

Bachelorarbeit

Entwurf und Implementierung einer
hochperformanten, serverbasierten
Kommunikationsplattform für Sensordaten
im Umfeld des automatisierten Fahrens in Rust

Michael Watzko

Sommersemester 2018
14.02.2018 - 22.06.2018

Erstprüfer: Prof. Dr. rer. nat. Dipl.-Inform. Manfred Dausmann
Zweitprüfer: ... Hannes Todenhausen



Firma: IT Designers GmbH
Betreuer: Dipl. Ing. (FH) Kevin Erath M.Sc.

Sperrvermerk

U SHALL NOT PASS

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Esslingen, den 15. Februar 2018

Michael Watzko

Danksagungen

“Alle Zitate aus dem Internet sind wahr!”

Albert Einstein

“Rust is a vampire language, it does not reflect at all!”

<https://www.youtube.com/watch?v=-Tj8Q12DaEQ>

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.2	Zielsetzung	1
1.3	Aufbau der Arbeit	1
2	Die Programmiersprache Rust	2
2.1	Geschichte	2
2.2	Zero Cost Abstraction	2
2.3	Was ist Rust?	2
2.4	Sprachfeatures	3
2.4.1	Option	3
2.4.2	Result	3
2.5	Warum Rust?	3
2.6	Kernfeatures	3
2.7	Schwächen	4
2.8	Performance Fallstricke	4
2.9	Aktuelle Verwendung von Rust	4
3	Hochperformant, serverbasierte Kommunikationsplattform ...?	5
3.1	Hochperformant	5
3.2	Serverbasierte Kommunikationsplattform	5
3.3	Low-Latency + Entwurfsmuster + Algorithmen?	5
3.4	ASN.1	5
3.5	PER	5
3.6	MEC-View Server und Umgebung	5
4	Anforderungen	6
4.1	Funktionale Anforderungen	6
4.2	Nichtfunktionale Anforderungen	6
4.3	Kein Protobuf weil	6
5	Systemanalyse	7
5.1	Systemkontextdiagramm	7
5.2	Schnittstellenanalyse	7
5.3	C++ Referenzsystem	7

6 Systementwurf	8
6.1 Änderungen bedingt durch Rust	8
7 Implementierung	9
8 Auswertung	10
9 Zusammenfassung und Fazit	I
Literatur	II
Literatur	II
List of abbreviations	III

1 Einleitung

1.1 Motivation

1.2 Zielsetzung

1.3 Aufbau der Arbeit

2 Die Programmiersprache Rust

Rust ist eine Programmiersprache, die versucht performant – und daher durch Abstraktionen mit keinem zusätzlichen “Kosten” **TODO: ref zero cost abstractions** – sichere Programmierung zu ermöglichen. Ziel ist eine **TODO: Systemprogrammiersprache**, die sowohl sicher **TODO: cite chapter** als auch performant ist und ohne eine Laufzeit ausgeführt werden kann. Verschiedene Fehlerquellen – wie “dangling pointers”, “double free” oder “memory leaks” **TODO: ref** – werden durch Abstraktionen und mit Hilfe des Compilers verhindert. Anders als Programmiersprachen, die dies mit Hilfe einer Laufzeit ermöglichen (zbsp. Java oder C#), wird dies in Rust durch eine statische Analyse und einem Eigentümerprinzip bei der Kompilation gewährleistet.

TODO: type safety language **TODO: no undefined behavior, oreilly**

2.1 Geschichte

2.2 Zero Cost Abstraction

2.3 Was ist Rust?

TODO: functional programming -> no global state, no exceptions, find literature

Rust ist...

TODO: Rust -> MIR -> assembler

TODO: MIR/assemblerbeispiele?

[\[5\]](#)

```
1 fn main() {  
2     println!("Hello World");  
3 }
```

Abbildung 2.1: “Hello World” in Rust

2.4 Sprachfeatures

2.4.1 Option

2.4.2 Result

2.5 Warum Rust?

“[...]Leute, die [...] sichere Programmierung haben wollen, [...] können das bei Rust haben, ohne die [von D] undeterministischen Laufzeiten oder Abstraktionskosten schlucken zu müssen.” [6]

“It’s not bad programmers, it’s that C is a hostile language” (Seite 54, [8])

“I’m thinking that C is actively hostile to writing and maintaining reliable code” (Seite 129, [8])

“[...] Rust makes it safe, and provides nice tools” (Seite 130, [8])

“Rust hilft beim Fehlervermeiden” [4]

“Rust is [...] a language that cares about very tight control” [3]

TODO: unused only rust [1]

2.6 Kernfeatures

<https://www.youtube.com/watch?v=d1uraoHM8Gg>

TODO: no dangling pointers

TODO: no need for a runtime, all static analytics

TODO: memory safety

TODO: data-race freedom

TODO: active community

TODO: concurrency: no undefined behavior

TODO: ffi binding Foreign Function Interface¹

TODO: zero cost abstraction

TODO: package manager: cargo

<https://www.youtube.com/watch?v=-Tj8Q12DaEQ>

TODO: static type system with local type inference

¹ Beschreibt den Mechanismus wie ein Programm das in einer Programmiersprache geschrieben ist, Funktionen aufrufen kann, die einer anderen Programmiersprache geschrieben wurden. [2]

TODO: explicit notion of mutability

TODO: zero-cost abstraction *(do not introduce new cost through implementation of abstraction)

TODO: errors are values not exceptions TODO: no null

TODO: static automatic memory management no garbage collection

TODO: often compared to GO and D (44min)

2.7 Schwächen

<https://www.youtube.com/watch?v=-Tj8Q12DaEQ>

TODO: compile-times

TODO: Rust is a vampire language, it does not reflect at all!

TODO: depending on the field -> majority of libraries?

2.8 Performance Fallstricke

TODO: [7]

2.9 Aktuelle Verwendung von Rust

TODO: firefox

<https://www.youtube.com/watch?v=-Tj8Q12DaEQ>

TODO: GTK binding heavily to rust

3 Hochperformant, serverbasierte Kommunikationsplattform ...?

3.1 Hochperformant

3.2 Serverbasierte Kommunikationsplattform

3.3 Low-Latency + Entwurfsmuster + Algorithmen?

3.4 ASN.1

3.5 PER

3.6 MEC-View Server und Umgebung

4 Anforderungen

4.1 Funktionale Anforderungen

4.2 Nichtfunktionale Anforderungