node4.html.

[181] Mindsphere - cloud-based, open iot operating system | software | siemens.
[Online]. Available: https://new.siemens.com/global/en/products/software/mindsphere.html.

[182] Amazon simple queue service. [Online]. Available: https://aws.amazon.com/sqs/.

[183] Stomp protocol specification, version 1.2, 2012.

[184] Free time tracking software & app - tmetric. [Online]. Available: https://tmetric.com/.

[185] J. Postel, Transmission control protocol, RFC 793, Sep. 1981.
DOI: 10.17487/RFC0793. [Online]. Available: https://rfc-editor.org/rfc/rfc793.txt.

[186] C. Larman, Applying uml and patterns. Prentice Hall, 2004, vol. 3, ISBN: 9780131489066.

[187] D. Ritchie and K. Thompson, The unix time-sharing system, Jul. 1978.
[Online]. Available: https://archive.org/details/bstj57-6-1905.

[188] Unix(7) - linux manual page.
[Online]. Available: http://man7.org/linux/man-pages/man7/unix.7.html.

[189] T. Oberstein and A. Goedde, The web application messaging protocol, IETF, Aug. 2019.
[Online]. Available:
https://wamp-proto.org/_static/gen/wamp_latest_ietf.html#rfc.authors.

[190] A. Melnikov and I. Fette, The WebSocket Protocol, RFC 6455, Dec. 2011.
DOI: 10.17487/RFC6455. [Online]. Available: https://rfc-editor.org/rfc/rfc6455.txt.

[191] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau,
Extensible markup language (xml) 1.0 (fifth edition), 2008.
[Online]. Available: https://www.w3.org/TR/2008/REC-xml-20081126/.

[192] Istio / quick start evaluation install.
[Online]. Available: https://istio.io/docs/setup/install/kubernetes/.

[193] G. M. Poore, 2017.
[Online]. Available: http://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf.

[194] Rust support for visual studio code, GitHub, 2019.
[Online]. Available: https://github.com/rust-lang/rls-vscode.

[195] Installing knative | knative.
[Online]. Available: https://knative.dev/v0.9-docs/install/index.html.

[196] Install on minikube | knative.
[Online]. Available: https://knative.dev/v0.9-docs/install/knative-with-minikube/.

[197] Istio / setup. [Online]. Available: https://istio.io/docs/setup/#downloading-the-release.

[198] Knative install using gloo on a kubernetes cluster.
[Online]. Available: https://knative.dev/docs/install/knative-with-gloo/.

A. Aufgabenstellung

Aufgabenstellung Studienarbeit
Linus Basig, Fabrizio Lazzaretti

CloudEvent Router

1. Betreuer

Diese Studienarbeit wird in Zusammenarbeit mit CARU AG durchgeführt.

Betreuer HSR:

Prof. Dr. Olaf Zimmermann, Institut für Software, ozimmerm@hsr.ch

Ansprechpartner CARU AG:

Dr. Thomas Helbling thomas.helbling@caruhome.com

2. Baseline (Ausgangslage)

The Serverless Working Group of the Cloud Native Computing Foundation has proposed CloudEvents, a specification to describe events. This specification seeks to ease event declaration and delivery across services, platforms and beyond. It received a lot of interest and contributions from major cloud providers and Software-as-a-Service (SaaS) companies. One application area targeted by CloudEvents is the collection and processing of events produced by Internet-of-Things (IoT) devices. IoT devices often are not directly attached to the Internet and therefore require an intermediate to relay their events to the cloud for processing. This intermediate has often to translate the event from one transport layer binding to another.

CARU AG is an AgeTech company which enables the elderly to live independently through technology. It provides security and social integration without being intrusive. The CARU product is composed of an IoT device, which collects sensor data and provides a simple user interface (touch and voice), and a web page on which more services are provided. The product has a highly event-driven architecture, and CARU already use CloudEvents for the events produced by its cloud services.

It is desired to use a CloudEvent Router to bring cloud events to CARU's device. The router would allow the subsystems of the device and the cloud exchange information seamlessly.

3. Goals and Deliverables (Ziele der Arbeit und Liefergegenstände)

The proposed project goal is the design and implementation of a CloudEvent Router. This router should allow the routing of events between "ports" which bind to the transport layers (HTTP, MQTT, NATS, AMPQ) defined by the specification. For now, we propose the routing based on static rules, but the design should allow the implementation of dynamic routing in the future.

Expected work results:

- A scientifically crafted design proposal for the CloudEvent Router.
- A minimal implementation featuring at least two port implementations.

The router should be easy to extend with own port implementations which bind to different transport layers. The rules engine should be designed to be powerful, performant and easy to configure.

Additional critical success factors are:

- CARU would like to see the CloudEvent Router available to the public as a open source project and therefore require that the project is licensed under an appropriate license.
- CARU must be able to maintain the project in the feature if the authors are unable to do so.
- CARU has a strong preference for Rust as a programming language.
- Reuse of already existing open source projects has to be considered (e.g., Apache Camel, possibly also camel-jobs originating a from previous student project/term thesis at HSR FHO).

4. Unterstützung

Die erwartete und effektiv erhaltene Unterstützung wird durch die Studenten in Sitzungsprotokollen definiert und im SA-Bericht dokumentiert.

5. Zur Durchführung

Mit dem HSR-Betreuer finden in der Regel wöchentlich Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf zu veranlassen. Besprechungen mit dem Auftraggeber werden nach Bedarf durchgeführt.

Alle Besprechungen, bei denen eine Vorbereitung durch den Betreuer nötig ist, sind von den Studenten mit einer Traktandenliste vorzubereiten. Beschlüsse sind in einem Protokoll zu dokumentieren. Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. Arbeitszeiten sind zu dokumentieren.

Die Spezifikation der Anforderungen geschieht durch die Studenten in Absprache mit dem Betreuer. Bei Disputen entscheidet der Betreuer in Rücksprache mit den Studenten über die definitiv für die Studienarbeit relevanten Anforderungen.

Vorstudie, Anforderungsdokumentation und Architekturdokumentation sollten im Laufe des Projektes mittels Milestones mit dem Auftraggeber und dem Betreuer in einem stabilen Zustand abgenommen werden. Zu den abgegebenen Arbeitsresultaten wird ein vorläufiges Feedback abgegeben. Eine definitive Beurteilung erfolgt auf Grund der am Abgabetermin abgelieferten Dokumentation.

Die Rechte an den Ergebnissen der Studienarbeit werden in einer separaten Vereinbarung definiert. (gemäss Vorlage Studiengang: die Ergebnisse der Arbeit dürfen sowohl von den Studenten wie von der HSR und Prof. Zimmermann nach Abschluss der Arbeit verwendet und weiter entwickelt werden, da sie als Open Source Projekt veröffentlicht werden sollen).

6. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen. Die zu erstellenden Dokumente sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollten den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Bei der Projektdokumentation und der Abgabe sind die Anleitungen des Studienganges inklusive Anhängen zu beachten.

7. Termine

Siehe HSR-Webseiten. Allfällige weitere Termine sind am Sekretariat der Abteilung Informatik zu erfragen und sollten entsprechend in einem Sitzungsprotokoll dokumentiert werden.

8. Beurteilung

Eine erfolgreiche Studienarbeit zählt 8 ECTS-Punkte pro Studierenden. Für 1 ECTS-Punkt ist eine Arbeitsleistung von ca. 25 bis 30 Stunden budgetiert.

Siehe http://studien.hsr.ch/allModules/23498_M_SAI.html für die Modulbeschreibung der Studienarbeiten.

Für die Beurteilung sind die HSR-Betreuer verantwortlich unter Einbezug des Feedbacks der Projektpartner.

Gesichtspunkt	Gewicht
1. Organisation, Durchführung	1/5
2. Berichte (Abstract, Management Summary, technische u. persönliche Berichte) sowie Gliederung, Darstellung und Sprache der gesamten Dokumentation.	1/5
3. Inhalt*)	3/5

*) Die Unterteilung und Gewichtung von 3. Inhalt wird im Laufe dieser Arbeit festgelegt.

Im Übrigen gelten die Bestimmungen der Abt. Informatik zur Durchführung von Studienarbeiten.

Rapperswil, den 18.09.2019

Prof. Dr. Olaf Zimmermann
Institut für Software
Hochschule für Technik Rapperswil

B. Architecture Documentation (arc42)

CloudEvents Router

arc42

Student Research Project Thesis

University of Applied Sciences Rapperswil (HSR)
University of Applied Sciences of Eastern Switzerland (FHO)
Department of Computer Science

Fall Term 2019/2020

Authors	Linus Basig, Fabrizio Lazzaretti
Advisor	Prof. Dr. Olaf Zimmermann
Project Partner	CARU AG

Printing Date	December 19, 2019

About arc42

arc42, the Template for documentation of software and system architecture.

By Dr. Gernot Starke, Dr. Peter Hruschka and contributors.

Template Revision: 7.0 EN (based on asciidoc), January 2017

Contents

1 Introduction and Goals

The goal of this project is to create a CloudEvents Router to bring CloudEvents to the CARU device. The router would allow the subsystems of the device and the cloud exchange information seamlessly.

The name of the product is CERK. CERK stands for CloudEvents Router with a MicroKernel.

1.1 Functional Requirements (Business Goals)

Table 1.1 shows a list of the functional-requirements distilled from the use cases in subsection 1.1.1. Only the functional requirements in Table 1.1 are part of the scope of the project.

No	Requirement
FR-1	The router must understand the CloudEvents specification version ~~0.3~~ 1.0.[1]
FR-2	The router must be able to run on a modern Linux distribution
FR-3	A simple way to define routing rules based on the CloudEvents attributes must be designed
FR-4	The router should offer a UNIX socket interface to send and receive CloudEvents
FR-5	The router should offer a MQTT version 3.1.1 interface to send and receive CloudEvents
FR-6	The router should provide a simple interface to access debugging information

Table 1.1: Functional Requirements

1.1.1 Use Cases

The use cases give insights where these requirements are coming from and give context to the problems they should address. Only the main success scenarios are relevant for the requirements. The extensions are meant to be kept in mind when designing the system and are not part of the project's scope. The implementation and testing on real CARU Smart Sensor hardware or in the cloud is not part of the project's scope.

Routing on the Edge (Inter-Process)

Context There are multiple services running on the CARU Smart Sensor's main processor. Each of these services has a specific task like interfacing with a hardware component, communicating over the network or coordinating other services with the help of a state machine. The services are communicating with each other by emitting events and consuming the events of the other services.

Currently, most of these services are running in one single Python process and use an in-process broadcast to publish their events. There are already a few services which, for performance or tooling reasons, are implemented in another programming language and run in a separate process. For each

[1] CloudEvents specification version 1.0 was released on the 24. of October and it was decided with the customer to support it.

of these external processes, there needs to be a service in the Python process to communicate with the external process and make its events available. This is an unnecessary step and adds undesired complexity.

Main Success Scenario The CloudEvents Router enables all services to run in separate processes by facilitating the controlled exchange of events between them. The exchange is defined by the routing rules.

1. Service A, B, C and D are communicating to each other via the CloudEvents Router.
2. Service A publishes an event by writing it to a UNIX socket it shares with the CloudEvents Router.
3. The CloudEvents Router receives the event and loads relevant routing rules from a configuration file.
4. The CloudEvents Router applies the rules to the CloudEvents attributes of the event to figure out which other services are interested in the event. Service C and D are interested in the event.
5. The CloudEvents Router forwards the event to the UNIX sockets associated to service C and D.
6. Service C and D receive the event on their socket and do something useful with it.

Extensions

Hardware Emulation The CloudEvents Router enables services, which interface with hardware components, to be easily replaced by a faker services which simulates the hardware and emits the corresponding events.

1. A developer wants to test something without deploying his changes to a CARU Smart Sensor.
2. The developer runs a command and the CloudEvents Router, the hardware-independent services and the faker services get started.
3. All the services start communicate via the CloudEvents Router.
4. The developer interacts with a web-page provided by the faker service to simulate the interaction with the device.

Local Debugging The CloudEvents Router enables the easy inspection of the events passed between the services.

1. A developer needs to debug an unexpected behavior of a CARU Smart Sensor.
2. He logs into the device and uses a tool to connect to the CloudEvents Router using a special debug UNIX socket
3. The CloudEvents Router starts sending the received events, the routing decisions and performance metrics over the socket.
4. The tool displays this information and helps the developer to find the issue.

Routing in the Cloud

Context CARU uses the public cloud to offer a variety of services related to their CARU Smart Sensor. The functionality is similarly organized as on the device and the different services are also communicating with each other by emitting events.

Currently, all the events exchanged between the cloud services are structured according to the CloudEvents specification version 0.3. They use a proprietary product of the cloud provider to route the events between the different services. The routing capabilities of the proprietary product are quite limited. (Whitelisting, Blacklisting and Prefix-Matching) [2]

Main Success Scenario The CloudEvents Router enables the services in the cloud to receive CloudEvents based on more complex routing rules than currently available.

Extensions

Dynamic Routing The CloudEvents Router offers an api to dynamically adapt the configuration and routing rules without an interruption.

1. A customer wants some specific events emitted from his CARU Smart Sensors to be forwarded to a message broker he owns.
2. He enters the filter conditions and the configuration for his broker into the configuration console.
3. The service owing the configuration console calls the api of the CloudEvents Router to configure the event transport and the routing rule.
4. The deployed CloudEvents Router instances pick up the new configuration and routing rule and start forwarding the relevant events.

Routing between the Edge and the Cloud

Context The CARU Smart Sensor is connected to the cloud via the MQTT protocol version 3.1.1. One of the services on the main processor of the CARU Smart Sensor is responsible for sending and receiving messages over this connection. [1]

Currently, the events exchanged between the services on the CARU Smart Sensor and the events exchanged between the device and the cloud are not following the CloudEvents specifications. When they are received by the cloud they have to be converted into a cloud event and republished.

Main Success Scenario The CloudEvents Router enables events to be seamlessly exchanged between services on the CARU Smart Sensor and services in the cloud.

1. Service A and B are connected to a CloudEvents Router on the device via MQTT. Cloud Service C and D are connected to a CloudEvents Router in the cloud. The CloudEvents Router on the device is connected to the CloudEvents Router in the cloud via MQTT broker.
2. Service A publishes an event to the CloudEvents Router on the device via UNIX socket.
3. The CloudEvents Router on the device applies the routing rules and knows service B and the CloudEvents Router in the cloud is interested in the event.

[2] docs.aws.amazon.com/sns/latest/dg/sns-message-filtering.html

4. The CloudEvents Router on the device forwards the event to the local service B via UNIX socket and to the cloud via MQTT.
5. The CloudEvents Router in the cloud receives via MQTT and applies its routing rules. It knows service D is interested in the event.
6. The CloudEvents Router in the cloud forwards the event to service D over the appropriate event transport.

Extensions

Remote Debugging The CloudEvents Router enables the remote inspection of the events passed between the services on the CARU Smart Sensor.

1. A developer needs to debug an unexpected behavior of a CARU Smart Sensor.
2. He logs into the debug cloud console and selects the device he wants to debug.
3. The debug cloud console sends a command via MQTT to the CloudEvents Router on the device set it into debug mode.
4. The CloudEvents Router on the device starts sending the received events, the routing decisions and performance metrics over MQTT to the cloud.
5. The debug cloud console displays this information and helps the developer to find the issue.

MQTT Quality of Service (QoS) The CloudEvents Router respects the QoS properties of the transport protocols.

1. The CloudEvents Router receives an event from via MQTT. The MQTT message has the QoS set to 1. This means the message should be delivered at least once.
2. The CloudEvents Router applies the routing rules and knows that services A and B are interested in the event.
3. The CloudEvents Router forwards the event to service A and B.
4. After successfully sending the event to service A and B, the CloudEvents Router acknowledges that it has processed the MQTT message.

Embedded Routing

Context The CARU Smart Sensor has, besides the main processor, an additional low power processor running a Real Time Operating System (RTOS). The task of the low power processor is interacting with the environment sensors and managing the power of the device.

Currently, the main processor communicates with the low power processor over a serial line using the Modbus[3].

Main Success Scenario The CloudEvents Router enables events to be seamlessly exchanged between the low power processor, the main processor and the cloud.

[3] modbus.org

1.2 Non-Functional Requirements (Quality Goals)

The most important quality goals can be found in Table 1.2 and are derived from the project proposal and problem definition document.[2, 3]

Broader quality requirements are listed in the quality tree in Figure 10.1.

Refer to section 10.2 for the quantification of the non-functional requirements with landing zones.

No	Goal	Scenario
NFR-1	Modularity	It should be easy to implement additional ports and configuration loaders.
NFR-2	Cross Platform	It should be easy to deploy the router to different environments especially to the cloud and embedded Linux platforms.
NFR-3	Open Source	Easy findable and followable documentation to setup the project and run it.

Table 1.2: Nonfunctional Requirements

1.3 Stakeholders

In Table 1.3 lists the stakeholders of this project.

Role	Stakeholder	Expectation
Project Team Member	Fabrizio Lazzaretti (Student)	co-project management, co-system engineer, co-development
Project Team Member	Linus Basig (Student)	co-project management, co-system engineer, co-development
Project Team Supervisor	Olaf Zimmermann (HSR)	easy understanding of architecture
Sponsor	Thomas Helbling (CARU)	understanding of the product idea
Customer	Reto Aebersold (CARU)	Knowledge of the external interfaces, easy understanding of architecture

Table 1.3: Stakeholders

2 Architecture Constraints

All the constraints are derived from the project proposal and problem definition document.[2, 3]

2.1 Technical Constraints

2.1.1 Programming Language

Rust should be used as the main programming language.

Description Rust is a low-level programming language with guaranteed thread and memory safety. It achieves that with its unique ”ownership model”. The goal is to build safe and reliable software.[4, 5]
Rust is used by Mozilla Firefox, Dropbox and many others.[6, 7]
In 2019, Rust has also been elected in the well-known survey of Stack Overflow for being the "most loved" programming language.[8]

Reason CARU would like to receive a first-hand experience report on getting started with Rust

2.1.2 Version Control

Git with GitHub should be used as the version control system.

Descriptions

Git Git is an open-source distributed version control system. The tool was created in 2005 by the Linux community.[9]

GitHub GitHub is a source code management and collaboration platform based on Git. It is a well known platform for open-source code and was acquired in 2018 by Microsoft.[10]

Reason CARU would like the product to be open-sourced at the end and GitHub is a widely used hosting platform for open-source projects

2.2 Organizational Constraints

2.2.1 Apache 2 License

The Product should be licensed under the Apache 2 license.

Description Apache 2 is an open-source license from the Apache Software Foundation. The license allow modifications and redistribution of the software. It has no copyleft.[11]

Reason CARU would like to see the product available to the public under the Apache 2 license

3 System Scope and Context

Figure 3.1 shows CARU's envisioned system context with the unified event plane implemented. Whenever events are exchanged, they are formatted according to the CloudEvents specification. The CloudEvents Routers are placed in each system and connected over an appropriate transport protocol to its neighbours. Consequently, the services are then also connected to their CloudEvents Router over an appropriate transport protocol. In some cases, the appropriate transport protocols have not been determined yet and their definition is out of the scope of this thesis. The figure shows an exemplary system context diagram derived from the use cases (see chapter 1). Because our solution should be configurable and extendable, there is an infinite amount of possible system contexts.

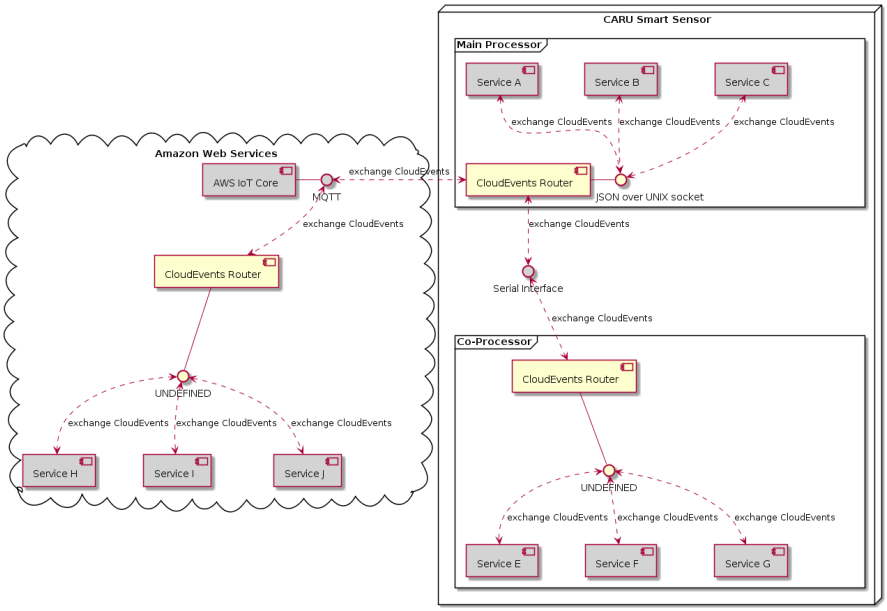

Figure 3.1: CARU's Envisioned System Context

3.1 Interfaces

This section shows more details about the interfaces displayed in **??**.

3.1.1 MQTT

MQTT is a lightweight publish/subscribe messaging transport protocol for machine to machine communication.[12, 13]

Amazon Web Services (AWS) Internet of Things (IoT) Core AWS provides a managed MQTT broker which is used by CARU to connect their devices to the cloud.[14]

CloudEvents Router The router connects to the AWS IoT Core broker over MQTT.

3.1.2 JSON over UNIX socket

"JSON is a syntax for serializing objects, arrays, numbers, strings, booleans, and null. It is based upon JavaScript syntax but is distinct from it: some JavaScript is not JSON."[15] "The `AF_UNIX` socket family is used to communicate between processes on the same machine efficiently."[16]

CloudEvents Router The router offers a UNIX socket to allow other processes running on the main processor to connect to it and exchange CloudEvents. The CloudEvents are serialized as JSON.

3.1.3 Serial Interface

"A serial interface is a communication interface between two digital systems that transmits data as a series of voltage pulses down a wire."[17]

CloudEvents Router The routers on the main processor and its co-processor exchange CloudEvents over a serial interface.

3.1.4 Other Interfaces

Some of the interfaces are not defined yet and are out of the scope of this thesis. They are in the document to illustrate the aspired setup.

4 Solution Strategy

The most important Solution Strategies are explained in the following sections. The strategies are sorted by their influence on the outcome of the project.

4.1 Content-Based Router Pattern

The router should be implemented according to the Content-Based Router pattern described in "Enterprise Integration Patterns" by Hohpe and Wolf.[18]

Helps with FR-3

Details "The Content-Based Router examines the message content and routes the message onto a different channel based on data contained in the message. The routing can be based on a number of criteria such as existence of fields, specific field values etc."[18]

4.2 Rust

The Rust programming language was requested by the customer.

Helps with Constraints

Details ADR-002

Rust is a low-level programming language with guaranteed thread and memory safety. It achieves that with its unique "ownership model". The goal is to build safe and reliable software.[4, 5]
Rust is used by Mozilla Firefox, Dropbox and many others.[6, 7]
In 2019, Rust has also been elected in the well-known survey of Stack Overflow for being the "most loved" programming language.[8]

4.3 Microkernel Pattern

The Microkernel Pattern helps with the modularization and portability of our solution.

Helps with NFR-1 and NFR-2

Details ADR-003 and MicroKernel

4.4 Prototyping

Before the final product is built, multiple prototypes were crated to minimize the risk and decide which concepts should be used for the final product.

4.5 Pair-Programming

The main components of the router were implemented with pair-programming style to bring the best possible quality.

Details Pair-programming is a style of programming where two programmers working collaboratively on the same code, design or test at the same time. The desired goal behind this strategy is to have better code quality and better know-how distribution.[19, 20]

4.6 Code Review with GitHub pull request

The master branches of the repositories were setup up as protected so that new code could be only merged into the master branch by pull request. Every pull request needs at least one reviewer. With this strategy, we can guarantee that every code was reviewed before it was merged into the master branch.

4.7 GitHub

GitHub helps us to make an open-source product that can be found and used by other programmers.

Helps with NFR-3

Details https://github.com/ce-rust/cerk
GitHub is a source code management and collaboration platform based on Git. It is a well known platform for open-source code and was acquired in 2018 by Microsoft.[10]

4.8 Apache 2 License

The Apache 2 license helps us to make an open-source product that is interesting for companies to use.

Helps with NFR-3

Details Apache 2 is an open-source license from the Apache Software Foundation. The license allow modifications and redistribution of the software. It has no copyleft.[11]

4.9 Continuous Integration

The Continuous Integration (CI) helps us to always have a compiling and runnable product.

An example user interface of the output is provided in Figure 4.1.

Details Buildkite was used as the CI. Buildkite is a service for coordinating buildpipelines which get executed on own infrastructure. They provide a build agent that can easily be deployed to AWS and other platforms. Buildkite is well integrated with GitHub.[21]

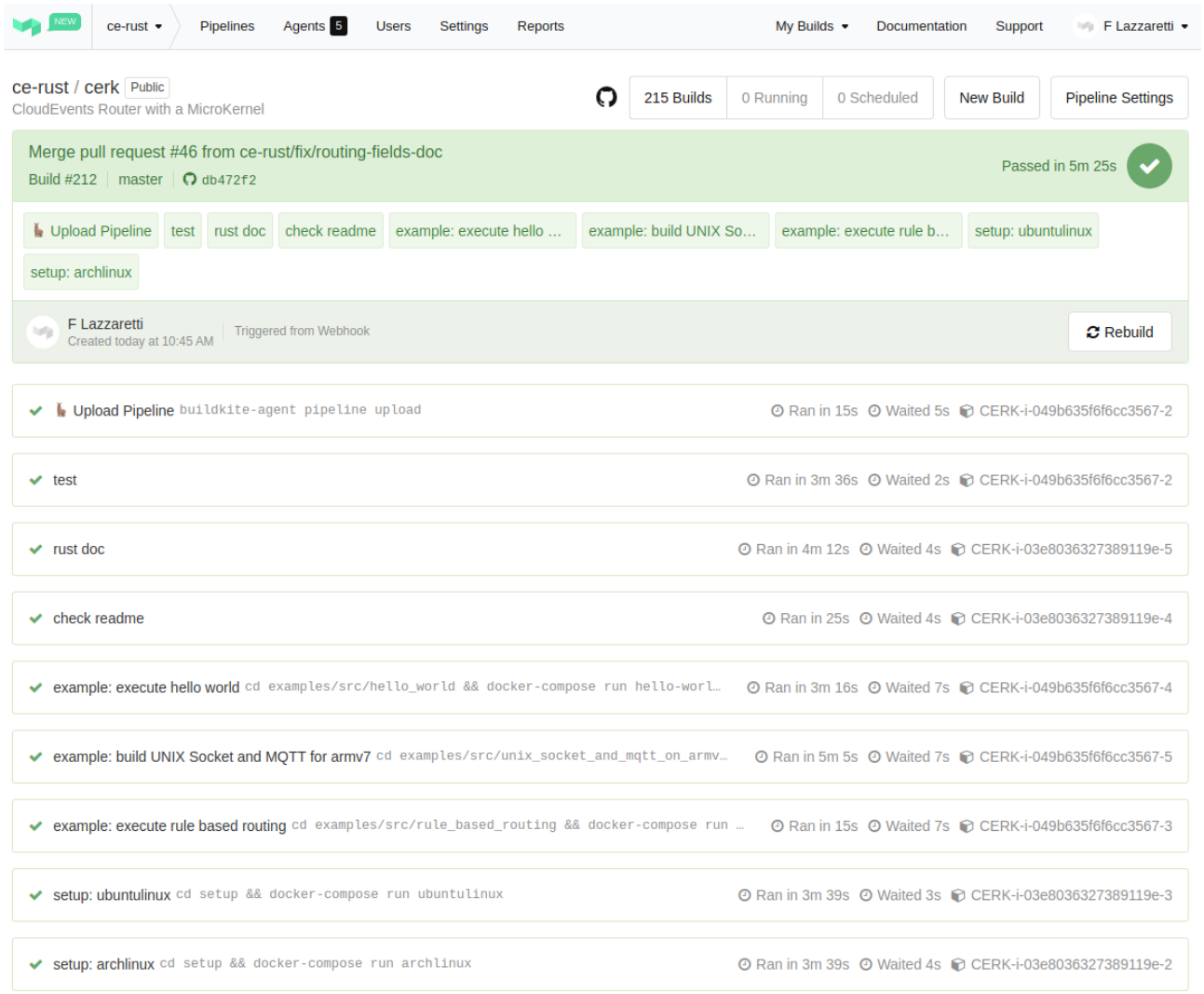

Figure 4.1: The CI of CERK. The figure shows a passed CI job on Buildkite. The job for the CERK CI contains 8 separate jobs.

5 Building Block View

In Figure 5.1 the high level decomposition is shown. In addition to the decomposition, the responsibilities are included in the diagram.

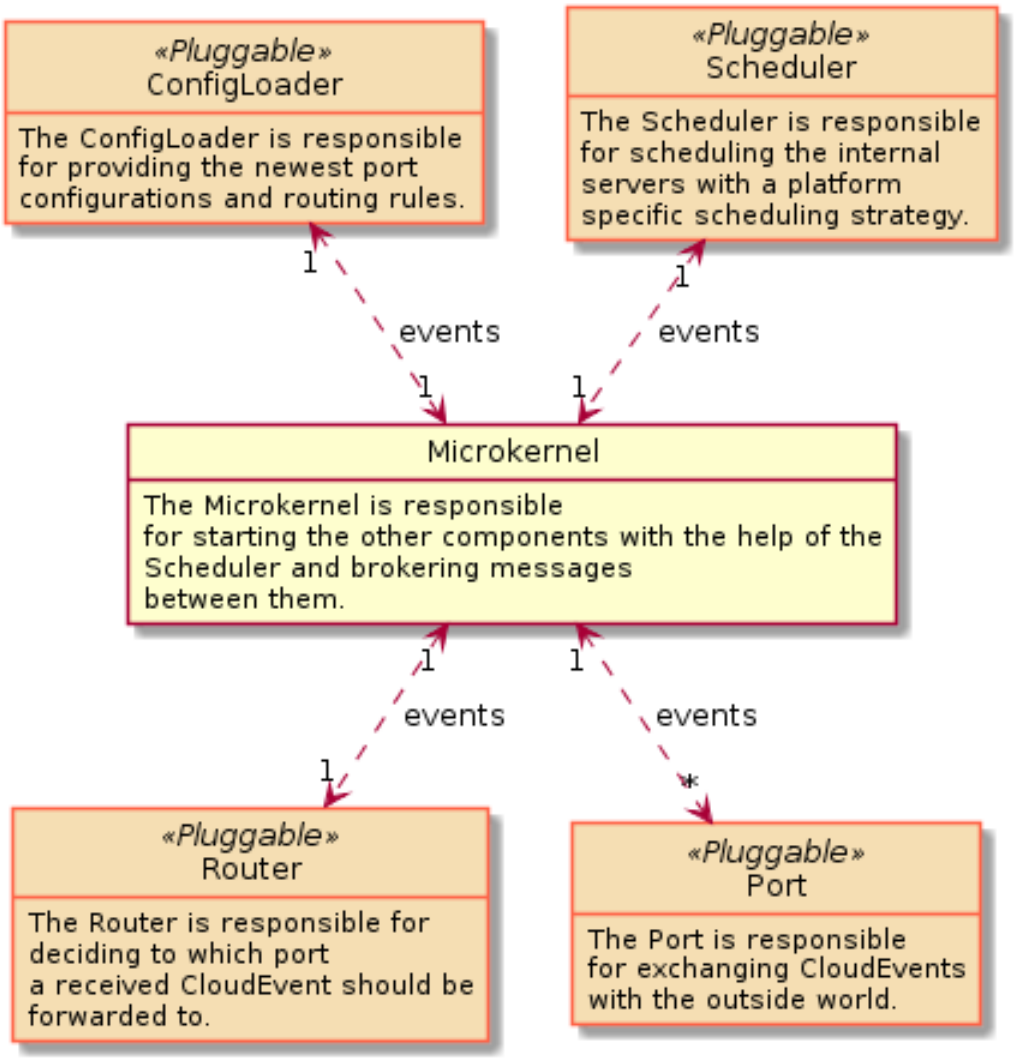

Figure 5.1: Building Blocks with their Responsibilities

5.1 Microkernel Events

Table 5.1 gives an overview over all events which are exchanged between the components.

Name	Description
Init	The Init event indicates to the receiver that it should start interacting with the outside world. The event is produced by the kernel when all components are scheduled.
ScheduleInternalServer	The ScheduleInternalServer event tells the Scheduler to schedule a new internal server. The event is produced by the kernel, one event for each component.
InernalServerScheduled	The InernalServerScheduled event indicates to the receiver that a new internal server was successfully scheduled. The event get produced by the scheduler, after a component was scheduled (because of a ScheduleInternalServer event), the receiver is the kernel.
IncommingCloudEvent	The IncommingCloudEvent event indicates to the receiver that a new CloudEvent has been received from the outside world. The event is produced by an input port and is sent to the kernel. The kernel sends the same Event to the router.
OutgoingCloudEvent	The OutgoingCloudEvent event indicates to the receiver that a CloudEvents has been routed and is ready to be forwarded to the outside world. The event is created by the router, send to the kernel and then to the output adapter. A single event is created for every message and receiver.
ConfigUpdated	The ConfigUpdated event indicates to the receiver that the config has changed and a configuration update should be applied. The event is produced by the router to the kernel and then to the component.
Batch	The Batch event can be used to make sure a collection of events are processed by the Microkernel in one batch to prevent race conditions.

Table 5.1: Microkernel Events; The events that are exchanged between the components

6 Runtime View

In this chapter we will describe the interactions of the building blocks during five runtime scenarios. They are sorted in the order of their occurrence during normal operations. Some scenarios can happen multiple times. Every section describes one runtime scenario, at the end all scenarios are shown combined in Figure 6.6.

6.1 Runtime Scenario 1: Bootstrap

Calling bootstrap function in Figure 6.1 initiates the start sequence of the application. The `bootstrap` function starts the scheduler, which then schedules the Microkernel.

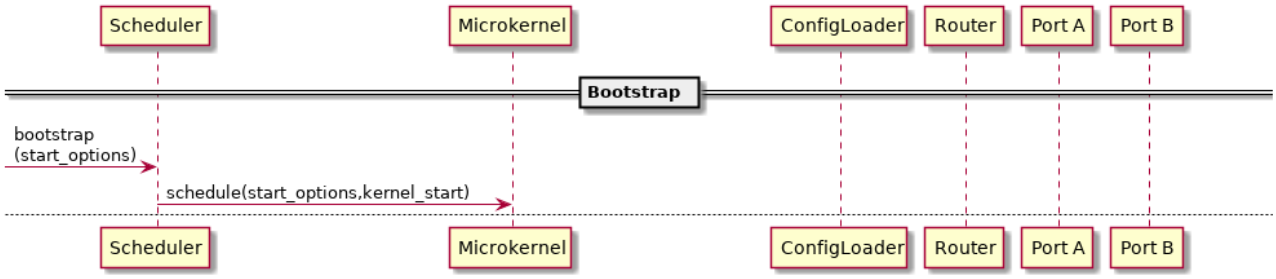

Figure 6.1: Runtime Scenario: Bootstrap

6.2 Runtime Scenario 2: Initialization

After the bootstrap sequence ended, the Microkernel schedules all the other building blocks with the help of the Scheduler (Figure 6.2). The list of the components to schedule is provided by the `start_options`.

For each component the Microkernel sends an event to the Scheduler. The Scheduler then schedules the component. A handle for the inbox channel of the Microkernel is provided as a parameter. With this handle the new component can send messages to the Microkernel. After the component is scheduled, the Scheduler sends an event back to the Microkernel to let it know that the component is scheduled. In this message, a handle for the inbox channel of the scheduled component is provided. With this handle the Microkernel can send messages to the component.

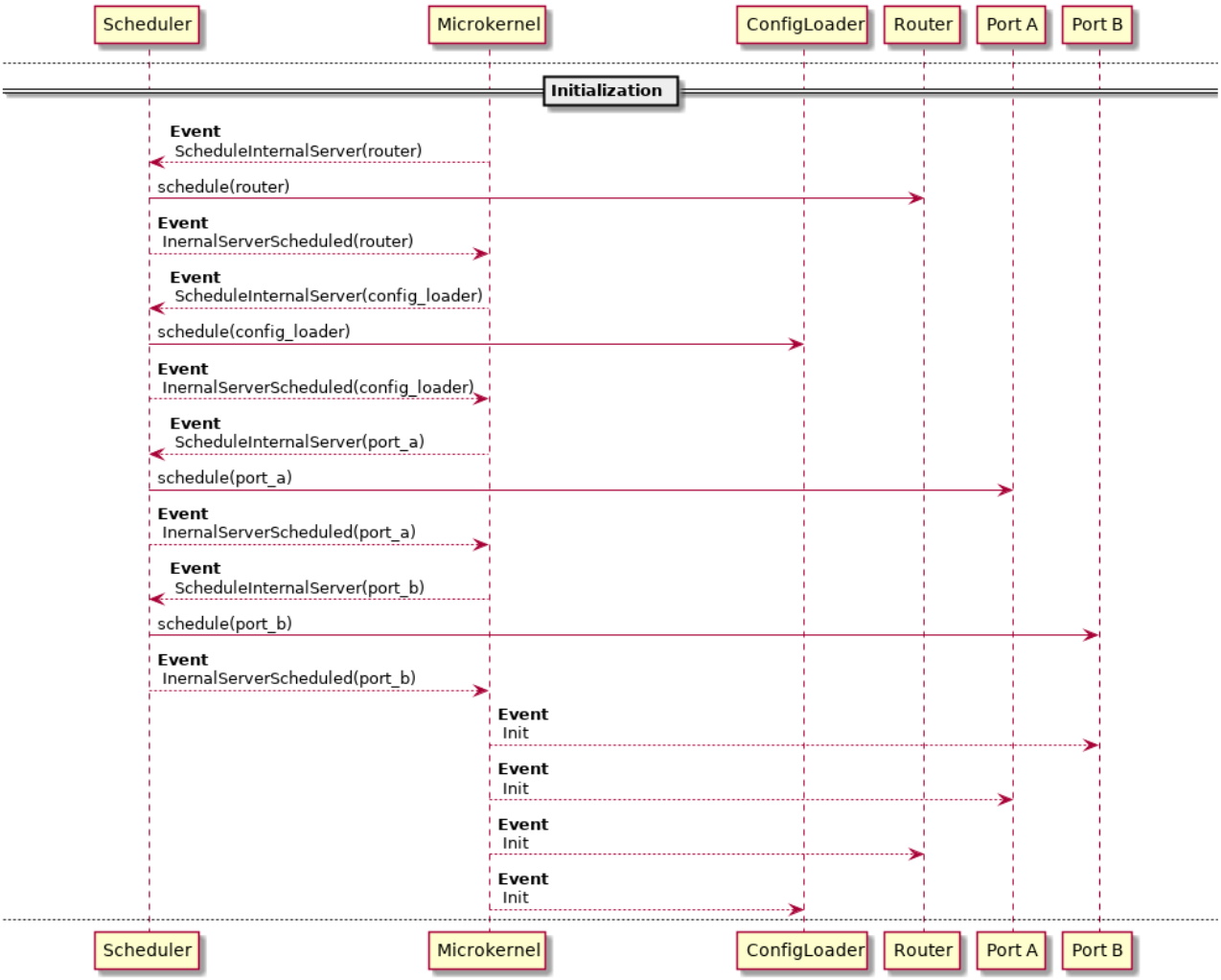

Figure 6.2: Runtime Scenario: Initialization

6.3 Runtime Scenario 3: Initial Configuration

In the scenario shown in Figure 6.3 the initialization has already happened. Now the ConfigLoader loads the configuration and sends it to the Microkernel. The Microkernel forwards the received configuration to the relevant component.

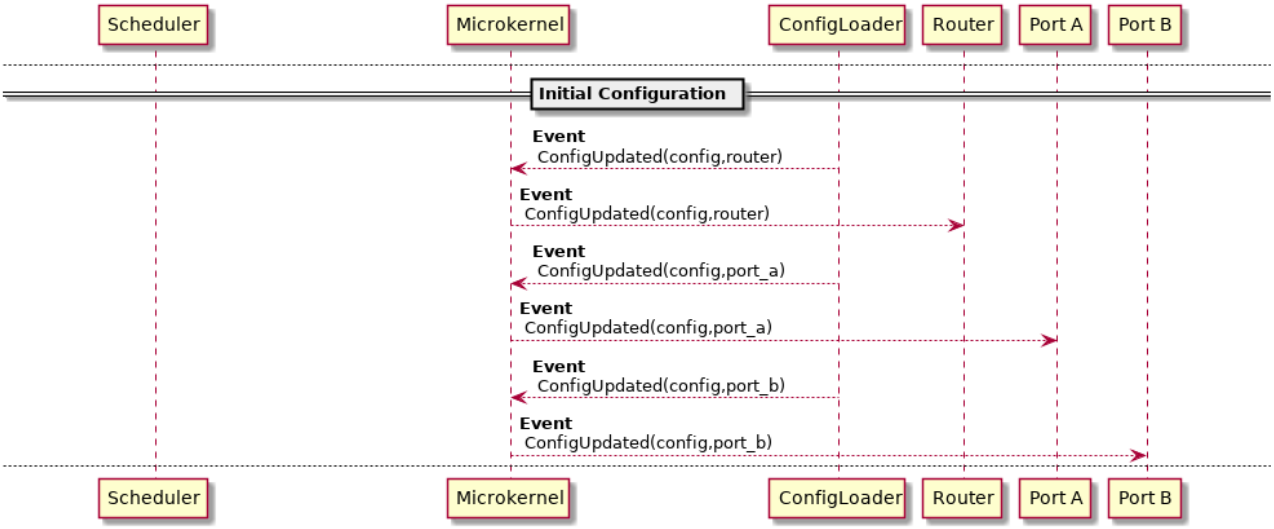

Figure 6.3: Runtime Scenario: Initial Configuration

6.4 Runtime Scenario 4: Routing

Figure 6.4 shows the routing scenario. One of the ports receive a CloudEvent from the outside world. This port deserializes and forwards the event to the Microkernel which forwards it again to the router. The Router decides, based on the configuration it received earlier, to which ports the event should be routed to. After the router has decided the event gets sent to the designated ports via the Microkernel. The ports serialize and forward the the event to the outside world.

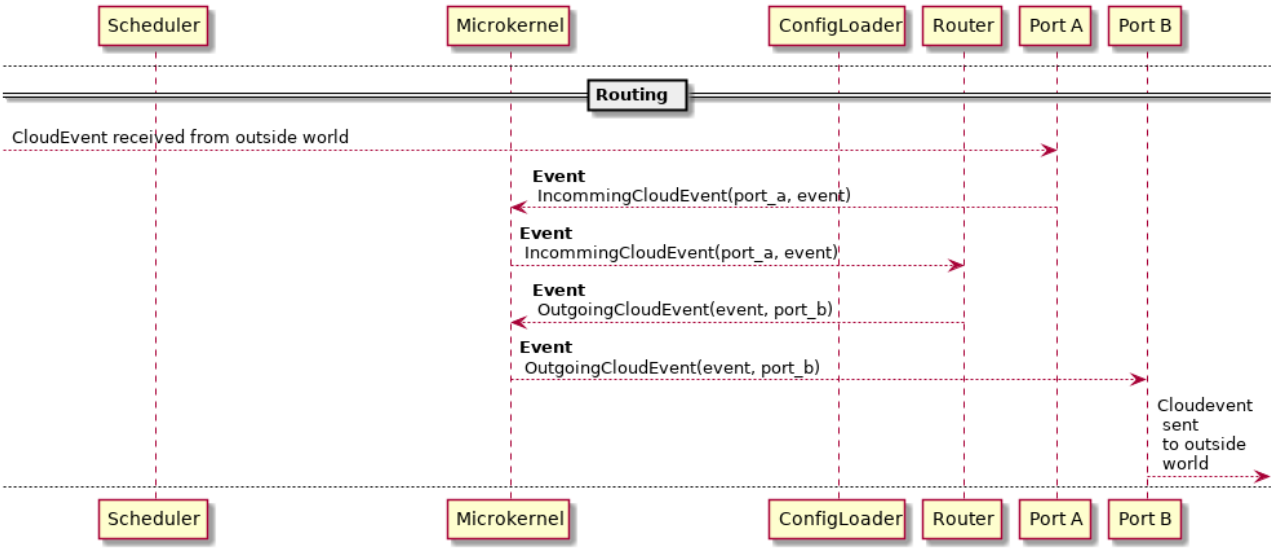

Figure 6.4: Runtime Scenario: Routing

6.5 Runtime Scenario 5: Configuration Update

The ConfigLoader can receive configuration updates during runtime from the outside world as shown in Figure 6.5. When the ConfigLoader receives an update, it sends the update to the affected port or router. The affected port or router reconfigures itself and acts according to the new configuration.

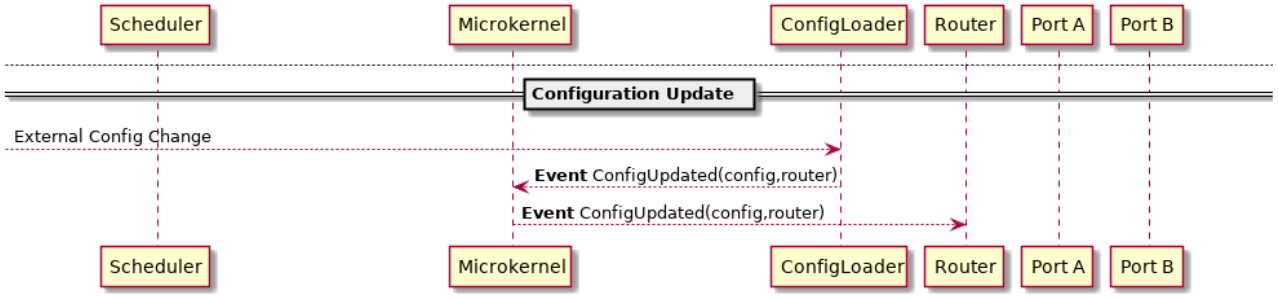

Figure 6.5: Runtime Scenario: Configuration Update

Figure 6.6: Runtime Scenarios showing the interaction between the components of CERK and the outside world

7 Deployment View

The detailed deployment of the CloudEvents Router is not part of the scope of this project. The only guardrails to keep in mind during development are the platforms CARU intents to use the router on.

7.1 Target Platforms at CARU

The following subsections provide an overview of the platforms used by CARU. The router should be able to run on the main processor of the CARU Smart Sensor and in the cloud. The co-processor of the CARU Smart Sensor is not the main target and the router may never be used on it.

7.1.1 CARU Smart Sensor

The CARU Smart Sensor is a product placed in the living environment of an elderly person to collect various room parameters, learn the behaviors of the inhabitant and alarm relatives or caretakers in case of deviations. Its user experience is optimized for elderly people.[22]
The device has two central processing units (CPUs) with two different operating systems on it.

Main Processor of the CARU Smart Sensor

The CARU Smart Sensor main processor has limited capabilities:

CPU Architecture ARM Cortex-A7 (armv7l)

CPU Cores 1

CPU Threads 1

CPU Clock 198 - 528 MHz

RAM 500 MB

Operating System Linux

Co-Processor of the CARU Smart Sensor

The CARU Smart Sensor low power co-processor has limited capabilities:

CPU Architecture ARM Cortex-M4 (Armv7E-M)

CPU Cores 1

CPU Threads 1

CPU Clock 80 MHz

RAM 160 KB

Operating System FreeRTOS

7.1.2 Virtual Machine in the Cloud

The virtual machines are hosted by AWS and have almost unlimited capabilities[23]:

CPU Architecture ARM, Intel, AMD

CPU Cores 1 - 224

CPU Threads 1 - 448

CPU Clock 2.3 GHz - 4.0 GHz

RAM 500 MB - 24 TB

Operating System Linux, Windows

8 Cross-Cutting Concepts

8.1 Architecture Patterns

8.1.1 Microkernel Pattern

”The Microkernel architectural pattern applies to software systems that must be able to adapt to changing system requirements. It seperates a minimal functional core from extended functionality and customer-specific parts. The Microkernel also serves as a socket for plugging in these extensions and coordinating their collaboration.”[24]

We took the pattern description from Pattern-Oriented Software Architecture as reference.

The Microkernel pattern defines five kinds of participating components: internal servers, external servers, adapters, clients and the Microkernel itself. It is the Microkernels task to provide the core mechanisms, offer communication facilities and manage the resources. Internal servers implement additional services and encapsulate some system specifics. External servers provide programming interfaces to its clients. Both, internal and external servers, collaborate with the Microkernel component. In addition to these internal components, the pattern also defines clients and adapters. Clients are separate applications that consume services from the Microkernel. An adapter is code that is part of the client, but is provided by the creators of the Microkernel. Nowadays, adapters are often called Software Development Kits (SKD). SDKs help the developers to interact with external server's APIs by adding an abstraction layer to it.[24]

The only significant difference between our interpretation of the Microkernel Pattern and the description from the authors is that we merged the internal and external servers. We call both of them internal servers because, in our context, we did not see any differences in how we interact with them. Treating them the same but naming them differently seemed unnecessary to us.

8.2 Implementation Rules

Rust has build in macros to check the code for different aspects while compiling. We used the `missing_docs` check as macro `#![deny(missing_docs)]`. This macro requires the programmer to add a documentation to every public element. If the programmer does not add a documentation, the code will not compile.[4]

The CI compiles the product on every Git push to GitHub. A successful CI pipeline run is required to finish a pull request. So it is not possible to merge a undocumented pice of library code to the master.

We also require that the code is well formatted. This is done with `cargo fmt`. Sadly this is not tested in the CI pipeline, yet. The pull request reviewer have to check this manually.

9 Architecture Decisions

9.1 ADR-001: MADR for Architecture Decision Documentation

Status accepted

Date 03.11.2019

9.1.1 Context and Problem Statement

We need to document our architecture in a way that a reader understands our thought process behind it.

9.1.2 Decision Drivers

lean The documentation should be lean.

simple The documentation should be easy to understand.

complete The documentation should reveal the thought process which lead to the decisions.

9.1.3 Considered Options

ADR Architecture Decision Record[25]

MADR Markdown Architectural Decision Records (version 2.1.2)[26]

Y-Statement Sustainable Architectural Decisions[27]

9.1.4 Decision Outcome

We decided to use Markdown Architectural Decision Records (MADR) because of the dedicated section about the comparison of the considered options, the good documentation and open license (CC0).

Because we are already using LaTeX for this document we will only use the structure of MADR and use it with LaTeX instead of Markdown.

Positive Consequences

- Our architecture decision documentation is structured.
- Our architecture decisions are reconstructable.
- Architecture documentation and decisions are in the same document.

Negative Consequences

- MADR related tooling not usable because we are using it with LaTeX.
- Architecture decisions and code in different repositories.[28, 29]
- The LaTeX format is not as easy to publish as a webpage as Markdown.

9.1.5 Pros and Cons of the Options

Option MADR

good considers context, decision and consequences

good considers and compares alternative solutions

good specific instructions (GitHub repository[26])

good open license (CC0)

Option Architecture Decision Record (ARD)

good considers context, decision and consequences

good good documentation (blog post[25])

bad lacks consideration of alternative solutions

bad no license

Option Y-Statement

good considers context, decision and consequences

good considers alternative solutions

bad distributed documentation (article[27], presentation slides[30], third party summary[31])

bad too compact (in the context of this thesis)

bad no license

9.1.6 Links

- http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions/
- https://github.com/adr/madr/tree/2.1.2/
- https://www.infoq.com/articles/sustainable-architectural-design-decisions/

9.2 ADR-002: Rust as the Programming Language

Status accepted

Date 26.10.2019

9.2.1 Context and Problem Statement

We need to select a programming language for the implementation of our solution.

9.2.2 Decision Drivers

customer The customer has a strong preference for Rust because he wants to gain experience with it.[2]

portability The programming language should support multiple platforms. Especially support of resource restricted platforms like the Cortex-M4 processor with FreeRTOS is a wish of the customer. (see NFR-2)

footprint The programs produced by the programming language should have a low resource footprint because of the resource constraint target environments.[2]

stability The produced program should run for a long time without a memory leaks or crashes.

9.2.3 Considered Options

Rust The Rust programming language[32]

C/C++ The C and C++ programming languages[33, 34]

Go The Go programming language[35]

Python The Python programming language[36]

9.2.4 Decision Outcome

We decided to use Rust because the programs it creates have a small footprint and can run on various platforms (including bare metal and FreeRTOS[37, 38]). It's unique "ownership model" and the preferences of the customer gave it the edge over C/C++.

Positive Consequences

- Rust has a small footprint because it has no runtime or garbage collector.[39]
- "Rust's rich type system and ownership model guarantee memory-safety and thread-safety"[32]
- Rust can be compiled for many platforms, including bare metal and FreeRTOS.[37, 38]
- Rust is preferred by the customer.[2]

Negative Consequences

- Rust is new for us and we have no experience with it.
- Rust is a low-level programming language and we have not a lot of experience with low-level programming languages.
- Rust is a young programming language with a smaller ecosystem than others options.

9.2.5 Pros and Cons of the Options

Option Rust

good statically typed

good the "Rust ownership model" guarantees memory-safety and thread-safety[32, 40]

good can be compiled to be run without an operating system[37]

bad young language with small ecosystem (when compared to C/C++/Python)

bad no experience in the team

bad the "Rust ownership model" is unique and has a steep learning curve

Option C/C++

good statically typed

good can be compiled to be run without an operating system

good mature language with big ecosystem

good some experience in the team

good used by the customer

bad easy to make mistakes (memory leaks, stale pointers, ...)[41, 42]

bad old language with lots of legacy because of its backward-compatibility

Option Go

good statically typed

good simple multi-processing model

bad garbage collected

bad requires an operating system to run

bad young language with small ecosystem (when compared to C/C++/Python)

bad no experience in the team

Option Python

good can be run without an operating system (with some limitations)[43]

good experience in the team

good widely used by the customer

bad dynamically typed

bad interpreted language

bad garbage collected

9.2.6 Links

- https://www.rust-lang.org/
- https://isocpp.org/
- https://golang.org/
- https://www.python.org/

9.3 ADR-003: Use of the Microkernel Pattern

Status accepted

Date 26.10.2019

9.3.1 Context and Problem Statement

We need to select an architecture pattern that allows our solution to support multiple platforms, with and without operating system, and a mechanism to add adapters.

9.3.2 Decision Drivers

extendability It should be easy to add new adapters. (see NFR-1)

portability The pattern should allow the implementation of platform-specific service. (see NFR-2)

efficiency The pattern should have as little overhead as possible.

9.3.3 Considered Options

Microkernel Microkernel pattern (see pattern summary in subsection 8.1.1)

Microservices Microservices pattern[44]

Plugin Plugin pattern[45, p. 499-503]

9.3.4 Decision Outcome

We decided to use the Microkernel pattern because it requires no inter-process communication and we don't need the scalability of the Microservice pattern.

The decision was fixed, after the first running Microkernel prototype were running.[1]

Positive Consequences

- A Microkernel can be easily ported to different platforms.
- The overhead is low because everything is in one process.
- Core functionality can be shared between platform-specific implementations.

Negative Consequences

- A Microkernel is harder to scale than Microservices.

[1] https://github.com/ce-rust/architecture-prototype/commit/28db2968e01eaddb5b8a349f025387d13d178a9a

9.3.5 Pros and Cons of the Options

Option Microkernel

good allows platform-specific implementations of services

good can run services in one process

good manages communication between services

bad limited scaling possibilities

Option Microservices

good allows platform-specific implementations of services

good different parts of the system can potentially run on different platforms at the same time

good scalability beyond a single machines

bad services are separate processes by definition

bad overhead of inter-process communication

Option Plugin

good allows dynamic loading of different implementations

good separate binary for program and plugins

bad depends on operating system

bad platform-dependent plugin format (so, dylib, dll)[46]

bad dynamic linking hell[47]

bad `unsafe` Rust code (the Rust compiler can not give it's usual safety guarantees. because the plugin is dynamically loaded)[48]

9.3.6 Links

- see pattern summary in subsection 8.1.1
- https://martinfowler.com/articles/microservices.html

9.4 ADR-004: Use of the Threading Model

Status accepted

Date 26.10.2019

9.4.1 Context and Problem Statement

We need a way to run multiple blocking operations concurrently to be able to implement the different adapters.

9.4.2 Decision Drivers

overhead The overhead of running blocking operations concurrently should be as low as possible.

compatibility We want to reuse existing libraries to implement the adapters.

performance We want be be informed as fast as possible when new data is available.

portability We want to be able to support multiple platforms.

9.4.3 Considered Options

Async Rust runs an event loop and schedules different parts of the program concurrently.[49]

Polling We check regularly if new data is available.

Threading Rust tells the platform to schedule different parts of the program concurrently.[50]

9.4.4 Decision Outcome

We decided to use the threading model because it allows us to outsource the scheduling to the platform and leverage the full library ecosystem of Rust. There is also the possibility to run non input/output-bound tasks in parallel on platform which support it.

Because the implementation of threads is platform-specific, our microkernel will provide an abstraction (called Internal Server) for it which needs to be implemented for each platform.

The decision was fixed, after the first running Microkernel prototype with a threading model was implemented and did run successfully.[2]

Positive Consequences

- We can reuse existing libraries.
- The platform takes care of scheduling and wakes threads (Internal Servers) automatically when new data is available.

[2] https://github.com/ce-rust/architecture-prototype/commit/28db2968e01eaddb5b8a349f025387d13d178a9a

Negative Consequences

- There is an overhead for switching between threads.
- The implementation of the threading model differs from platform to platform.
- Our microkernel needs to provide the interface for threads (Internal Servers) that can be implemented for the different platforms.

9.4.5 Pros and Cons of the Options

Option Async

good low overhead because no context switching required[49]

good high performance for io-bound tasks[49]

bad not supported on bare metal[51]

bad only libraries built for async can be used

bad async was a beta feature when we started with the thesis and not a lot of libraries support it. It is stable since Rust 1.39.0.[52, 53]

bad no parallelism possible

Option Polling

good platform independent

good any library can be used

bad very high overhead if we poll often

bad very low performance if we poll seldomly

bad no parallelism

Option Threading

good most platforms implement the threading model

good any library can be used

good good performance because the platform schedules the thread as soon new data is available

good good scalability on platform with parallel processing capabilities

bad implementation depends on platform

bad overhead because of the switching between thread contexts

9.4.6 Links

- https://rust-lang.github.io/async-book/
- https://doc.rust-lang.org/std/thread/

9.5 ADR-005: Use of the Broker Pattern with Channels

Status accepted

Date 16.10.2019

9.5.1 Context and Problem Statement

Because we decided that we will be using a threading based approach, we need to define how the microkernel's Internal Servers (threads) communicate with each other.

9.5.2 Decision Drivers

idiomaticity The way the Internal Servers communicate should be idiomatic for Rust

overhead The communication should have a small overhead.

9.5.3 Considered Options

Broker with channels a central broker which receives and forwards data over channels[54, 55]

Broker with invocations a central broker which coordinates invocations between Internal Servers[56]

Mesh with channels Internal Servers are directly connected to each other over channels[55]

Mesh with invocations Internal Servers directly invoke each other

9.5.4 Decision Outcome

We decided to use a broker with channels for the communication between Internal Servers because it is the most idiomatic way for Rust.

The implementation of the channel is, like the threading implementation, platform-specific, our microkernel will provide an abstraction for channels (called Channel) which needs to be implemented for each platform.

Positive Consequences

- Channels are the idiomatic way for sharing data between threads in Rust.[57]
- No explicit locking needed.
- Decoupling of the Internal Servers (they only need to know the channel to the broker).

Negative Consequences

- The broker can become a bottleneck.
- Overhead of communicating over the broker.
- The implementation of channels differs from platform to platform.
- Our microkernel needs to provide the interface for channels that can be implemented for the different platforms.

9.5.5 Pros and Cons of the Options

Option Broker with Channels

good channels are idiomatic for Rust[57]

good channels don't need explicit locking

good decoupling (only the channel to the broker needs to be known)

bad communication over the broker is overhead

bad broker can become the bottleneck

bad channels are platform specific

bad request/response scenarios are harder to implement because it is message based

Option Broker with Invocations

good decoupling (only the reference to the broker needs to be known)

good invocations make request/response scenarios easier to implement than channels

bad Internal Servers share reference of broker[58]

bad access to the broker needs to be explicitly secured with locks[59]

bad communication over the broker is overhead

bad the broker can become the bottleneck

bad locks are platform specific

Option Mesh with Channels

good channels are idiomatic for Rust[57]

good channels don't need explicit locking

good no overhead because of indirect communication

bad coupling (channels to all other Internal Servers need to be known)

bad channels are platform specific

bad request/response scenarios are harder to implement because it is message based

Option Mesh with Invocations

good invocations make request/response easy

good no overhead because of indirect communication

bad coupling (references of all other Internal Servers need to be known)

bad Internal Servers share reference to each other[58]

bad access to the Internal Servers needs to be explicitly secured with locks[59]

bad locks are platform specific

9.5.6 Links

- https://doc.rust-lang.org/std/sync/index.html

10 Quality Requirements

section 10.1 describes the quality requirements with help of the quality tree. The section 10.2 add more details to the non-functional requirements.

10.1 Quality Scenarios and Quality Tree

Figure 10.1 shows our quality tree with the quality scenarios as leafs. The quality tree is included for completeness and is not scope defining. The quality scenarios should be kept in mind while designing the product.

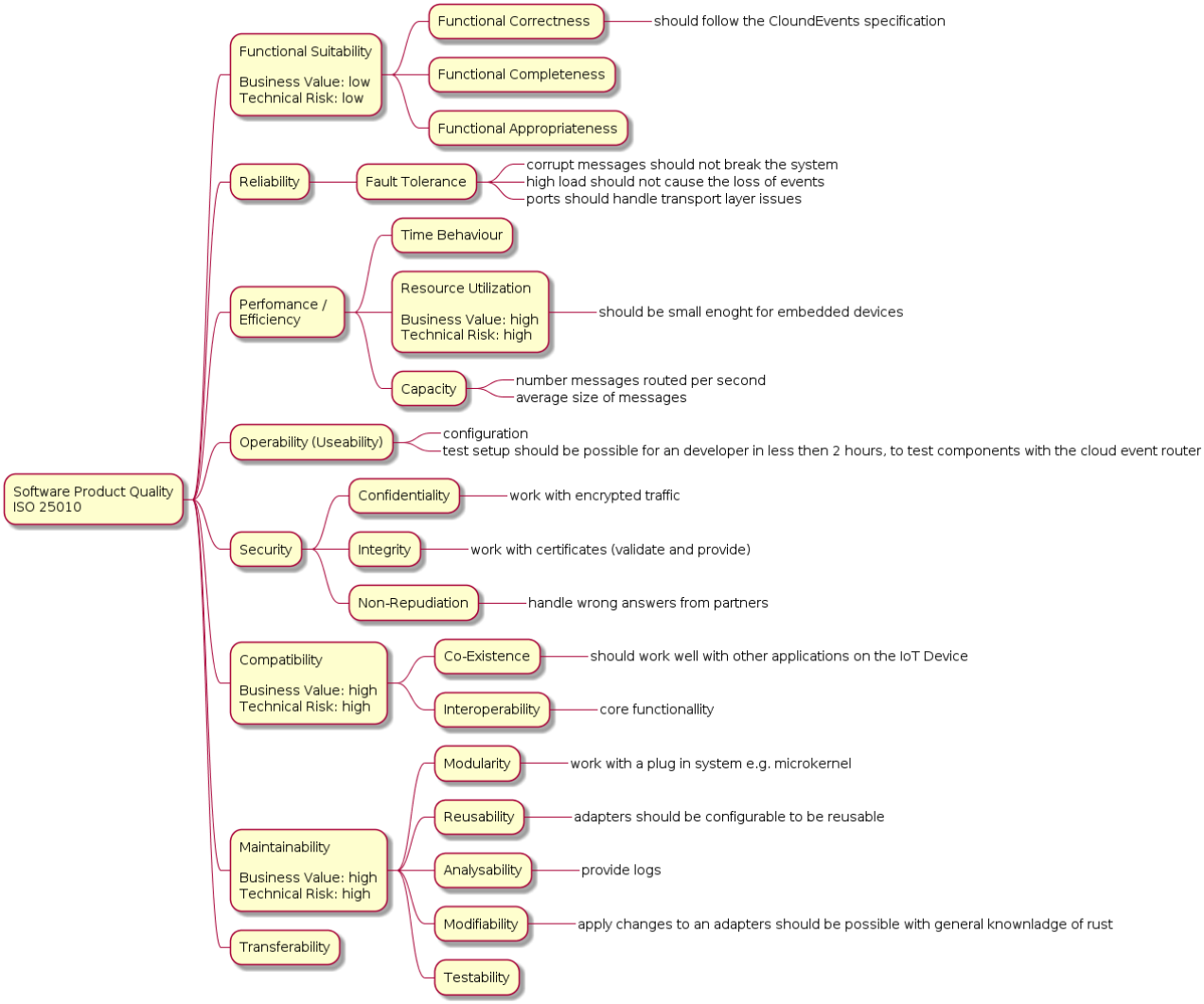

Figure 10.1: Quality Tree according to arc42 and ISO 25010 [60, 61]

10.2 Landing Zones

The primary landing zones add more details to the non-functional requirements defined in section 1.2. There are also secondary landing zones which do not have a corresponding non-functional requirement in section 1.2. These landing zones are less important than the ones with a corresponding non-functional requirement. Table 10.1 lists our landing zones with their minimum, target and outstanding result.

No	NFR	Description	minimum	target	outstanding
LZ-1	NFR-2	Supported operating systems	Ubuntu Linux	Embedded Linux	FreeRTOS
LZ-2	NFR-3	Easy findable and followable documentation to setup the project and run it.	README document exists	Documentation exist and contains clear setup and run documentation.	Multiple tutorials for different adapter setups are provided.
LZ-3	-	Events processed on a normal computer	5/s [1]	500/s [2]	1500/s [3]
LZ-4	-	Average event size	1 KByte	64 KByte [4]	64 KByte

Table 10.1: Landing Zones

[1] Current number of events processed per second on the CARU Smart Sensor
[2] Estimated number of events processed in the cloud by the end of the year
[3] Expected number of events processed in the cloud per second for 1000 devices
[4] Maximal size the CloudEvents specification allows [62]

11 Risks and Technical Debts

In this chapter, the known and most critical technical debts and risks are listed. In addition to the following sections also a list of issues is maintained on GitHub: https://github.com/ce-rust/cerk/issues.

11.1 Single Unit of Mitigation

At the moment no advanced error mitigation concept is implemented; the whole router is one ”Unit of Mitigation”. This means the whole router shuts down if an error occurs in any component.

11.2 Limited QoS Support

Another desirable feature that is not implemented yet is the support of QoS values other than 0 (”at most once’)’. An option to define the QoS like the MQTT protocol does would make sense for the router.[13]

List of Figures

List of Tables

Glossary

There are some overlapping glossary entries between the documents of this thesis. Each document contains only the glossary entries which are referred in that document.

"ownership model" The "ownership model" is the most unique feature of Rust. In Rust memory is allocated and freed explicit, but in contrast to C or other low level programming languages without the need of an garbage collector, Rust enforces a set of rules on this actions, to ensure memory safety."[4]. 11, 15, 54, see Rust

LaTeX "LaTeX, which is pronounced «Lah-tech» or «Lay-tech» (to rhyme with «blech» or «Bertolt Brecht»), is a document preparation system for high-quality typesetting. It is most often used for medium-to-large technical or scientific documents but it can be used for almost any form of publishing."[63]
LaTeXis distributed under the LaTeX project public license. It is a free software license.[64]. 31, 32, 51

Academic Free License Academic Free License is a open-source license provided by the Open Source Initiative (OSI). It is not a copyleft license.[65]. 53, see copyleft, OSI & open-source

AMQP "AMQP, the Advanced Message Queuing Protocol, is an open standard for interoperable messaging at the wire protocol level. Message brokers that implement AMQP can communicate with each other and exchange messages without the need for adapters or bridges. An AMQP message broker can provide first-class native language bindings for multiple programming languages; so AMQP-based messaging is a good choice for cross-platform compatibility across the Enterprise. "[66]
It is standardized by OASIS.[67]. 55

Apache 2 Apache 2 is an open-source license from the Apache Software Foundation. The license allow modifications and redistribution of the software. It has no copyleft.[11]. 3, 11, 16, 53, see copyleft & open-source

API Is a interface to communicate from one system to an other[68]. 55

arc42 arc42 is a template for documentation and communication of a system or software. It focus on lean and agile development approaches.[69] The template is available in available in different format, one of them is LaTeX.[70] It is open-source licensed under the Creative Commons Attribution-ShareAlike 4.0 International License.[71]. 2, 43, see LaTeX

AWS Amazon Web Services (AWS) is a cloud platform provided by Amazon[72]. 55

Buildkite Buildkite is a service for coordinating buildpipelines which get executed on own infrastructure. They provide a build agent that can easily be deployed to AWS and other platforms. Buildkite is well integrated with GitHub.[21]. 16, 17, see AWS, CI & GitHub

bytecode "Bytecode is program code that has been compiled from source code into low-level code designed for a software interpreter. It may be executed by a virtual machine (such as a Java Virtual Machine (JVM)) or further compiled into machine code, which is recognized by the processor."[73]. see Java & JVM

C C is a programming language developed by Dennis Ritchie. C++ is a superset of C. Later it was standardized by AMSI and ISO.[74]. 33, 34, 51–53, see C++

C++ C++ is a general-purpose programming language designed by Bjarne Stroustrup. Later it was standardized by AMSI and ISO.[74]. 33, 34, 51

C++ Bindings C++ Bindings are wrappers for a programming language to use C++ function in it.[75]. see C++

Cargo Cargo is the default package manager for Rust. It is really simple, compiles code, installs packages, runs tests, formats source files and generates the documentation. Cargo can be extended with additional subcommands by installing packages via Cargo itself.. 54, see Rust

CARU Smart Sensor The CARU Smart Sensor is a product placed in the living environment of an elderly person to collect various room parameters, learn the behaviors of the inhabitant and alarm relatives or caretakers in case of deviations. Its user experience is optimized for elderly people.[22]
The device has two CPUs with two different operating systems on it.. 3, 5–8, 27, 44, see FreeRTOS & Linux

CC0 "No Rights Reserved" License[76]. 31, 32

CI "Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early."[77]. 55

Cloud Native "Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach."[78]. 52, see CNCF

CloudEvents The CloudEvents standard is a standard from the Cloud Native Computing Foundation (CNCF). The goal of the standard is to "describe event data in common formats to provide interoperability across services, platforms and systems."[79] The standard reached version 1.0 on the 24th of October 2019.. 5–8, 13, 14, 20, 23, 27, 44, 52, 53, see CNCF

CNCF The Cloud Native Computing Foundation is part of the Linux Foundation. The goal of the foundation is to empower the usage of the cloud by providing standards and open-source projects, the use the term Cloud Native.[78] The foundation host project like Kubernetes[80] and Prometheus[81]. Apple, Microsoft, Google and Amazone Web Services are only a couple of the most famous partners.[82]. 55, see Cloud Native

copyleft Copyleft is a method for making a program including all versions and modifications free.[83]. 11, 16, 51, 52

DSL A custom language for a specific purpose. "It provides ways to work with exactly those abstractions you need in the respective context."[84]. 55

EKS Amazon Elastic Kubernetes Service is a managed Kuberentes cluster as a service provided by Amazon.[85]. 55, see Kubernetes

FaaS "Function as a service (FaaS) is a cloud computing model that enables users to develop applications and deploy functionalities without maintaining a server, increasing process efficiency. The concept behind FaaS is serverless computing and architecture, meaning the developer does not have to take server operations into consideration, as they are hosted externally. This is typically utilized when creating microservices such as web applications, data processors, chatbots and IT automation."[86]. 55

FreeRTOS FreeRTOS is a Real Time Operating System (RTOS) written in C distrubuted under the MIT license. The kernel binary is around 6KB to 12KB.[87]. 27, 33, 44, see RTOS

Git Git is an open-source distributed version control system. The tool was created in 2005 by the Linux community.[9]. 11, 16, 29, 52

GitHub GitHub is a source code management and collaboration platform based on Git. It is a well known platform for open-source code and was acquired in 2018 by Microsoft.[10]. 3, 11, 16, 29, 32, 45, 51, see Git

GNU 2 The GNU 2 or GNU General Public License v2 is is a copyleft license of the Free Software Foundation, the latest version is version 3.[88]. 52, 53, see GNU 3, copyleft & open-source

GNU 3 The GNU 3 or GNU General Public License v3 is a copyleft license of the Free Software Foundation, the latest version is version 3.[89]. 52, 53, see GNU 2, copyleft & open-source

Go Go is a statically typed general purpose programming language.[90] The language was developed by Google an is now open-sourced under the BSD-style license.[91]. 33, 34

HTTP "The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers"[92]. 55

Industry 4.0 "The term "Industry 4.0" refers to the transformation of the manufacturing sector by digital technologies, such as the internet of things (IoT), artificial intelligence (AI), machine learning, robotics, 3-D printing, visualization, virtual/augmented reality and analytics."[93]. see IoT

IoT "The internet of Things, or "IoT" for short, is about extending the power of the internet beyond computers and smartphones to a whole range of other things, processes and environments. Those "connected" things are used to gather information, send information back, or both."[94]. 53, 55

Java "Java is a high-level programming language developed by Sun Microsystems."[95] Java code is compiled to java bytecode and executed with a JVM. 51, 54, 55, see JVM & bytecode

JavaScript "JavaScript is a programming language commonly used in web development. It was originally developed by Netscape as a means to add dynamic and interactive elements to websites."[96] It is a scripting language based on the ECMAScript standard.[97]. 3, 53, 55

JDK The Java Develompen Kit is a combination of the Java runtime, the compile and other tooling that is needed for the development of Java applications.[98]. 55

JSON "JSON is a syntax for serializing objects, arrays, numbers, strings, booleans, and null. It is based upon JavaScript syntax but is distinct from it: some JavaScript is not JSON."[15]. 14, 55

JVM The Java Virtual Machine is a virtual machine running a java program. It implements a concrete commands for the abstract java specification. It executes the java bytecode and manages the memory.[99]. 55, see Java

Knative Knative is a Function as a Service (FaaS) platform on top of Kubernetes.. see Kubernetes & FaaS

Knative Eventing Knative Eventing is a set of loosely coupled services that route CloudEvents from producers to interested consumers. It is tightly integrated into Kubernetes.[100]. see Knative & Kubernetes

Kubernetes Kubernetes is an open-source container orchestration platform hosted by CNCF.[101]. 52, 53

Linux Linux is a operating system written in C with a microkernel architecture and licenced under the GNU 2 license.[74, 102, 103]. 5, 11, 52

Markdown "Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convert it to structurally valid XHTML (or HTML)."[104]. 31, 32

Microkernel "The Microkernel architectural pattern applies to software systems that must be able to adapt to changing system requirements. It seperates a minimal functional core from extended functionality and customer-specific parts. The Microkernel also serves as a socket for plugging in these extensions and coordinating their collaboration."[24]. 3, 4, 15, 20–23, 29, 36, 38, 49

MIT license The MIT License is a open-source license originally created by the Massachusetts Institute of Technology and documented by the OSI.[105]. 52, 53, see OSI & open-source

MQTT MQTT is a lightweight publish/subscribe messaging transport protocol for machine to machine communication.[12, 13]. 13, 55

MVP A product is at the state of a Minimum Viable Product when it gives the user an additional value, but as little extra feature as possible to get a good feedback loop for additional features[106]. 55

Node.js Node.js is an open-source JavaScript runtime build on top of the Google Chrome V8 JavaScript engine.[107]. see JavaScript

open-source "Generally, Open Source software is software that can be freely accessed, used, changed, and shared (in modified or unmodified form) by anyone. Open source software is made by many people, and distributed under licenses that comply with the Open Source Definition.
The internationally recognized Open Source Definition provides ten criteria that must be met for any software license, and the software distributed under that license, to be labeled "Open Source software.""[108]
GNU 2, GNU 3, Apache 2, Academic Free License and MIT license are all open-source licenses which have been approved by OSI. 9, 11, 16, 51–53, see OSI

OSI OSI is a California public non-profit corporation with the goal of educate about open source.[109]. 55, see open-source

pair-programming Pair-programming is a style of programming where two programmers working collaboratively on the same code, design or test at the same time. The desired goal behind this strategy is to have better code quality and better know-how distribution.[19, 20]. 16

PlantUML PlantUML is a tool to generate different Unified Modeling Language (UML)) and non UML diagrams. The diagrams are generated by writing with the PlantUML Language. The tool is written in Java and has various output formats, one of the im JPGE.[110]. see UML

pull request "Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch."[111]. 3, 16, 29, see Git & GitHub

Python Python is a high-level general purpose programming language; It is used as scripting language, for Web application and many other things[112]. 5, 6, 33–35

QoS Quality of Service is defined in the term of telecommunication as "Totality of characteristics of a telecommunications service that bear on its ability to satisfy stated and implied needs of the user of the service."[113]. 55

RTOS Real Time Operating Systems are a type of operating systems where "The scheduler ... is designed to provide a predictable (normally described as deterministic) execution pattern. This is particularly of interest to embedded systems as embedded systems often have real time requirements. A real time requirements is one that specifies that the embedded system must respond to a certain event within a strictly defined time (the deadline). A guarantee to meet real time requirements can only be made if the behaviour of the operating system's scheduler can be predicted (and is therefore deterministic)."[114]. 56

Rust Rust is a low-level programming language with guaranteed thread and memory safety. It achieves that with its unique "ownership model". The goal is to build safe and reliable software.[4, 5]
Rust is used by Mozilla Firefox, Dropbox and many others.[6, 7]
In 2019, Rust has also been elected in the well-known survey of Stack Overflow for being the "most loved" programming language.[8]. 3, 4, 11, 15, 33, 34, 37–41, 51, 52, 54, see "ownership model" & Stack Overflow

rustc rustc is the Rust compiler provided by the Rust Team. It can be used directly (`rustc`) or with Cargo (`cargo build`).[115]. see Rust & Cargo

rustup rustup is a tool to install different versions of Rust, like stable, beta and nightly, and for different compilation targets.
It is the recommended way for developers to install Rust.[116]. see Rust & Cargo

SaaS Software as a service (SaaS) is a software distribution model in which a third-party provider hosts applications and makes them available to customers over the Internet.[117]. 56

SDK Software Development Kits are libraries which ease the access to Application Programming Interface (API)s. 56

Serial Interface "A serial interface is a communication interface between two digital systems that transmits data as a series of voltage pulses down a wire."[17]. 14

Serverless Framework The Serverless Framework is a FaaS abstraction layer.. 54, see FaaS

SNS Amazon Simple Notification Service (SNS) is a fully managed publish/subscribe messaging system. It allows the filtering based on meta-data sent with the messages.[118]. 56

SQS Amazon Simple Queue Service (SQS) is a fully managed message queuing service from Amazon with first-in, first-out (FIFO) support[119]. 56

Stack Overflow Stack Overflow is primary a Q&A forum for coding related problems.. 11, 15, 54

STOMP A Simple Text Oriented Message Protocol, designed for asynchronous message passing[120]. 56

TCP The Transmission Control Protocol (TCP) is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks.[121]. 56

UML UML is a standard for diagramming notation. It was created by Booch and Rumbaugh in 1994.[122]. 54, 56

UNIX socket "The `AF_UNIX` socket family is used to communicate between processes on the same machine efficiently."[16] . 3, 5–8, 14, 54, see UNIX

WAMP "WAMP is a routed protocol that provides two messaging patterns: Publish & Subscribe and routed Remote Procedure Calls. It is intended to connect application components in distributed applications. WAMP uses WebSocket as its default transport, but can be transmitted via any other protocol that allows for ordered, reliable, bi-directional, and message-oriented communications."[123]. 56

XML XML is a syntax for documents. "XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure."[124]. 56

Acronyms

ADR ARD (Architecture Decision Record). 31, 32, Glossary: ADR

AMQP Advanced Message Queuing Protocol[67]. Glossary: AMQP

AMSI American National Standards Institute. 51

API Application Programming Interface. 29, Glossary: API

AWS Amazon Web Services[72]. 13, 16, 28, 51, Glossary: AWS

CERK CloudEvents Router with a MicroKernel. 5

CI Continuous Integration. 16, 17, 29, 47, Glossary: CI

CNCF . 52, Glossary: CNCF

CPU central processing unit. 27, 52

DSL Domain Specific Language. Glossary: DSL

EKS Elasitc Kubernetes Service . Glossary: EKS

FaaS Function as a Service. 53, 54, Glossary: FaaS

FR Functional Requirements. 5

HTTP HyperText Transfer Protocol. Glossary: HTTP

IoT Internet of Things. 13, Glossary: IoT

ISO International Organization for Standardization. 51

JDK Java Development Kit. Glossary: JDK

JSON JavaScript Object Notation. 3, 14, Glossary: JSON

JVM Java Virtual Machine. 51, 53, Glossary: JVM

KB Kilobytes, equal to 1000 Bytes. 27, 52

MADR Markdown Architectural Decision Records[26]. 31, 32

MB Megabytes, equal to 1000 Kilobytes. 27

MQTT Message Queuing Telemetry Transport[13]. 3, 5, 7, 8, 13, 45, Glossary: MQTT

MVP Minimum Viable Product[106]. Glossary: MVP

NFR Non-Functional Requirements. 9, 44

OSI Open Source Initiative. 51, 53, Glossary: OSI

QoS Quality of Service. 8, 45, Glossary: QoS

RTOS Real Time Operating System. 8, 52, Glossary: RTOS

SaaS Software as a Service. Glossary: SaaS

SDK Software Development Kit. 29, Glossary: SDK

SNS Simple Notification Service . Glossary: SNS

SQS Simple Queue Service[119] . Glossary: SQS

STOMP Simple Text Oriented Messaging Protocol[120]. Glossary: STOMP

TCP Transmission Control Protocol[121]. Glossary: TCP

UML Unified Modeling Language. 54, Glossary: UML

WAMP Web Application Messaging Protocol[123]. Glossary: WAMP

XML Extensible Markup Language[124]. Glossary: XML

Bibliography

[1] R. Aebersold, Requirements gathering, 2019.

[2] T. Helbling, "Project thesis proposal - cloudevent router," Apr. 2019.

[3] O. Zimmermann, "Aufgabenstellung studienarbeit," Sep. 2019.

[4] C. N. Steve Klabnik, The rust programming language. https://doc.rust-lang.org/book/: Random House LCC US, Aug. 12, 2019, ISBN: 1718500440. [Online]. Available: https://www.ebook.de/de/product/37149179/steve_klabnik_carol_nichols_the_rust_programming_language_covers_rust_2018.html.

[5] D. Bryant, "A quantum leap for the web," Medium, 2016. [Online]. Available: https://medium.com/mozilla-tech/a-quantum-leap-for-the-web-a3b7174b3c12.

[6] Rust. [Online]. Available: https://research.mozilla.org/rust/.

[7] Production users. [Online]. Available: https://www.rust-lang.org/production/users.

[8] "Stack overflow developer survey 2019," Tech. Rep., 2019. [Online]. Available: https://insights.stackoverflow.com/survey/2019.

[9] S. Chacon and B. Straub, Pro git. Apress, 2014.

[10] K. Johnson, Github passes 100 million repositories, Nov. 2018. [Online]. Available: https://venturebeat.com/2018/11/08/github-passes-100-million-repositories/.

[11] Apache license, version 2.0, 2019. [Online]. Available: https://www.apache.org/licenses/LICENSE-2.0.

[12] Mqtt homepage. [Online]. Available: http://mqtt.org/.

[13] A. Banks and R. Gupta, Mqtt version 3.1.1 plus errata 01, OASIS Standard Incorporating, Dec. 2015. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html.

[14] Aws iot core. [Online]. Available: https://aws.amazon.com/iot-core/.

[15] Json - javascript | mdn, Nov. 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON.

[16] Unix(7) - linux manual page. [Online]. Available: http://man7.org/linux/man-pages/man7/unix.7.html.

[17] What is a serial interface? [Online]. Available: https://www3.nd.edu/~lemmon/courses/ee224/web-manual/web-manual/lab9/node4.html.

[18] G. Hohpe and B. Woolf, Enterprise integration patterns, 1, Ed. Addison-Wesley, Dec. 20, 2019, 480 pp., ISBN: 978-0321200686. [Online]. Available: https://www.enterpriseintegrationpatterns.com/patterns/messaging/ContentBasedRouter.html.

[19] L. A. Williams, "The collaborative software process," PhD thesis, The University of Utah, 2000. [Online]. Available: http://collaboration.csc.ncsu.edu/laurie/Papers/dissertation.pdf.

[20] M. Fowler, "Pairprogrammingmisconceptions," 2006. [Online]. Available: https://martinfowler.com/bliki/PairProgrammingMisconceptions.html.

[21] Buildkite features, 2019. [Online]. Available: https://buildkite.com/features.

[22] Caru smart sensor | notruf für senioren | #agetech – caru - wir machen lebensräume intelligent. 2019. [Online]. Available: https://www.caruhome.com/.

[23] Amazon ec2 instance types. [Online]. Available: https://aws.amazon.com/ec2/instance-types.

[24] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, Pattern-oriented software architecture. Jul. 12, 1996, 476 pp., ISBN: 0471958697. [Online]. Available: https://www.ebook.de/de/product/3239671/frank_buschmann_regine_meunier_hans_rohnert_peter_sommerlad_michael_stal_pattern_oriented_software_architecture.html.

[25] M. Nygard, Documenting architecture decisions, Nov. 5, 2019. [Online]. Available: http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions.

[26] Markdown architectural decision records, Nov. 5, 2019. [Online]. Available: https://github.com/adr/madr/tree/2.1.2.

[27] U. Zdun, R. Capill, H. Tran, and O. Zimmermann, Sustainable architectural design decisions, Mar. 9, 2014. [Online]. Available: https://www.infoq.com/articles/sustainable-architectural-design-decisions/.

[28] Ce-rust: Thesis (studien arbeit), Nov. 5, 2019. [Online]. Available: https://github.com/ce-rust/sa.

[29] Cerk (cloudevent router kernel, Nov. 5, 2019. [Online]. Available: https://github.com/ce-rust/cerk.

[30] O. Zimmermann, Making architectural knowledge sustainable – the y-approach, Nov. 5, 2019. [Online]. Available: https://resources.sei.cmu.edu/asset_files/Presentation/2012_017_001_31349.pdf.

[31] Adr.github.io, Nov. 5, 2019. [Online]. Available: https://adr.github.io/#sustainable-architectural-decisions.

[32] Rust homepage, 2019. [Online]. Available: https://www.rust-lang.org.

[33] The gnu c reference manual. [Online]. Available: https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html.

[34] Standard c++, 2019. [Online]. Available: https://isocpp.org/.

[35] The go programming language, 2019. [Online]. Available: https://golang.org/.

[36] Python.org, 2019. [Online]. Available: https://www.python.org/.

[37] Rust - embedded devices, 2019. [Online]. Available: https://www.rust-lang.org/what/embedded.

[38] Freertos_rs. [Online]. Available: https://docs.rs/freertos_rs/0.3.0/freertos_rs/.

[39] Rust - validating references with lifetimes, 2019. [Online]. Available: https://doc.rust-lang.org/1.30.0/book/2018-edition/ch10-03-lifetime-syntax.html.

[40] Rust - what is ownership? 2019. [Online]. Available: https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html.

[41] I. Barland, Why c and c++ are awful programming languages, 2019. [Online]. Available: https://www.radford.edu/ibarland/Manifestoes/whyC++isBad.shtml.

[42] I. Joyner, C++??: A critique of c++, 1992. [Online]. Available: http://www.literateprogramming.com/c++critique.pdf.

[43] Micropython, Nov. 5, 2019. [Online]. Available: https://micropython.org/.

[44] M. Fowler, Microservices, Mar. 23, 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html.

[45] M. Fowler, Patterns of enterprise application architecture. Addison Wesley, Nov. 1, 2002, 533 pp., ISBN: 0321127420.
[Online]. Available: https://www.martinfowler.com/books/eaa.html.

[46] Linkage - the rust reference, Nov. 12, 2019.
[Online]. Available: https://doc.rust-lang.org/reference/linkage.html.

[47] Dll hell - a article about dll hell, Nov. 12, 2019. [Online]. Available:
https://web.archive.org/web/20080703211550/http://dllcity.com/dll-hell.php.

[48] Dynamic loading & plugins, Nov. 12, 2019.
[Online]. Available: https://michael-f-bryan.github.io/rust-ffi-guide/dynamic_loading.html.

[49] Why async? 2019. [Online]. Available:
https://rust-lang.github.io/async-book/01_getting_started/02_why_async.html.

[50] Using threads to run code simultaneously, 2019.
[Online]. Available: https://doc.rust-lang.org/book/ch16-01-threads.html.

[51] Cannot define async fn with no_std #56974.
[Online]. Available: https://github.com/rust-lang/rust/issues/56974.

[52] Async-await hits beta!
[Online]. Available: https://blog.rust-lang.org/2019/09/30/Async-await-hits-beta.html.

[53] N. Matsakis, Async-await on stable rust! 2019.
[Online]. Available: https://blog.rust-lang.org/2019/11/07/Async-await-stable.html.

[54] G. Hohpe and B. Woolf, Message broker, Nov. 5, 2019. [Online]. Available:
https://www.enterpriseintegrationpatterns.com/patterns/messaging/MessageBroker.html.

[55] Multi-producer, single-consumer fifo queue communication primitives, 2019.
[Online]. Available: https://doc.rust-lang.org/std/sync/mpsc/index.html.

[56] R. Hanmer, Solution: Use a broker, Nov. 5, 2019.
[Online]. Available: https://www.oreilly.com/library/view/pattern-oriented-software-architecture/9781119963998/chap12-sec005.html.

[57] Using message passing to transfer data between threads, 2019.
[Online]. Available: https://doc.rust-lang.org/book/ch16-02-message-passing.html.

[58] A thread-safe reference-counting pointer, 2019.
[Online]. Available: https://doc.rust-lang.org/std/sync/struct.Arc.html.

[59] Useful synchronization primitives, 2019.
[Online]. Available: https://doc.rust-lang.org/std/sync/index.html.

[60] G. Starke, Beispiele für qualitätsanforderungen an software, GitHub, Oct. 2017.
[Online]. Available: https://github.com/arc42/quality-requirements.

[61] Iso-25010 in agile architecture, 2012. [Online]. Available:
http://a2build.com/architectedagile/Architected%20Agile.html?ISO25010.html.

[62] Cloudevents specification v0.3, 2019.
[Online]. Available: https://github.com/cloudevents/spec/blob/v0.3/spec.md.

[63] Introduction to latex. [Online]. Available: https://www.latex-project.org/about/.

[64] The latex project public license. [Online]. Available: https://www.latex-project.org/lppl/.

[65] Academic free license ("afl") v. 3.0 | open source initiative.
[Online]. Available: https://opensource.org/licenses/AFL-3.0.

[66] 1.3. amqp - advanced message queuing protocol red hat enterprise mrg 3 | red hat customer portal, 2019. [Online]. Available:
https://access.redhat.com/documentation/en-us/red_hat_enterprise_mrg/3/html/messaging_programming_reference/amqp____advanced_message_queuing_protocol.

[67] R. Godfrey, D. Ingham, and R. Schloming,
OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0, OASIS, Oct. 2012.
[Online]. Available: https://www.oasis-open.org/standards#amqpv1.0.

[68] P. Eising, What exactly is an api? - perry eising - medium, Medium, Dec. 2017.
[Online]. Available: https://medium.com/@perrysetgo/what-exactly-is-an-api-69f36968a41f.

[69] Arc42 - arc42. [Online]. Available: https://arc42.org/.

[70] Download arc42 - arc42. [Online]. Available: https://arc42.org/download.

[71] Arc42 license - arc42. [Online]. Available: https://arc42.org/license.

[72] What is aws. [Online]. Available: https://aws.amazon.com/what-is-aws/.

[73] Knative serving license. [Online]. Available: https://techterms.com/definition/bytecode.

[74] B. Stroustrup, Programming. Addison Wesley, May 15, 2014, 1312 pp., ISBN: 0321992784.
[Online]. Available:
https://www.ebook.de/de/product/21839507/bjarne_stroustrup_programming.html.

[75] H. Patel, Python - c++ bindings | hardik patel, Dec. 2017.
[Online]. Available: https://www.hardikp.com/2017/12/30/python-cpp/.

[76] Creative commons - cc0, Nov. 5, 2019.
[Online]. Available: https://creativecommons.org/share-your-work/public-domain/cc0.

[77] Continuous integration | thoughtworks, 2019.
[Online]. Available: https://www.thoughtworks.com/continuous-integration.

[78] Cncf cloud native definition v1.0, 2019.
[Online]. Available: https://github.com/cncf/toc/blob/master/DEFINITION.md.

[79] Cloudevents/spec at v1.0, GitHub, 2019.
[Online]. Available: https://github.com/cloudevents/spec/tree/v1.0.

[80] [Online]. Available: https://www.cncf.io/cncf-kubernetes-project-journey/.

[81] Cncf prometheus project journey - cloud native computing foundation, 2019.
[Online]. Available: https://www.cncf.io/cncf-prometheus-project-journey/.

[82] Members - cloud native computing foundation.
[Online]. Available: https://www.cncf.io/about/members/.

[83] What is copyleft? [Online]. Available: https://www.gnu.org/licenses/copyleft.en.html.

[84] S. Efftinge, M. Volter, A. Haase, and B. Kolb, The pragmatic code generator programmer,
Sep. 2006. [Online]. Available:
https://www.theserverside.com/news/1365073/The-Pragmatic-Code-Generator-Programmer.

[85] Amazon elastic kubernetes service. [Online]. Available: https://aws.amazon.com/eks/.

[86] M. Rouse, "What is function as a service (faas)? - definition from whatis.com,"
TechTarget, 2018.

[87]
Freertos - market leading rtos (real time operating system) for embedded systems with internet of things
[Online]. Available: https://www.freertos.org/.

[88] Gnu general public license v2.0 - gnu project - free software foundation, Sep. 2017.
[Online]. Available: https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html.

[89] The gnu general public license v3.0 - gnu project - free software foundation, Nov. 2016.
[Online]. Available: https://www.gnu.org/licenses/gpl-3.0.html.

[90] C. Doxsey, An introduction to programming in go. 2012, ISBN: 978-1478355823.
[Online]. Available: http://www.golang-book.com/books/intro.

[91] The go project - the go programming language, 2019.
[Online]. Available: https://golang.org/project/.

[92] H. F. Nielsen, J. Mogul, L. M. Masinter, R. T. Fielding, J. Gettys, P. J. Leach, and T. Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, RFC 2616, Jun. 1999. DOI: 10.17487/RFC2616. [Online]. Available: https://rfc-editor.org/rfc/rfc2616.txt.

[93] P. Mukhopadhyay, "Sales transformations for "industry 4.0"," Forbes, Oct. 2019. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2019/10/16/sales-transformations-for-industry-4-0/#23aafb725a42.

[94] C. McClelland, "What is iot? - a simple explanation of the internet of things," IoT For All, May 2019. [Online]. Available: https://www.iotforall.com/what-is-iot-simple-explanation/.

[95] Apr. 2012. [Online]. Available: https://techterms.com/definition/java.

[96] Javascript definition, Aug. 2014. [Online]. Available: https://techterms.com/definition/javascript.

[97] Ecmascript® 2019 language specification, Ecma International. [Online]. Available: https://www.ecma-international.org/ecma-262/10.0/index.html.

[98] Jdk definition from pc magazine encyclopedia. [Online]. Available: https://www.pcmag.com/encyclopedia/term/45608/jdk.

[99] B. Venners, Inside the java virtual machine. McGraw Hill, 1999, ISBN: 0-07-135093-4.

[100] Knative eventing, GitHub. [Online]. Available: https://github.com/knative/eventing/.

[101] Cncf kubernetes project journey. [Online]. Available: https://www.cncf.io/cncf-kubernetes-project-journey/.

[102] Linux from foldoc, 2000. [Online]. Available: http://foldoc.org/linux.

[103] The linux kernel archives - faq. [Online]. Available: https://www.kernel.org/category/faq.html.

[104] J. Gruber, Daring fireball: Markdown. [Online]. Available: https://daringfireball.net/projects/markdown/.

[105] G. Haff, The mysterious history of the mit license | opensource.com, Apr. 2019. [Online]. Available: https://opensource.com/article/19/4/history-mit-license.

[106] "Minimum viable product (mvp)," [Online]. Available: https://www.techopedia.com/definition/27809/minimum-viable-product-mvp.

[107] Introduction to node.js. [Online]. Available: https://nodejs.dev/.

[108] Frequently answered questions: What is "open source" software? [Online]. Available: https://opensource.org/faq#osd.

[109] About the open source initiative | open source initiative. [Online]. Available: https://opensource.org/about.

[110] Drawing uml with plantuml, plantuml language reference guide, 2019. [Online]. Available: http://plantuml.com/guide.

[111] About pull requests. [Online]. Available: https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests.

[112] D. Kuhlman, A python book: Beginning python, advanced python, and python exercises, 1.1a. Apr. 2012. [Online]. Available: https://web.archive.org/web/20120623165941/http://cutter.rexx.com/~dkuhlman/python_book_01.html.

[113] E.800 : Definitions of terms related to quality of service, International Telecommunication Union (ITU), Sep. 2008. [Online]. Available: https://www.itu.int/rec/T-REC-E.800-200809-I/en.

[114] Why rtos and what is rtos? [Online]. Available: https://www.freertos.org/about-RTOS.html.

[115] The rustc book. [Online]. Available: https://doc.rust-lang.org/rustc/what-is-rustc.html.

[116] [Online]. Available: https://www.rust-lang.org/tools/install.

[117] M. Rouse, "What is software as a service (saas)? - definition from whatis.com," TechTarget, Feb. 2019.

[118] Amazon simple notification service. [Online]. Available: https://aws.amazon.com/sns/.

[119] Amazon simple queue service. [Online]. Available: https://aws.amazon.com/sqs/.

[120] Stomp protocol specification, version 1.2, 2012.

[121] J. Postel, Transmission control protocol, RFC 793, Sep. 1981. DOI: 10.17487/RFC0793. [Online]. Available: https://rfc-editor.org/rfc/rfc793.txt.

[122] C. Larman, Applying uml and patterns. Prentice Hall, 2004, vol. 3, ISBN: 9780131489066.

[123] T. Oberstein and A. Goedde, The web application messaging protocol, IETF, Aug. 2019. [Online]. Available: https://wamp-proto.org/_static/gen/wamp_latest_ietf.html#rfc.authors.

[124] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, Extensible markup language (xml) 1.0 (fifth edition), 2008. [Online]. Available: https://www.w3.org/TR/2008/REC-xml-20081126/.

C. Rust Experience Report

Rust Experience Report

Linus Basig, Fabrizio Lazzaretti

December 19, 2019

During this semester, we wrote a CloudEvents router in Rust, which has been our first experience with this programming language.

Rust is a low-level programming language with guaranteed thread and memory safety. It achieves that with its unique ”ownership model”. The goal is to build safe and reliable software.[1, 2]
Rust is used by Mozilla Firefox, Dropbox and many others.[3, 4]
In 2019, Rust has also been elected in the well-known survey of Stack Overflow for being the "most loved" programming language.[5]

The ”ownership model” is the most unique feature of Rust. In Rust memory is allocated and freed explicit, but in contrast to C or other low level programming languages without the need of an garbage collector, Rust enforces a set of rules on this actions, to ensure memory safety.”[1]

We learned the basics of Rust by reading the book The Rust Programming Language[1]. The freely available book is straight forward to read and provides a good introduction into the basic concepts such as pattern-matching, ownership, error handling and the tooling with cargo, the package manager for Rust.

It is really easy to implement the ”hello world” program in Rust. The syntax is similar to C as shown in the example below. As soon as we wanted to implement something more complex than the ”hello world” program, we realized that the ownership model was not as easy as initially thought.

```
use ferris_says::say; // from the previous step
use std::io::{stdout, BufWriter};

fn main() {
    let stdout = stdout();
    let out = b"Hello fellow Rustaceans!";
    let width = 24;

    let mut writer = BufWriter::new(stdout.lock());
    say(out, width, &mut writer).unwrap();
}
```

Listing 1: A small Rust application from the getting-started page of the Rust home page[6]

Struggle with the ownership model

In the small sample program in Listing 1 Rust’s unique ”ownership model” can already be seen in action. The reference to the `BufWriter` is declared as mutable (`mut`). As a next step, the ownership of the object is borrowed as a mutable reference (`&mut writer`) to the function `say()`. In this case

Rust's ownership model guarantees that `writer` is valid after the function `say()` returns because `writer` ownership was borrowed and not transferred to `say()`.

At the beginning, the biggest problems were related to the ownership model of Rust. It verifies that every variable is only used in one place, which is crucial to guarantee memory safety.[1]

This feature might not be intuitive for users more familiar with other programming languages. Both of us had to spend a significant amount of time to get acquainted with this mode of work.

In general, the error messages in Rust are really comprehensible and make it easy to solve the encountered problem.

```
fn my_function(number: &String) {
    print!("received number {}", number);
}

#[derive(Debug)]
enum State {
    Init(String),
}

fn main() {
    let state = State::Init(String::from("test"));
    loop {
        match state {
            State::Init(name) => my_function(&name),
        }
    }
}
```

```
   Compiling ownership-problem v0.1.0
    ↪  (/home/fab/hsr/ce-rust/sa/rust-samples/ownership-problem)
error[E0382]: use of moved value
  --> ownership-problem/src/main.rs:14:25
   |
14 |             State::Init(name) => my_function(&name),
   |                         ^^^^ value moved here, in previous iteration of loop
   |
   = note: move occurs because value has type `std::string::String`, which does not
    ↪  implement the `Copy` trait

error: aborting due to previous error

For more information about this error, try `rustc --explain E0382`.
error: could not compile `ownership-problem`.

To learn more, run the command again with --verbose.
```

Listing 2: The first struggle in Rust and the ownership model. The code does not compile because the variable is moved to the called function `my_function` after the first loop iteration.

Traits and Object-Oriented Approach

At the beginning we used an object-oriented approach for the implementation in Rust. But as we got to know the language better, we realized that using a classical object-oriented approach with heavy use of inheritance does not work as well with Rust as with other languages. The traits in rust are not exactly the same as classes or interfaces in other languages.

Traits are a language feature to define interfaces for different types in a standard way (they are also used for markers).

Traits are by default statically dispatched with zero-cost abstraction, but can also be dynamically dispatched.

With the default, static dispatch generics are removed at compile time like templates in C++, but unlike in C++, the traits are also type-checked without a concrete implementation. In addition, other trait functions are compiled per type.

There is an overhead when using dynamic dispatching, but there is no need to compile every concrete type statically. The compiler internally generates a ”trait type” which uses a vtable (a vtable is a table to store pointers to the virtual functions) to resolve the concrete type. This provides bigger flexibility: with dynamic dispatching the concrete implementation could change during runtime, based on the used concrete type. The downside of this approach is the fact that the functions use a trait type which is `unsized`. The compiler doesn't know the size of the object at compile time, this means that the dynamically dispatched trait type cannot be used in the code without pointer.

However, to us the most confusing part was the classical upcasting, which does not work from one trait to another out of the box. This is because Rust has one vtable entry for the type. This type is identified by a unique type id. A workaround is shown in 3.

Thus, using the type with a ”trait type” is more restricted and not as easy to use as a generic in C#.[7–9]

```
trait Base: AsBase {
  // ...
}

trait AsBase {
  fn as_base(&self) -> &Base;
}

impl<T: Base> AsBase for T {
  fn as_base(&self) -> &Base {
    self
  }
}
```

Listing 3: A minimal Example to show how traits types could be changed for the compiler. This is not done implicitly because in Rust every object has only one pointer at the type and one at the data. Example from Stack Overflow[1]

[1] https://stackoverflow.com/a/28664881/2801860

Enums and Pattern Matching

Classical enums, also called enumerations like the enums in C store a numerical value for each given label.

The enums in Rust not only allow the usage of values but also of attached data. The attached value could be different on every enum value. An example is provided in Listing 4.[1]

The pattern matching allow to execute certain code on a certain enum value. This is a powerful tool and combined with the ability to directly access the date attached to the enum, it is really intuitive to read and write. The Rust compile checks on every compilation if every pattern match is exhaustive. For a user this is very convenient, as it ascertains that every case is always handled. If the use case does not require to handle all cases, a default clause could be defined at the end as done in Listing 4.

```
pub enum BrokerEvent {
  Init,
  ConfigUpdated(Config, InternalServerId),
  OutgoingCloudEvent(CloudEvent, InternalServerId),
  // ...
}
// ...
match inbox.receive() {
  BrokerEvent::Init => info!("{} initiated", id),
  BrokerEvent::ConfigUpdated(_, _) => info!("{} received ConfigUpdated", id),
  BrokerEvent::OutgoingCloudEvent(cloud_event, _) => info!(
  "{} received cloud event with id={}!",
  id,
  get_event_field!(cloud_event, event_id)
  ),
  broker_event => warn!("event {} not implemented", broker_event),
}
```

Listing 4: A small excerpt from the CERK repository, showing the use of enums and a pattern matching on the enum that is return from the function `inbox.receive()`.

Tools and IDE support

The language is not widely used, which is why IDE support is not the greatest. However, there is a good plug-in for Visual Studio Code called ”Rust (rls)” which shows error, adds some navigation features and shows some documentation.[10] Additionally, the errors from the compiler are very helpful, which makes an IDE error help not as necessary as in other languages.

Package management is done with Cargo, the default package manager for Rust. It is really simple, supports compiling, installs packages, runs tests, formats code and generates the documentation. Cargo can be extended with additional subcommands by installing packages via Cargo itself.

Next Steps with Rust

Before we propose any next steps, we would like to summarize the possible motives to initiate the adoption of Rust.

The main argument for Rust is its ownership model and the safety guarantees that come with it. We believe the benefits are worth the small effort of internalizing the new concept.

When comparing to Python, the missing garbage collector is certainly beneficiary regarding performance and resource consumption. Also, dependency management is more straightforward with Rust. All the dependencies get compiled into one binary; no Python packages need to be installed on the device.

At the end of the day, it is very enjoyable to write code in Rust. The compiler becomes your best friend, constantly preventing you from making avoidable mistakes.

Even though the first couple of hours were rough, it became very pleasant to work with Rust after a while. We were eventually fully convinced of Rust after we first spent a few hours on our own, creating small prototypes, and then pair-programed very focused on a concrete task. This phase boosted our understanding of the language, and we had many epiphanies.

So our recommendation for the adoption of Rust would be to find a small, well-defined service and task a pair of curious software engineers to implement it in Rust. Then split the pair and let them implement a different service with a different partner. And so on.

References

[1] C. N. Steve Klabnik, The rust programming language. https://doc.rust-lang.org/book/: Random House LCC US, Aug. 12, 2019, ISBN: 1718500440. [Online]. Available: `https://www.ebook.de/de/product/37149179/steve_klabnik_carol_nichols_the_rust_programming_language_covers_rust_2018.html`.

[2] D. Bryant, "A quantum leap for the web," Medium, 2016. [Online]. Available: `https://medium.com/mozilla-tech/a-quantum-leap-for-the-web-a3b7174b3c12`.

[3] Rust. [Online]. Available: `https://research.mozilla.org/rust/`.

[4] Production users. [Online]. Available: `https://www.rust-lang.org/production/users`.

[5] "Stack overflow developer survey 2019," Tech. Rep., 2019. [Online]. Available: `https://insights.stackoverflow.com/survey/2019`.

[6] Getting started - rust programming language. [Online]. Available: `https://www.rust-lang.org/learn/get-started`.

[7] A. Turon, "Abstraction without overhead: Traits in rust," Rust Blog, May 2015. [Online]. Available: `https://blog.rust-lang.org/2015/05/11/traits.html`.

[8] J. L.-d. Toit, "Traits and trait objects in rust," 2018. [Online]. Available: `https://joshleeb.com/posts/rust-traits-and-trait-objects/`.

[9] I. Dubrov, Dynamic casting for traits, Jun. 2018. [Online]. Available: `http://idubrov.name/rust/2018/06/16/dynamic-casting-traits.html`.

[10] Rust support for visual studio code, GitHub, 2019. [Online]. Available: `https://github.com/rust-lang/rls-vscode`.

D. Tools

In this chapter we describe the tools we used to write the thesis and to create our product.

D.1. General Tools

The tools in this section were used in the documentation and for the product.

D.1.1. GitHub

For our project hosting we decided to use GitHub.

GitHub is a source code management and collaboration platform based on Git. It is a well known platform for open-source code and was acquired in 2018 by Microsoft.[142]

We decided to use GitHub, because it is the most used platform for open source code repositories.

GitHub's project management functionalities are quite limited. There are issues and milestones. Planning with them is quite hard for bigger projects, but it was just enough for our project size.

D.1.2. T-Metric

We used T-Metric for time tracking.

T-Metric is a time tracking app with a functionally limited free plan for up to 5 people. It has integrations for various products, such as GitHub.[184]

The tool is quite simple, and so are the reporting functions. The collected data can be exported as a CSV or a PDF.

D.2. Tools for the Documentation

This document was written with LaTeX.

"LaTeX, which is pronounced «Lah-tech» or «Lay-tech» (to rhyme with «blech» or «Bertolt Brecht»), is a document preparation system for high-quality typesetting. It is most often used for medium-to-large technical or scientific documents but it can be used for almost any form of publishing."[113] LaTeXis distributed under the LaTeX project public license. It is a free software license.[114]

Both team members were familiar with LaTeX before this thesis started, but we both improved our skills in this text writing format.

We prefer LaTeX over common graphical text processors like Word because the files are plain text and are therefore suitable to be versioned with source control tools like Git. The advantage of using Git for versioning the documentation is that we were able to have the same glsgithub pull request based review procedure as we use for our code. This procedure ensures that both partners have read every bit of text before it is added to the master branch.

D.2.1. Code Highlighting

We used the minted LaTeX extension for code syntax highlighting in our documents. Minted supports Rust syntax highlighting for inline code snippets or code from an external file.[193]

D.2.2. Diagrams

The diagrams in the documentation are not directly build with LaTeX. We used PlantUML for this.

PlantUML is a tool to generate different UML and non UML diagrams. The diagrams are generated by writing with the PlantUML Language. The tool is written in Java and has various output formats, one of the im JPGE.[168]

PlantUML source files are also plain text and therefore support our strategy of Git versioning and GitHub pull request review procedure as well.

D.2.3. CI

A disadvantage of writing the documentation in LaTeX and not a graphical editor is that it has to be compiled. We encountered a few situations where some source files became corrupted and refused to compile. To debug these issues was quite frustrating because the error messages and warnings are very cryptic. LaTeX source files are not very readable for people who are not familiar with the syntax.

In order to counteract these points we have built in a CI to build the pdf files automatically on every Git push.

The CI has two tasks:

1. build the diagrams
2. build the pdfs

For the CI we used Buildkite. Buildkite is explained in more detail in subsection D.4.2

D.3. Tools for the Product

Our program is written in Rust, so our main tool was the rust compiler: rustc.

rustc is the Rust compiler provided by the Rust Team. It can be used directly (`rustc`) or with Cargo (`cargo build`).[177]

The package management was done with Cargo.

Cargo is the default package manager for Rust. It is really simple, compiles code, installs packages, runs tests, formats source files and generates the documentation. Cargo can be extended with additional subcommands by installing packages via Cargo itself.

D.4. rustup

We installed and switched between Rust versions with rustup (Rust has a release cycle of 6 weeks[178]).

rustup is a tool to install different versions of Rust, like stable, beta and nightly, and for different compilation targets.
It is the recommended way for developers to install Rust.[178]

D.4.1. IDE

The language is not widely used, which is why IDE support is not the greatest. However, there is a good plug-in for Visual Studio Code called "Rust (rls)" which shows error, adds some navigation features and shows some documentation.[194] Additionally, the errors from the compiler are very helpful, which makes an IDE error help not as necessary as in other languages.

D.4.2. CI

We used Buildkite to automatically run the tests for our code after each push.

Buildkite is a service for coordinating buildpipelines which get executed on own infrastructure. They provide a build agent that can easily be deployed to AWS and other platforms. Buildkite is well integrated with GitHub.[124]

E. Evolution of the Building Blocks

In this appendix the evolution that our product made is described with the help of different diagrams.

At the beginning we had the borad idea of a Microkernel based router, with one component per adapter (Figure E.1). Then we started with the details and got quite complex approach that used traits, shown in Figure E.2. This approach got even more complex in the second version in Figure E.3 and Figure E.4.

With a more functional approach we were able to mitigate the unnecessary complexity and arrived at the simple building blocks view in Figure E.5.

The main reason for that was our classical object-oriented approach. The problem with that was that, in the beginning, it looked like Rust fully supports this approach. But as we got to know the language better, we realized that using a classical object-oriented approach with heavy us of inheritance does not work as good with Rust as with other languages.

More details about our experience with Rust and the trait model as an object-oriented approach could be find in Appendix C.

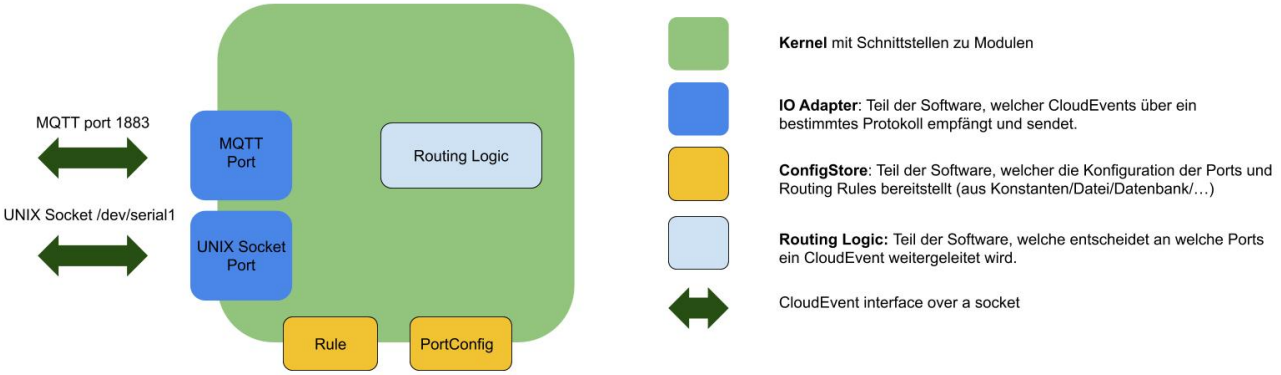

Figure E.1.: Initial Concepts: Image from the first presentation

First Version

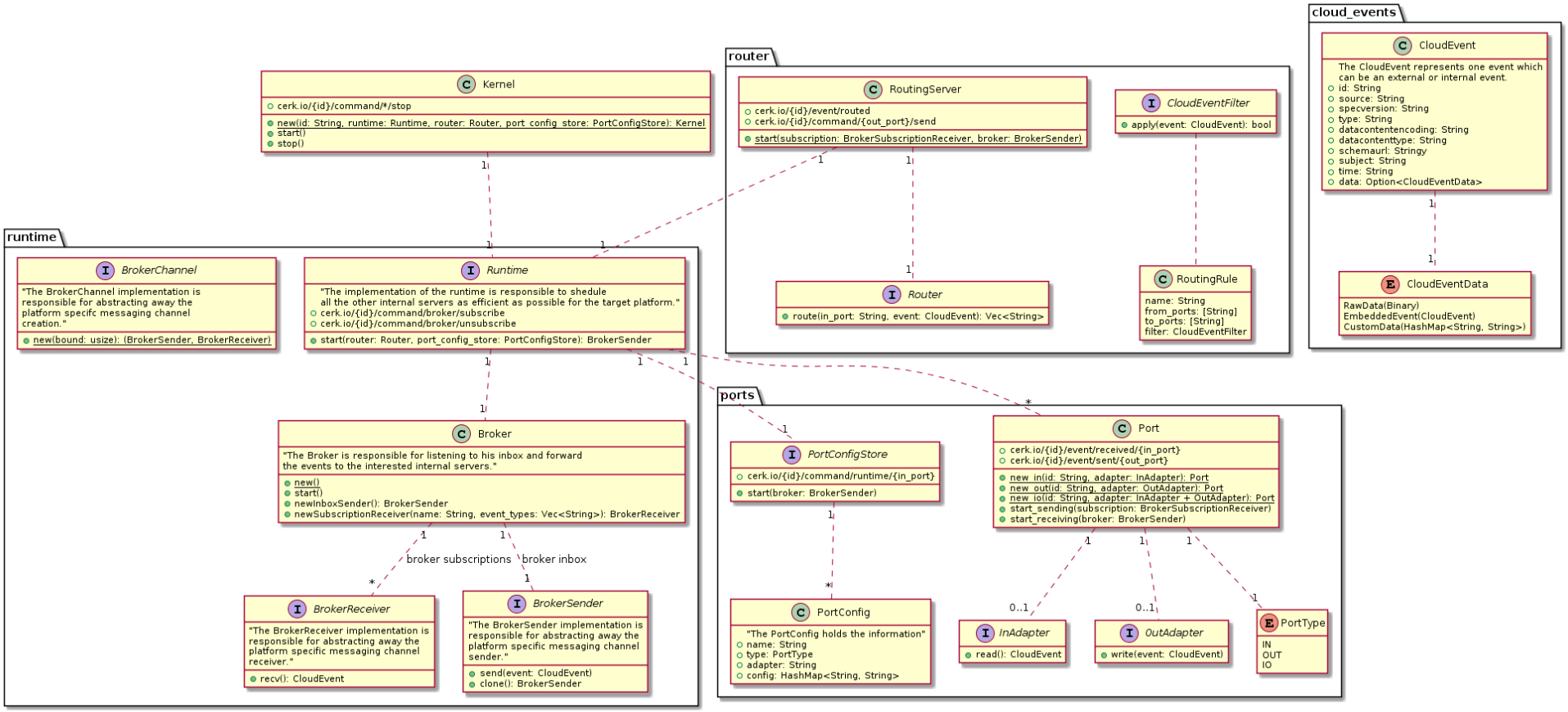

Figure E.2.: First Version: Traits Diagram

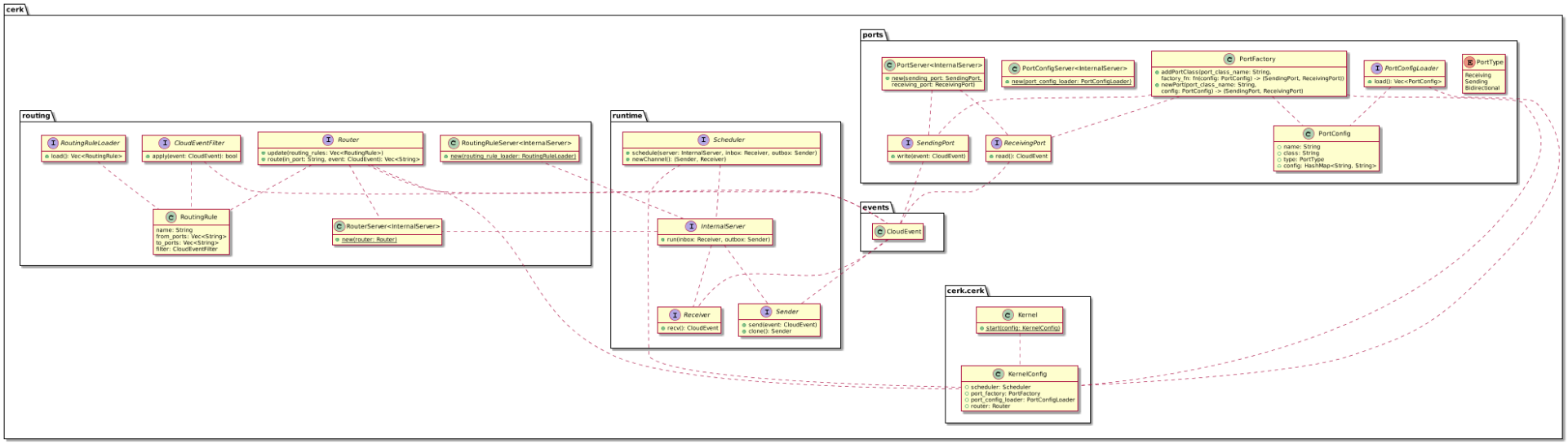

Figure E.3.: Second Version: Traits Diagram

Second Version

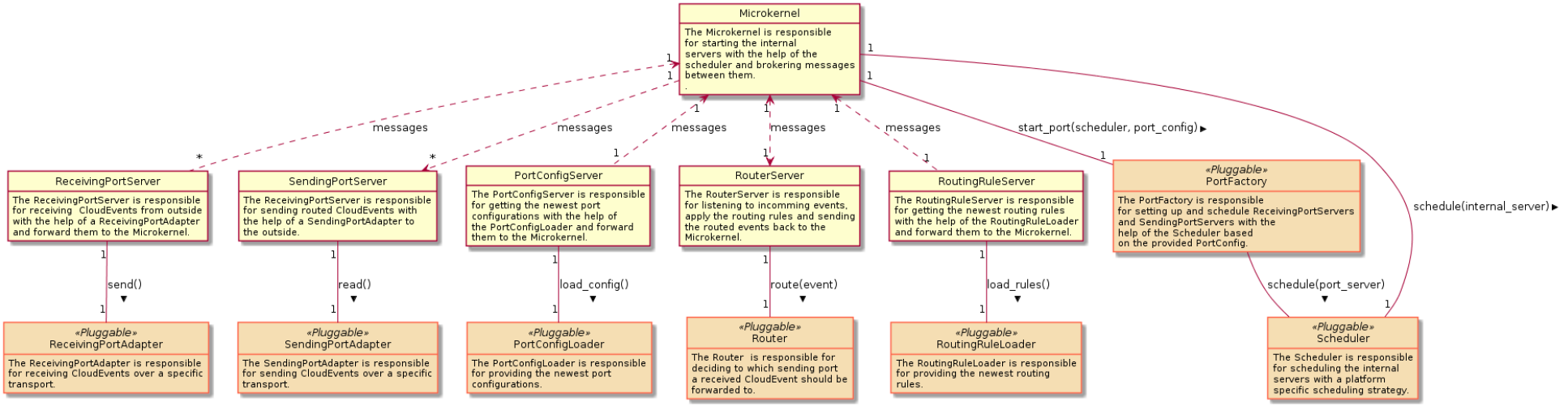

Figure E.4.: Second Version: Building Blocks

Final Version

We changed our approach and used a more functional style which reduced the complexity noticeably. In the end our high level decomposition was even simpler as our initial draft.

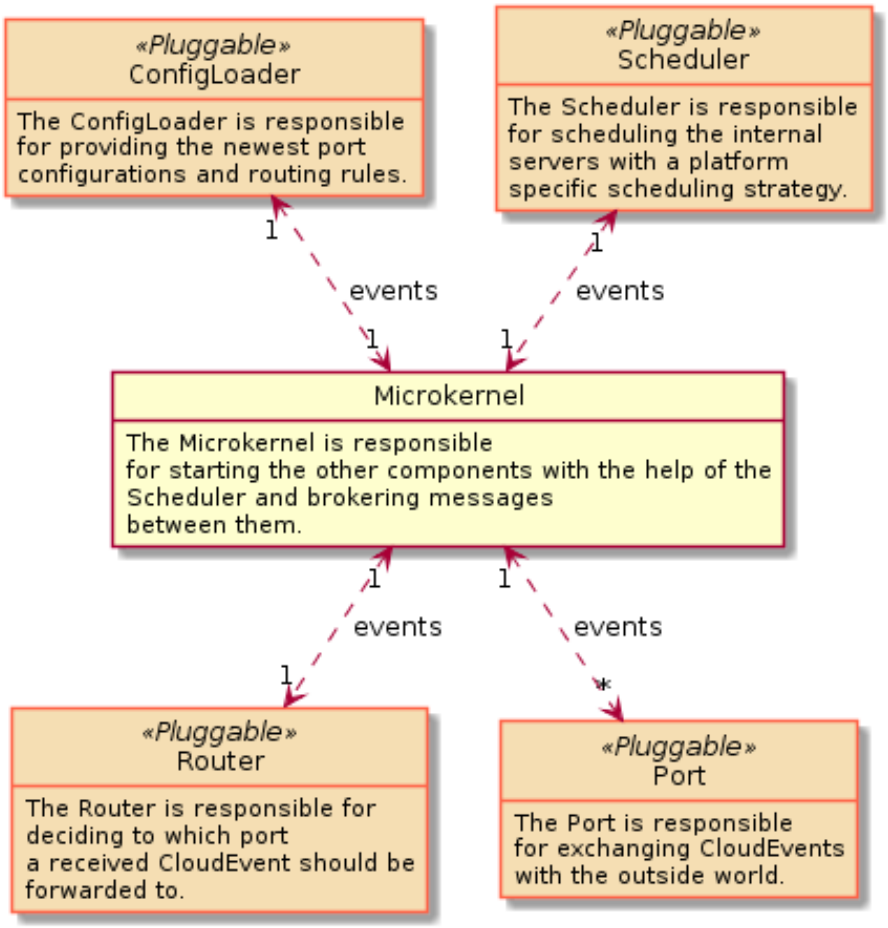

Figure E.5.: Final Version: Building Blocks

F. Test Protocol: First Apache Camel Tutorial

F.1. Requirement

- java JDK
- Maven

F.2. Environment

```
[H]
> uname -srv
Linux 4.19.69-1-MANJARO #1 SMP PREEMPT Thu Aug 29 08:51:46 UTC 2019
> java --version
openjdk 12.0.2 2019-07-16
OpenJDK Runtime Environment (build 12.0.2+10)
OpenJDK 64-Bit Server VM (build 12.0.2+10, mixed mode)
> mvn --version
Apache Maven 3.6.1 (NON-CANONICAL_2019-07-24T20:49:02Z_root; 2019-07-24T22:49:02+02:00)
Maven home: /opt/maven
Java version: 12.0.2, vendor: N/A, runtime: /usr/lib/jvm/java-12-openjdk
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.19.69-1-manjaro", arch: "amd64", family: "unix"
```

F.3. Code

```
public final class CamelJmsToFileExample {
  public static void main(String args[]) throws Exception {
    CamelContext context = new DefaultCamelContext();
    // Set up the ActiveMQ JMS Components
    ConnectionFactory connectionFactory = new
     ↪  ActiveMQConnectionFactory("vm://localhost?broker.persistent=false");
    // Note we can explicit name the component
    context.addComponent("test-jms",
     ↪  JmsComponent.jmsComponentAutoAcknowledge(connectionFactory));
    // Add some configuration by hand ...
    context.addRoutes(new RouteBuilder() {
      public void configure() {
        from("test-jms:queue:test.queue").to("file://test");
      }
    });
    // Camel template - a handy class for kicking off exchanges
    ProducerTemplate template = context.createProducerTemplate();
    // Now everything is set up - lets start the context
    context.start();
    // Now send some test text to a component - for this case a JMS Queue
    for (int i = 0; i < 10; i++) {
      template.sendBody("test-jms:queue:test.queue", "Test Message: " + i);
    }

    // wait a bit and then stop
    Thread.sleep(1000);
    context.stop();
  }
}
```

Listing 3: First Apache Camel Tutorial, code shorted [70]

F.4. Source

Documentation Walk through an Example Code - Apache Camel[69]

Repository https://github.com/apache/camel.git [70]

Branch master

Commit cdb6342f004294caf63e59b752a008287644d0d9

F.5. Commands

1. `cd camel/examples/camel-example-jms-file`
2. `mvn compile`
3. `mvn exec:java -PExample`

F.6. Execution Output

```
[INFO] BuildTimeEventSpy is registered.
[INFO] Scanning for projects...
[INFO]
[INFO] ----------< org.apache.camel.example:camel-example-jms-file >-----------
[INFO] Building Camel :: Example :: JMS-File 3.0.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec-maven-plugin:1.6.0:java (default-cli) @ camel-example-jms-file ---
[e.CamelJmsToFileExample.main()] DefaultCamelContext            INFO  Apache Camel
↪  3.0.0-SNAPSHOT (CamelContext: camel-1) is starting
[e.CamelJmsToFileExample.main()] DefaultManagementStrategy      INFO  JMX is
↪  disabled
[e.CamelJmsToFileExample.main()] DefaultCamelContext            INFO  StreamCaching
↪  is not in use. If using streams then its recommended to enable stream caching.
↪  See more details at http://camel.apache.org/stream-caching.html
[e.CamelJmsToFileExample.main()] BrokerService                  INFO  Using
↪  Persistence Adapter: MemoryPersistenceAdapter
[                JMX connector] ManagementContext               INFO  JMX consoles
↪  can connect to service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi
[e.CamelJmsToFileExample.main()] BrokerService                  INFO  Apache
↪  ActiveMQ 5.15.10 (localhost, ID:manjo-pc-38319-1570967402805-0:1) is starting
[e.CamelJmsToFileExample.main()] BrokerService                  INFO  Apache
↪  ActiveMQ 5.15.10 (localhost, ID:manjo-pc-38319-1570967402805-0:1) started
[e.CamelJmsToFileExample.main()] BrokerService                  INFO  For help or
↪  more information please see: http://activemq.apache.org
[e.CamelJmsToFileExample.main()] BrokerService                  WARN  Temporary
↪  Store limit is 51200 mb (current store usage is 0 mb). The data directory:
↪  /home/fab/hsr/ce-rust/camel/examples/camel-example-jms-file only has 11556 mb of
↪  usable space. - resetting to maximum available disk space: 11556 mb
[e.CamelJmsToFileExample.main()] TransportConnector             INFO  Connector
↪  vm://localhost started
[e.CamelJmsToFileExample.main()] DefaultCamelContext            INFO  Route: route1
↪  started and consuming from: test-jms://queue:test.queue
[e.CamelJmsToFileExample.main()] DefaultCamelContext            INFO  Total 1
↪  routes, of which 1 are started
[e.CamelJmsToFileExample.main()] DefaultCamelContext            INFO  Apache Camel
↪  3.0.0-SNAPSHOT (CamelContext: camel-1) started in 0.485 seconds
[e.CamelJmsToFileExample.main()] DefaultCamelContext            INFO  Apache Camel
↪  3.0.0-SNAPSHOT (CamelContext: camel-1) is shutting down
[e.CamelJmsToFileExample.main()] DefaultShutdownStrategy        INFO  Starting to
↪  graceful shutdown 1 routes (timeout 300 seconds)
[el-1) thread #2 - ShutdownTask] TransportConnector             INFO  Connector
↪  vm://localhost stopped
[el-1) thread #2 - ShutdownTask] BrokerService                  INFO  Apache
↪  ActiveMQ 5.15.10 (localhost, ID:manjo-pc-38319-1570967402805-0:1) is shutting
↪  down
[el-1) thread #2 - ShutdownTask] BrokerService                  INFO  Apache
↪  ActiveMQ 5.15.10 (localhost, ID:manjo-pc-38319-1570967402805-0:1) uptime 2.391
↪  seconds
[el-1) thread #2 - ShutdownTask] BrokerService                  INFO  Apache
↪  ActiveMQ 5.15.10 (localhost, ID:manjo-pc-38319-1570967402805-0:1) is shutdown
```

```
[el-1) thread #2 - ShutdownTask] DefaultShutdownStrategy        INFO  Route: route1
↪  shutdown complete, was consuming from: test-jms://queue:test.queue
[e.CamelJmsToFileExample.main()] DefaultShutdownStrategy        INFO  Graceful
↪  shutdown of 1 routes completed in 1 seconds
[e.CamelJmsToFileExample.main()] DefaultCamelContext            INFO  Apache Camel
↪  3.0.0-SNAPSHOT (CamelContext: camel-1) uptime 2.657 seconds
[e.CamelJmsToFileExample.main()] DefaultCamelContext            INFO  Apache Camel
↪  3.0.0-SNAPSHOT (CamelContext: camel-1) is shutdown in 1.022 seconds
[WARNING] thread
↪  Thread[ForkJoinPool.commonPool-worker-3,5,org.apache.camel.example.jmstofile.CamelJmsToFil
↪  was interrupted but is still alive after waiting at least 15000msecs
[WARNING] thread
↪  Thread[ForkJoinPool.commonPool-worker-3,5,org.apache.camel.example.jmstofile.CamelJmsToFil
↪  will linger despite being asked to die via interruption
[WARNING] NOTE: 1 thread(s) did not finish despite being asked to  via interruption.
↪  This is not a problem with exec:java, it is a problem with the running code.
↪  Although not serious, it should be remedied.
[WARNING] Couldn't destroy threadgroup
↪  org.codehaus.mojo.exec.ExecJavaMojo$IsolatedThreadGroup[name=org.apache.camel.example.jmst
java.lang.IllegalThreadStateException
    at java.lang.ThreadGroup.destroy (ThreadGroup.java:776)
    at org.codehaus.mojo.exec.ExecJavaMojo.execute (ExecJavaMojo.java:321)
    at org.apache.maven.plugin.DefaultBuildPluginManager.executeMojo
     ↪  (DefaultBuildPluginManager.java:137)
    at org.apache.maven.lifecycle.internal.MojoExecutor.execute
     ↪  (MojoExecutor.java:210)
    at org.apache.maven.lifecycle.internal.MojoExecutor.execute
     ↪  (MojoExecutor.java:156)
    at org.apache.maven.lifecycle.internal.MojoExecutor.execute
     ↪  (MojoExecutor.java:148)
    at org.apache.maven.lifecycle.internal.LifecycleModuleBuilder.buildProject
     ↪  (LifecycleModuleBuilder.java:117)
    at org.apache.maven.lifecycle.internal.LifecycleModuleBuilder.buildProject
     ↪  (LifecycleModuleBuilder.java:81)
    at
     ↪  org.apache.maven.lifecycle.internal.builder.singlethreaded.SingleThreadedBuilder.build
     ↪  (SingleThreadedBuilder.java:56)
    at org.apache.maven.lifecycle.internal.LifecycleStarter.execute
     ↪  (LifecycleStarter.java:128)
    at org.apache.maven.DefaultMaven.doExecute (DefaultMaven.java:305)
    at org.apache.maven.DefaultMaven.doExecute (DefaultMaven.java:192)
    at org.apache.maven.DefaultMaven.execute (DefaultMaven.java:105)
    at org.apache.maven.cli.MavenCli.execute (MavenCli.java:956)
    at org.apache.maven.cli.MavenCli.doMain (MavenCli.java:288)
    at org.apache.maven.cli.MavenCli.main (MavenCli.java:192)
    at jdk.internal.reflect.NativeMethodAccessorImpl.invoke0 (Native Method)
    at jdk.internal.reflect.NativeMethodAccessorImpl.invoke
     ↪  (NativeMethodAccessorImpl.java:62)
    at jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke
     ↪  (DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke (Method.java:567)
    at org.codehaus.plexus.classworlds.launcher.Launcher.launchEnhanced
     ↪  (Launcher.java:282)
```

```
    at org.codehaus.plexus.classworlds.launcher.Launcher.launch (Launcher.java:225)
    at org.codehaus.plexus.classworlds.launcher.Launcher.mainWithExitCode
     ↪  (Launcher.java:406)
    at org.codehaus.plexus.classworlds.launcher.Launcher.main (Launcher.java:347)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  19.321 s
[INFO] Finished at: 2019-10-13T13:50:20+02:00
[INFO] ------------------------------------------------------------------------
```

G. Test Protocol: First Node-RED Tutorial

G.1. Requirement

- Docker

G.2. Environment

```
> docker -v
Docker version 19.03.1-ce, build 74b1e89e8a
```

G.3. Code

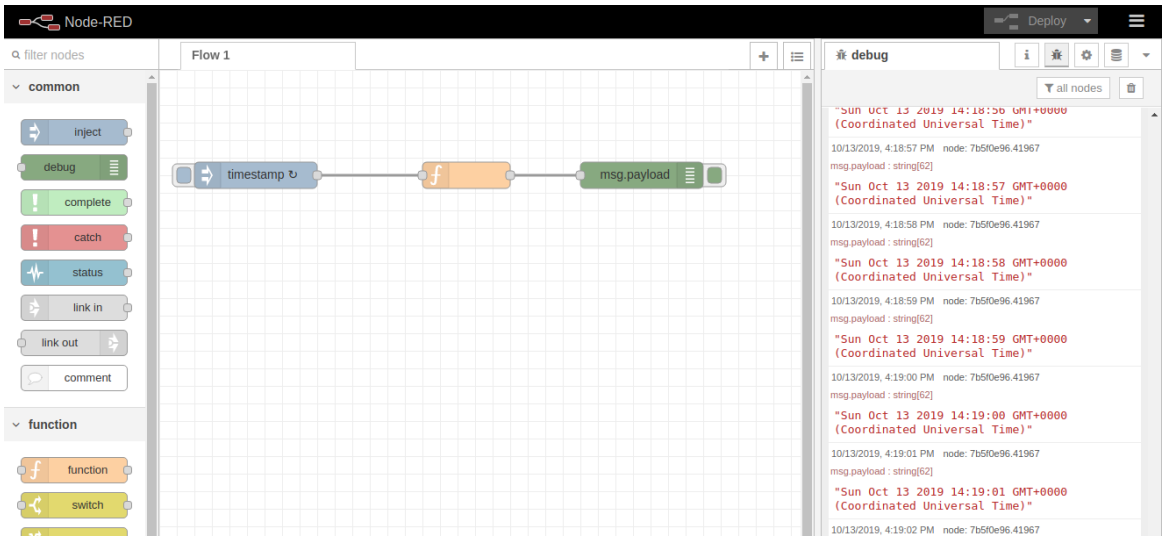

Figure G.1.: result of the first tutorial; left: the nodes that could be inserted in the flow; center: the flow that is created during the tutorial; right: the debugging console

G.4. Source

Documentation Creating your first flow : Node-RED[72]

Docker Image nodered/node-red:latest

Docker Image Id `sha256:243071efb3349b24cc20886ab8606fb13f349c4b6a1be7a15b11d2af189b327a`

G.5. Commands

1. `docker run -it -p 1880:1880 --name mynodered nodered/node-red`
2. drag and drop components like described in the tutorial[72]

G.6. Final Result


```
[
{
  "id": "5cb4b715.ae8af8",
  "type": "tab",
  "label": "Flow 1",
  "disabled": false,
  "info": ""
},
{
  "id": "cdb9909f.37467",
  "type": "inject",
  "z": "5cb4b715.ae8af8",
  "name": "",
  "topic": "",
  "payload": "",
  "payloadType": "date",
  "repeat": "1",
  "crontab": "",
  "once": true,
  "onceDelay": 0.1,
  "x": 110,
  "y": 120,
  "wires": [
  [
  "9649a09b.47068"
  ]
  ]
},
{
  "id": "7b5f0e96.41967",
  "type": "debug",
  "z": "5cb4b715.ae8af8",
  "name": "",
  "active": true,
  "tosidebar": true,
  "console": false,
  "tostatus": false,
  "complete": "false",
  "x": 550,
  "y": 120,
  "wires": []
},
{
  "id": "9649a09b.47068",
```

```
  "type": "function",
  "z": "5cb4b715.ae8af8",
  "name": "",
  "func": "// Create a Date object from the payload\n
    var date = new Date(msg.payload);\n
    // Change the payload to be a formatted Date string\nmsg.payload = date.toString();\n
    // Return the message so it can be sent on\n
    return msg;",
  "outputs": 1,
  "noerr": 0,
  "x": 350,
  "y": 120,
  "wires": [
  [
  "7b5f0e96.41967"
  ]
  ]
}
]
```

Listing 4: Export of the Flow from Node-RED in **??**, the file is reformatted

H. Test Protocol: First Knative Eventing Tutorial

The Knative tutorial with Istio failed. The Knative tutorial with Gloo also showed unexpected behaviors after the first couple steps.

H.1. Knative with Istio

The installation of Knative with Isio has failed. In the following subsection, the test protocol is provided.

H.1.1. Requirement

Despite the fact that they say Knative Eventing could be used without Knative, the installation guide requires to install Knative Serving, which use either Istio (Istio is a service mesh with an ingress) or Gloo (a lightweight gateway for Knative).[50, 195]

- A Kubernetes cluster, here minikube with virtualbox driver (`minikube start --vm-driver virtualbox`)
- kubectl (Kubernetes CLI)
- go (and go home have to be in the path `export PATH=$PATH:$(go env GOPATH)/bin`)

H.1.2. Environment

```
> uname -srv
Linux 4.19.69-1-MANJARO #1 SMP PREEMPT Thu Aug 29 08:51:46 UTC 2019
> vboxmanage --version
6.0.10r132055
> go version
go version go1.13.1 linux/amd64
>  minikube version
minikube version: v1.4.0
commit: 7969c25a98a018b94ea87d949350f3271e9d64b6
>  docker --version
Docker version 19.03.3-ce, build a872fc2f86
> kubectl version
Client Version: version.Info{Major:"1", Minor:"16", GitVersion:"v1.16.1",
 ↪  GitCommit:"d647ddbd755faf07169599a625faf302ffc34458", GitTreeState:"clean",
 ↪  BuildDate:"2019-10-02T17:01:15Z", GoVersion:"go1.12.10", Compiler:"gc",
 ↪  Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"12", GitVersion:"v1.12.0",
 ↪  GitCommit:"0ed33881dc4355495f623c6f22e7dd0b7632b7c0", GitTreeState:"clean",
 ↪  BuildDate:"2018-09-27T16:55:41Z", GoVersion:"go1.10.4", Compiler:"gc",
 ↪  Platform:"linux/amd64"}
```

H.1.3. Setup

Source

Documentation Install on Minikube | Knative[196]

Commands

1. setup minikube

   ```
   minikube start --memory=10000 --cpus=6 \
   --kubernetes-version=v1.13.0 \
   --vm-driver=virtualbox \
   --disk-size=30g \
   --extra-config=apiserver.enable-admission-plugins="LimitRanger,NamespaceExists,NamespaceL
   ```

   with `--memory=8192` as showen in the tutorial was not working stable, with constant CPU load of 95% on a 4 core desktop computer with 4.3 GHz.

2. install Istio

   a) setup the installation[197]

      i. download the release `curl -L https://git.io/getLatestIstio | ISTIO_VERSION=1.3.2 sh -`

      ii. `cd istio-1.3.2`

      iii. `export PATH=$PWD/bin:$PATH`

      iv. `istioctl verify-install`
      this firstly get an error because Istio 1.3.2 require at least Kubernetes v1.13, the Knative tutorial propose Kubernetes v1.12

   b) install Istio[192]

      i. install the Custom Resource Definitions (CRD)s `for i in install/kubernetes/helm/istio-ini`

      ii. install the mutual Transport Layer Security (TLS) mode `kubectl apply -f install/kubernetes/`

      iii. verify the installation with `kubectl get svc -n istio-system` and `kubectl get pods -n istio`

3. install the CRDs

   ```
   kubectl apply --selector knative.dev/crd-install=true \
   --filename
    ↪ https://github.com/knative/serving/releases/download/v0.9.0/serving.yaml \
   --filename
    ↪ https://github.com/knative/eventing/releases/download/v0.9.0/release.yaml \
   --filename
    ↪ https://github.com/knative/serving/releases/download/v0.9.0/monitoring.yaml
   ```

4. install services

   ```
   kubectl apply --filename
    ↪ https://github.com/knative/serving/releases/download/v0.9.0/serving.yaml
    ↪ \
   --filename
    ↪ https://github.com/knative/eventing/releases/download/v0.9.0/release.yaml
    ↪ \
   --filename
    ↪ https://github.com/knative/serving/releases/download/v0.9.0/monitoring.yaml
   ```

5. wait for services to be ready `kubectl get pods --all-namespaces -w`

H.1.4. Example

Documentation partly from Container Source Example | Knative[76], partly from Broker and Trigger | Knative[52]

1. `kubectl label namespace default knative-eventing-injection=enabled`
2. `kubectl -n default get broker default`
3.
```
cat << EOF | kubectl apply -f -
  apiVersion: serving.knative.dev/v1alpha1
  kind: Service
  metadata:
    name: my-service
    namespace: default
  spec:
    template:
      spec:
        containers:
        -  # This corresponds to
           #
            ↪  https://github.com/knative/eventing-contrib/blob/v0.2.1/cmd/message_dumper/
           image:
            ↪  gcr.io/knative-releases/github.com/knative/eventing-sources/cmd/message_du

  ---

  apiVersion: eventing.knative.dev/v1alpha1
  kind: Trigger
  metadata:
    name: my-service-trigger
    namespace: default
  spec:
    filter:
      attributes:
        type: dev.knative.foo.bar
    subscriber:
      ref:
        apiVersion: serving.knative.dev/v1alpha1
        kind: Service
        name: my-service

  ---

  apiVersion: eventing.knative.dev/v1alpha1
  kind: Trigger
  metadata:
    name: my-service-trigger
    namespace: default
  spec:
    broker: default # Defaulted by the Webhook.
    filter:
```

```
      attributes:
        type: dev.knative.foo.bar
    subscriber:
      ref:
        apiVersion: serving.knative.dev/v1alpha1
        kind: Service
        name: my-service

---

apiVersion: sources.eventing.knative.dev/v1alpha1
kind: ContainerSource
metadata:
  name: heartbeats-sender
spec:
  template:
    spec:
      containers:
        - image: github.com/knative/eventing-contrib/cmd/heartbeats/
          name: heartbeats-sender
          args:
            - --eventType=dev.knative.foo.bar
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: POD_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
  sink:
    apiVersion: eventing.knative.dev/v1alpha1
    kind: Broker
    name: default
EOF
```

4. event is logged `kubectl logs eventing-controller-564985795d-vncj4 -n knative-eventing`

   but not in the Service `kubectl logs -l eventing.knative.dev/my-service`

H.2. Knative with Gloo

We also started the Knative tutorial with Gloo, but also this one does not seem to be well maintained. The commands don't work with the newest versions.

H.2.1. Requirement

- A Kubernetes cluster, here minikube with virtualbox driver (`minikube start --vm-driver virtualbox`)
- kubectl (Kubernetes CLI)

- go (and go home have to be in the path `export PATH=$PATH:$(go env GOPATH)/bin`)

H.2.2. Environment

```
> uname -srv
Linux 4.19.84-1-MANJARO #1 SMP PREEMPT Wed Nov 13 00:07:37 UTC 2019
> vboxmanage --version
6.0.14r132055
> go version
go version go1.13.5 linux/amd64
>  minikube version
minikube version: v1.5.2
commit: 792dbf92a1de583fcee76f8791cff12e0c9440ad
>  docker --version
Docker version 19.03.4-ce, build 9013bf583a
> kubectl version
Client Version: version.Info{Major:"1", Minor:"17", GitVersion:"v1.17.0",
 ↪  GitCommit:"70132b0f130acc0bed193d9ba59dd186f0e634cf", GitTreeState:"clean",
 ↪  BuildDate:"2019-12-07T21:20:10Z", GoVersion:"go1.13.4", Compiler:"gc",
 ↪  Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"16", GitVersion:"v1.16.2",
 ↪  GitCommit:"c97fe5036ef3df2967d086711e6c0c405941e14b", GitTreeState:"clean",
 ↪  BuildDate:"2019-10-15T19:09:08Z", GoVersion:"go1.12.10", Compiler:"gc",
 ↪  Platform:"linux/amd64"}
```

H.2.3. Setup

Source

Documentation Knative Install using Gloo on a Kubernetes Cluster[198]

Commands

1. install glooctl `curl -sL https://run.solo.io/gloo/install | sh`
2. add it to the path `export PATH=$HOME/.gloo/bin:$PATH`
3. verify the installation the tutorial documents the following command: `glooctl --version`, which does not exist – it is `glooctl version`
4. install Knative `glooctl install knative` the provided command has a timeout error

I. Protocol: Apache Camel Dependency

This protocol show how the dependency tree for Apache Camel was generated.

I.1. Requirement

- java JDK
- Maven

I.2. Environment

```
[H]
> uname -srv
Linux 4.19.69-1-MANJARO #1 SMP PREEMPT Thu Aug 29 08:51:46 UTC 2019
> java --version
openjdk 12.0.2 2019-07-16
OpenJDK Runtime Environment (build 12.0.2+10)
OpenJDK 64-Bit Server VM (build 12.0.2+10, mixed mode)
> mvn --version
Apache Maven 3.6.1 (NON-CANONICAL_2019-07-24T20:49:02Z_root; 2019-07-24T22:49:02+02:00)
Maven home: /opt/maven
Java version: 12.0.2, vendor: N/A, runtime: /usr/lib/jvm/java-12-openjdk
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.19.69-1-manjaro", arch: "amd64", family: "unix"
```

I.3. Step by step

1. Create a Maven Project with the pom.xml in Listing 5
2. compile with `mvn compile`
3. execute `mvn dependency:tree` to get the dependencies, shown in Listing 6

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
          ↪  http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>apache-camel-dependency</groupId>
    <artifactId>com.ce-rust</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.apache.camel</groupId>
            <artifactId>camel-core</artifactId>
            <version>2.24.2</version>
        </dependency>
    </dependencies>
</project>
```

Listing 5: pom.xml for dependency resolution of Apache Camel Core

```
> mvn dependency:tree
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------< apache-camel-dependency:com.ce-rust >-----------------
[INFO] Building com.ce-rust 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-dependency-plugin:2.8:tree (default-cli) @ com.ce-rust ---
[INFO] apache-camel-dependency:com.ce-rust:jar:1.0-SNAPSHOT
[INFO] \- org.apache.camel:camel-core:jar:2.24.2:compile
[INFO]    +- org.slf4j:slf4j-api:jar:1.7.26:compile
[INFO]    +- com.sun.xml.bind:jaxb-core:jar:2.3.0:compile
[INFO]    \- com.sun.xml.bind:jaxb-impl:jar:2.3.0:compile
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.792 s
[INFO] Finished at: 2019-10-16T08:40:43+02:00
[INFO] ------------------------------------------------------------------------
```

Listing 6: command and output for dependency resolution of Apache Camel Core

J. Protocol: Node-RED Dependency

This protocol show how the dependency tree for Node-RED was generated.

J.1. Requirement

- Node.js with npm

J.2. Environment

```
[H]
> uname -srv
Linux 4.19.69-1-MANJARO #1 SMP PREEMPT Thu Aug 29 08:51:46 UTC 2019
> node -v
v11.12.0
> npm -v
6.7.0
```

J.3. Step by step

1. Create a Folder with the package.json in Listing 7
2. compile with `npm i`
3. execute `npm ls --prod` to get the dependencies, shown in Listing 8

```
{
  "name": "node-red-dependency",
  "version": "1.0.0",
  "dependencies": {
    "node-red": "1.0.2"
  }
}
```

Listing 7: package.json for dependency resolution of Node-RED

```
node-red-dependency@1.0.0 /home/fab/hsr/ce-rust/node-red-dependency
`-- node-red@1.0.2
  +-- @node-red/editor-api@1.0.2
  | +-- @node-red/editor-client@1.0.2
  | +-- @node-red/util@1.0.2 deduped
  | +-- bcrypt@3.0.6 deduped
  | +-- bcryptjs@2.4.3 deduped
  | +-- body-parser@1.19.0
```

```
| | +-- bytes@3.1.0
| | +-- content-type@1.0.4 deduped
| | +-- debug@2.6.9
| | | `-- ms@2.0.0
| | +-- depd@1.1.2 deduped
| | +-- http-errors@1.7.2
| | | +-- depd@1.1.2 deduped
| | | +-- inherits@2.0.3
| | | +-- setprototypeof@1.1.1 deduped
| | | +-- statuses@1.5.0 deduped
| | | `-- toidentifier@1.0.0
| | +-- iconv-lite@0.4.24
| | | `-- safer-buffer@2.1.2 deduped
| | +-- on-finished@2.3.0 deduped
| | +-- qs@6.7.0 deduped
| | +-- raw-body@2.4.0
| | | +-- bytes@3.1.0 deduped
| | | +-- http-errors@1.7.2 deduped
| | | +-- iconv-lite@0.4.24 deduped
| | | `-- unpipe@1.0.0 deduped
| | `-- type-is@1.6.18 deduped
| +-- clone@2.1.2
| +-- cors@2.8.5
| | +-- object-assign@4.1.1
| | `-- vary@1.1.2 deduped
| +-- express@4.17.1 deduped
| +-- express-session@1.16.2
| | +-- cookie@0.3.1
| | +-- cookie-signature@1.0.6 deduped
| | +-- debug@2.6.9
| | | `-- ms@2.0.0
| | +-- depd@2.0.0
| | +-- on-headers@1.0.2 deduped
| | +-- parseurl@1.3.3 deduped
| | +-- safe-buffer@5.1.2 deduped
| | `-- uid-safe@2.1.5
| |   `-- random-bytes@1.0.0
| +-- memorystore@1.6.1
| | +-- debug@3.1.0
| | | `-- ms@2.0.0
| | `-- lru-cache@4.1.5
| |   +-- pseudomap@1.0.2
| |   `-- yallist@2.1.2
| +-- mime@2.4.4
| +-- mustache@3.0.2
| +-- oauth2orize@1.11.0
| | +-- debug@2.6.9
| | | `-- ms@2.0.0
| | +-- uid2@0.0.3
| | `-- utils-merge@1.0.1 deduped
| +-- passport@0.4.0
| | +-- passport-strategy@1.0.0
| | `-- pause@0.0.1
```

```
| +-- passport-http-bearer@1.0.1
| | `-- passport-strategy@1.0.0 deduped
| +-- passport-oauth2-client-password@0.1.2
| | `-- passport-strategy@1.0.0 deduped
| +-- when@3.7.8
| `-- ws@6.2.1
|   `-- async-limiter@1.0.1
+-- @node-red/nodes@1.0.2
| +-- ajv@6.10.2
| | +-- fast-deep-equal@2.0.1
| | +-- fast-json-stable-stringify@2.0.0
| | +-- json-schema-traverse@0.4.1
| | `-- uri-js@4.2.2
| |   `-- punycode@2.1.1
| +-- body-parser@1.19.0 deduped
| +-- cheerio@0.22.0
| | +-- css-select@1.2.0
| | | +-- boolbase@1.0.0
| | | +-- css-what@2.1.3
| | | +-- domutils@1.5.1
| | | | +-- dom-serializer@0.1.1 deduped
| | | | `-- domelementtype@1.3.1 deduped
| | | `-- nth-check@1.0.2
| | |   `-- boolbase@1.0.0 deduped
| | +-- dom-serializer@0.1.1
| | | +-- domelementtype@1.3.1
| | | `-- entities@1.1.2 deduped
| | +-- entities@1.1.2
| | +-- htmlparser2@3.10.1
| | | +-- domelementtype@1.3.1 deduped
| | | +-- domhandler@2.4.2
| | | | `-- domelementtype@1.3.1 deduped
| | | +-- domutils@1.5.1 deduped
| | | +-- entities@1.1.2 deduped
| | | +-- inherits@2.0.4 deduped
| | | `-- readable-stream@3.4.0
| | |   +-- inherits@2.0.4 deduped
| | |   +-- string_decoder@1.1.1 deduped
| | |   `-- util-deprecate@1.0.2 deduped
| | +-- lodash.assignin@4.2.0
| | +-- lodash.bind@4.2.1
| | +-- lodash.defaults@4.2.0
| | +-- lodash.filter@4.6.0
| | +-- lodash.flatten@4.4.0
| | +-- lodash.foreach@4.5.0
| | +-- lodash.map@4.6.0
| | +-- lodash.merge@4.6.2
| | +-- lodash.pick@4.4.0
| | +-- lodash.reduce@4.6.0
| | +-- lodash.reject@4.6.0
| | `-- lodash.some@4.6.0
| +-- content-type@1.0.4
| +-- cookie@0.4.0
```

```
| +-- cookie-parser@1.4.4
| | +-- cookie@0.3.1
| | `-- cookie-signature@1.0.6 deduped
| +-- cors@2.8.5 deduped
| +-- cron@1.7.1
| | `-- moment-timezone@0.5.27
| |   `-- moment@2.24.0
| +-- denque@1.4.1
| +-- fs-extra@8.1.0 deduped
| +-- fs.notify@0.0.4
| | +-- async@0.1.22
| | `-- retry@0.6.1
| +-- hash-sum@2.0.0
| +-- https-proxy-agent@2.2.2
| | +-- agent-base@4.3.0
| | | `-- es6-promisify@5.0.0
| | |   `-- es6-promise@4.2.8
| | `-- debug@3.2.6
| |   `-- ms@2.1.2
| +-- iconv-lite@0.5.0
| | `-- safer-buffer@2.1.2
| +-- is-utf8@0.2.1
| +-- js-yaml@3.13.1
| | +-- argparse@1.0.10
| | | `-- sprintf-js@1.0.3
| | `-- esprima@4.0.1
| +-- media-typer@1.1.0
| +-- mqtt@2.18.8
| | +-- commist@1.1.0
| | | +-- leven@2.1.0
| | | `-- minimist@1.2.0
| | +-- concat-stream@1.6.2
| | | +-- buffer-from@1.1.1
| | | +-- inherits@2.0.4 deduped
| | | +-- readable-stream@2.3.6 deduped
| | | `-- typedarray@0.0.6
| | +-- end-of-stream@1.4.4
| | | `-- once@1.4.0
| | |   `-- wrappy@1.0.2
| | +-- es6-map@0.1.5
| | | +-- d@1.0.1
| | | | +-- es5-ext@0.10.51 deduped
| | | | `-- type@1.2.0
| | | +-- es5-ext@0.10.51
| | | | +-- es6-iterator@2.0.3 deduped
| | | | +-- es6-symbol@3.1.2 deduped
| | | | `-- next-tick@1.0.0
| | | +-- es6-iterator@2.0.3
| | | | +-- d@1.0.1 deduped
| | | | +-- es5-ext@0.10.51 deduped
| | | | `-- es6-symbol@3.1.2 deduped
| | | +-- es6-set@0.1.5
| | | | +-- d@1.0.1 deduped
```

```
| | | | +-- es5-ext@0.10.51 deduped
| | | | +-- es6-iterator@2.0.3 deduped
| | | | +-- es6-symbol@3.1.1
| | | | | +-- d@1.0.1 deduped
| | | | | `-- es5-ext@0.10.51 deduped
| | | | `-- event-emitter@0.3.5 deduped
| | | +-- es6-symbol@3.1.2
| | | | +-- d@1.0.1 deduped
| | | | `-- es5-ext@0.10.51 deduped
| | | `-- event-emitter@0.3.5
| | |   +-- d@1.0.1 deduped
| | |   `-- es5-ext@0.10.51 deduped
| | +-- help-me@1.1.0
| | | +-- callback-stream@1.1.0
| | | | +-- inherits@2.0.4 deduped
| | | | `-- readable-stream@2.3.6 deduped
| | | +-- glob-stream@6.1.0
| | | | +-- extend@3.0.2 deduped
| | | | +-- glob@7.1.4 deduped
| | | | +-- glob-parent@3.1.0
| | | | | +-- is-glob@3.1.0
| | | | | | `-- is-extglob@2.1.1
| | | | | `-- path-dirname@1.0.2
| | | | +-- is-negated-glob@1.0.0
| | | | +-- ordered-read-streams@1.0.1
| | | | | `-- readable-stream@2.3.6 deduped
| | | | +-- pumpify@1.5.1
| | | | | +-- duplexify@3.7.1 deduped
| | | | | +-- inherits@2.0.4 deduped
| | | | | `-- pump@2.0.1
| | | | |   +-- end-of-stream@1.4.4 deduped
| | | | |   `-- once@1.4.0 deduped
| | | | +-- readable-stream@2.3.6 deduped
| | | | +-- remove-trailing-separator@1.1.0
| | | | +-- to-absolute-glob@2.0.2
| | | | | +-- is-absolute@1.0.0
| | | | | | +-- is-relative@1.0.0
| | | | | | | `-- is-unc-path@1.0.0
| | | | | | |   `-- unc-path-regex@0.1.2
| | | | | | `-- is-windows@1.0.2
| | | | | `-- is-negated-glob@1.0.0 deduped
| | | | `-- unique-stream@2.3.1
| | | |   +-- json-stable-stringify-without-jsonify@1.0.1
| | | |   `-- through2-filter@3.0.0
| | | |     +-- through2@2.0.5 deduped
| | | |     `-- xtend@4.0.2 deduped
| | | +-- through2@2.0.5
| | | | +-- readable-stream@2.3.6 deduped
| | | | `-- xtend@4.0.2 deduped
| | | `-- xtend@4.0.2 deduped
| | +-- inherits@2.0.4
| | +-- minimist@1.2.0
| | +-- mqtt-packet@5.6.1
```

```
| | | +-- bl@1.2.2
| | | | +-- readable-stream@2.3.6 deduped
| | | | `-- safe-buffer@5.1.2 deduped
| | | +-- inherits@2.0.4 deduped
| | | +-- process-nextick-args@2.0.1
| | | `-- safe-buffer@5.1.2 deduped
| | +-- pump@3.0.0
| | | +-- end-of-stream@1.4.4 deduped
| | | `-- once@1.4.0 deduped
| | +-- readable-stream@2.3.6
| | | +-- core-util-is@1.0.2
| | | +-- inherits@2.0.4 deduped
| | | +-- isarray@1.0.0
| | | +-- process-nextick-args@2.0.1 deduped
| | | +-- safe-buffer@5.1.2 deduped
| | | +-- string_decoder@1.1.1
| | | | `-- safe-buffer@5.1.2 deduped
| | | `-- util-deprecate@1.0.2
| | +-- reinterval@1.1.0
| | +-- split2@2.2.0
| | | `-- through2@2.0.5 deduped
| | +-- websocket-stream@5.5.0
| | | +-- duplexify@3.7.1
| | | | +-- end-of-stream@1.4.4 deduped
| | | | +-- inherits@2.0.4 deduped
| | | | +-- readable-stream@2.3.6 deduped
| | | | `-- stream-shift@1.0.0
| | | +-- inherits@2.0.4 deduped
| | | +-- readable-stream@2.3.6 deduped
| | | +-- safe-buffer@5.1.2 deduped
| | | +-- ws@3.3.3
| | | | +-- async-limiter@1.0.1 deduped
| | | | +-- safe-buffer@5.1.2 deduped
| | | | `-- ultron@1.1.1
| | | `-- xtend@4.0.2 deduped
| | `-- xtend@4.0.2
| +-- multer@1.4.2
| | +-- append-field@1.0.0
| | +-- busboy@0.2.14
| | | +-- dicer@0.2.5
| | | | +-- readable-stream@1.1.14
| | | | | +-- core-util-is@1.0.2 deduped
| | | | | +-- inherits@2.0.4 deduped
| | | | | +-- isarray@0.0.1
| | | | | `-- string_decoder@0.10.31
| | | | `-- streamsearch@0.1.2
| | | `-- readable-stream@1.1.14
| | |   +-- core-util-is@1.0.2 deduped
| | |   +-- inherits@2.0.4 deduped
| | |   +-- isarray@0.0.1
| | |   `-- string_decoder@0.10.31
| | +-- concat-stream@1.6.2 deduped
| | +-- mkdirp@0.5.1
```

```
| | | `-- minimist@0.0.8
| | +-- object-assign@4.1.1 deduped
| | +-- on-finished@2.3.0 deduped
| | +-- type-is@1.6.18 deduped
| | `-- xtend@4.0.2 deduped
| +-- mustache@3.0.2 deduped
| +-- on-headers@1.0.2
| +-- raw-body@2.4.1
| | +-- bytes@3.1.0 deduped
| | +-- http-errors@1.7.3
| | | +-- depd@1.1.2 deduped
| | | +-- inherits@2.0.4 deduped
| | | +-- setprototypeof@1.1.1 deduped
| | | +-- statuses@1.5.0 deduped
| | | `-- toidentifier@1.0.0 deduped
| | +-- iconv-lite@0.4.24
| | | `-- safer-buffer@2.1.2 deduped
| | `-- unpipe@1.0.0
| +-- request@2.88.0
| | +-- aws-sign2@0.7.0
| | +-- aws4@1.8.0
| | +-- caseless@0.12.0
| | +-- combined-stream@1.0.8
| | | `-- delayed-stream@1.0.0
| | +-- extend@3.0.2
| | +-- forever-agent@0.6.1
| | +-- form-data@2.3.3
| | | +-- asynckit@0.4.0
| | | +-- combined-stream@1.0.8 deduped
| | | `-- mime-types@2.1.24 deduped
| | +-- har-validator@5.1.3
| | | +-- ajv@6.10.2 deduped
| | | `-- har-schema@2.0.0
| | +-- http-signature@1.2.0
| | | +-- assert-plus@1.0.0
| | | +-- jsprim@1.4.1
| | | | +-- assert-plus@1.0.0 deduped
| | | | +-- extsprintf@1.3.0
| | | | +-- json-schema@0.2.3
| | | | `-- verror@1.10.0
| | | |   +-- assert-plus@1.0.0 deduped
| | | |   +-- core-util-is@1.0.2 deduped
| | | |   `-- extsprintf@1.3.0 deduped
| | | `-- sshpk@1.16.1
| | |   +-- asn1@0.2.4
| | |   | `-- safer-buffer@2.1.2 deduped
| | |   +-- assert-plus@1.0.0 deduped
| | |   +-- bcrypt-pbkdf@1.0.2
| | |   | `-- tweetnacl@0.14.5 deduped
| | |   +-- dashdash@1.14.1
| | |   | `-- assert-plus@1.0.0 deduped
| | |   +-- ecc-jsbn@0.1.2
| | |   | +-- jsbn@0.1.1 deduped
```

```
| | |   | `-- safer-buffer@2.1.2 deduped
| | |   +-- getpass@0.1.7
| | |   | `-- assert-plus@1.0.0 deduped
| | |   +-- jsbn@0.1.1
| | |   +-- safer-buffer@2.1.2 deduped
| | |   `-- tweetnacl@0.14.5
| | +-- is-typedarray@1.0.0
| | +-- isstream@0.1.2
| | +-- json-stringify-safe@5.0.1 deduped
| | +-- mime-types@2.1.24
| | | `-- mime-db@1.40.0
| | +-- oauth-sign@0.9.0
| | +-- performance-now@2.1.0
| | +-- qs@6.5.2
| | +-- safe-buffer@5.1.2 deduped
| | +-- tough-cookie@2.4.3
| | | +-- psl@1.4.0
| | | `-- punycode@1.4.1
| | +-- tunnel-agent@0.6.0
| | | `-- safe-buffer@5.1.2 deduped
| | `-- uuid@3.3.3
| +-- ws@6.2.1 deduped
| `-- xml2js@0.4.19
|   +-- sax@1.2.4
|   `-- xmlbuilder@9.0.7
+-- @node-red/runtime@1.0.2
| +-- @node-red/registry@1.0.2
| | +-- @node-red/util@1.0.2 deduped
| | +-- semver@6.3.0 deduped
| | +-- uglify-js@3.6.0
| | | +-- commander@2.20.3
| | | `-- source-map@0.6.1
| | `-- when@3.7.8 deduped
| +-- @node-red/util@1.0.2 deduped
| +-- clone@2.1.2 deduped
| +-- express@4.17.1 deduped
| +-- fs-extra@8.1.0 deduped
| +-- json-stringify-safe@5.0.1
| `-- when@3.7.8 deduped
+-- @node-red/util@1.0.2
| +-- clone@2.1.2 deduped
| +-- i18next@15.1.2
| | `-- @babel/runtime@7.6.3
| |   `-- regenerator-runtime@0.13.3
| +-- json-stringify-safe@5.0.1 deduped
| +-- jsonata@1.6.5
| `-- when@3.7.8 deduped
+-- basic-auth@2.0.1
| `-- safe-buffer@5.1.2
+-- bcrypt@3.0.6
| +-- nan@2.13.2
| `-- node-pre-gyp@0.12.0
|   +-- detect-libc@1.0.3
```

```
|   +-- mkdirp@0.5.1 deduped
|   +-- needle@2.4.0
|   | +-- debug@3.2.6 deduped
|   | +-- iconv-lite@0.4.24 deduped
|   | `-- sax@1.2.4 deduped
|   +-- nopt@4.0.1 deduped
|   +-- npm-packlist@1.4.6
|   | +-- ignore-walk@3.0.3
|   | | `-- minimatch@3.0.4
|   | |   `-- brace-expansion@1.1.11
|   | |     +-- balanced-match@1.0.0
|   | |     `-- concat-map@0.0.1
|   | `-- npm-bundled@1.0.6
|   +-- npmlog@4.1.2
|   | +-- are-we-there-yet@1.1.5
|   | | +-- delegates@1.0.0
|   | | `-- readable-stream@2.3.6 deduped
|   | +-- console-control-strings@1.1.0
|   | +-- gauge@2.7.4
|   | | +-- aproba@1.2.0
|   | | +-- console-control-strings@1.1.0 deduped
|   | | +-- has-unicode@2.0.1
|   | | +-- object-assign@4.1.1 deduped
|   | | +-- signal-exit@3.0.2
|   | | +-- string-width@1.0.2
|   | | | +-- code-point-at@1.1.0
|   | | | +-- is-fullwidth-code-point@1.0.0
|   | | | | `-- number-is-nan@1.0.1
|   | | | `-- strip-ansi@3.0.1 deduped
|   | | +-- strip-ansi@3.0.1
|   | | | `-- ansi-regex@2.1.1
|   | | `-- wide-align@1.1.3
|   | |   `-- string-width@1.0.2 deduped
|   | `-- set-blocking@2.0.0
|   +-- rc@1.2.8
|   | +-- deep-extend@0.6.0
|   | +-- ini@1.3.5
|   | +-- minimist@1.2.0
|   | `-- strip-json-comments@2.0.1
|   +-- rimraf@2.7.1
|   | `-- glob@7.1.4
|   |   +-- fs.realpath@1.0.0
|   |   +-- inflight@1.0.6
|   |   | +-- once@1.4.0 deduped
|   |   | `-- wrappy@1.0.2 deduped
|   |   +-- inherits@2.0.4 deduped
|   |   +-- minimatch@3.0.4 deduped
|   |   +-- once@1.4.0 deduped
|   |   `-- path-is-absolute@1.0.1
|   +-- semver@5.7.1
|   `-- tar@4.4.13
|     +-- chownr@1.1.3
|     +-- fs-minipass@1.2.7
```

```
|     | `-- minipass@2.9.0 deduped
|     +-- minipass@2.9.0
|     | +-- safe-buffer@5.1.2 deduped
|     | `-- yallist@3.1.1 deduped
|     +-- minizlib@1.3.3
|     | `-- minipass@2.9.0 deduped
|     +-- mkdirp@0.5.1 deduped
|     +-- safe-buffer@5.1.2 deduped
|     `-- yallist@3.1.1
+-- bcryptjs@2.4.3
+-- express@4.17.1
| +-- accepts@1.3.7
| | +-- mime-types@2.1.24 deduped
| | `-- negotiator@0.6.2
| +-- array-flatten@1.1.1
| +-- body-parser@1.19.0 deduped
| +-- content-disposition@0.5.3
| | `-- safe-buffer@5.1.2 deduped
| +-- content-type@1.0.4 deduped
| +-- cookie@0.4.0 deduped
| +-- cookie-signature@1.0.6
| +-- debug@2.6.9
| | `-- ms@2.0.0
| +-- depd@1.1.2
| +-- encodeurl@1.0.2
| +-- escape-html@1.0.3
| +-- etag@1.8.1
| +-- finalhandler@1.1.2
| | +-- debug@2.6.9
| | | `-- ms@2.0.0
| | +-- encodeurl@1.0.2 deduped
| | +-- escape-html@1.0.3 deduped
| | +-- on-finished@2.3.0 deduped
| | +-- parseurl@1.3.3 deduped
| | +-- statuses@1.5.0 deduped
| | `-- unpipe@1.0.0 deduped
| +-- fresh@0.5.2
| +-- merge-descriptors@1.0.1
| +-- methods@1.1.2
| +-- on-finished@2.3.0
| | `-- ee-first@1.1.1
| +-- parseurl@1.3.3
| +-- path-to-regexp@0.1.7
| +-- proxy-addr@2.0.5
| | +-- forwarded@0.1.2
| | `-- ipaddr.js@1.9.0
| +-- qs@6.7.0
| +-- range-parser@1.2.1
| +-- safe-buffer@5.1.2 deduped
| +-- send@0.17.1
| | +-- debug@2.6.9
| | | `-- ms@2.0.0
| | +-- depd@1.1.2 deduped
```

```
| | +-- destroy@1.0.4
| | +-- encodeurl@1.0.2 deduped
| | +-- escape-html@1.0.3 deduped
| | +-- etag@1.8.1 deduped
| | +-- fresh@0.5.2 deduped
| | +-- http-errors@1.7.2 deduped
| | +-- mime@1.6.0
| | +-- ms@2.1.1
| | +-- on-finished@2.3.0 deduped
| | +-- range-parser@1.2.1 deduped
| | `-- statuses@1.5.0 deduped
| +-- serve-static@1.14.1
| | +-- encodeurl@1.0.2 deduped
| | +-- escape-html@1.0.3 deduped
| | +-- parseurl@1.3.3 deduped
| | `-- send@0.17.1 deduped
| +-- setprototypeof@1.1.1
| +-- statuses@1.5.0
| +-- type-is@1.6.18
| | +-- media-typer@0.3.0
| | `-- mime-types@2.1.24 deduped
| +-- utils-merge@1.0.1
| `-- vary@1.1.2
+-- fs-extra@8.1.0
| +-- graceful-fs@4.2.2
| +-- jsonfile@4.0.0
| | `-- graceful-fs@4.2.2 deduped
| `-- universalify@0.1.2
+-- node-red-node-rbe@0.2.5
+-- node-red-node-tail@0.0.3
| `-- tail@2.0.3
+-- nopt@4.0.1
| +-- abbrev@1.1.1
| `-- osenv@0.1.5
|   +-- os-homedir@1.0.2
|   `-- os-tmpdir@1.0.2
`-- semver@6.3.0
```

Listing 8: command and output for dependency resolution of Node-RED, this output was generated by `npm ls --prod --unicode=false`, additionally parameter `--unicode=false` is only to avoid LaTeX problems

K. Performance Test: CERK and Apache Camel

The two products, our CloudEvents Router and Apache Camel were tested on a Linux system.

A Script produces, with help of the Eclipse Mosquitto command-line tool, a high load of CloudEvents and sends them to a topic on the MQTT broker. One of the routers subscribes to the topic, routes the events and sends them back to a different topic. The script measures how long it took to receive all sent events. This process is visualized in Figure K.1. This test is performed for each implementation in isolation.

The MQTT QoS for receiving and sending were set to 2 on both routers to make sure no messages are dropped by the broker.

K.1. Environment

```
[H]
> uname -srv
Linux 4.19.85-1-MANJARO #1 SMP PREEMPT Thu Nov 21 10:38:39 UTC 2019
> docker -version
Docker version 19.03.4-ce, build 9013bf583a
> docker-compose --version
docker-compose version 1.24.1, build unknown
```

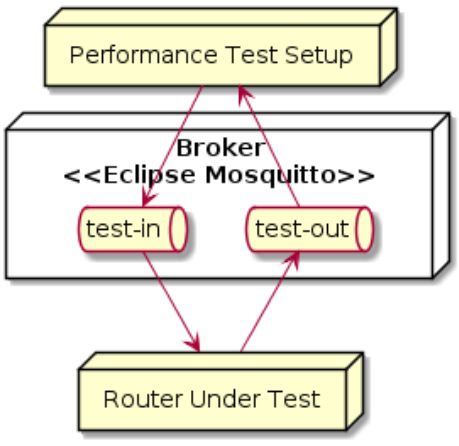

Figure K.1.: Test Setup: The events are generated, sent to a topic and then received from the router. The router forwards the event to another topic and the time of the whole process is measured.

K.2. Results

The test results are provided in Table K.1. Our CloudEvents Router showed much better results when limiting the resources and good results under high load in comparison to the Apache Camel router.

To calculate the pure routing time, without the time the message spends in transit, the column "Time Broker Only" has to be subtracted from the duration of the whole test.

So the CloudEvents Router for example had a throughput of $\frac{1000}{3.6-3.1} = 2000$ messages per second.

Test	Time Broker Only	Time CloudEvents Router	Time Apache Camel
1000 Messages sent and received without any resource limitations	3.1s	3.6s 2000 Msg./s	3.7s 1700 Msg./s
1000 Messages sent and received, resources limited to 1 CPU core and 20 MB of memory	3.1s	3.5s 2500 Msg./s	*crash*[1]
1000 Messages sent and received, resources limited to 1 CPU core and 1 GB of memory	3.1s	3.5s 2500 Msg./s	3.6s 2000 Msg./s
100 Messages sent and received, each message has a size of 64 KB, no resource limitations[2]	1.3s	1.7s 250 Msg./s	6.4s 20 Msg./s

Table K.1.: Performance Test Results: Comparison of the performance results of different test configurations.

K.2.1. Disclaimers on the Setup

In the following subsection lists some facts that blurred the results of the tests.

Different Routing Logic

The Apache Camel setup only takes event and routs them to the other MQTT topic. No validation on the CloudEvents specification is done. The CloudEvents Router deserializes and serializes each event and does validate the event against the CloudEvents specification.

Deployment of the Routers

Apache Camel runs in the official Maven container and gets started with the Maven goal `mvn exec:java`. The use of Maven may increase the required resources of the Camel Router. The router may have a smaller footprint when running it in a java-only container.

The CloudEvents Router was deployed in the resource unlimited test with the official Rust Docker image. For the test with limited resources, the official `debian:stable-slim` Docker image was used.

K.3. Sources

The sources of the tests are hosted on GitHub: https://github.com/ce-rust/performance-test.

[1] The Apache Camel test-setup was not able to start with the given resource limitations
[2] the maximal wire size of a CloudEvents is 64 KB according to the specification [1]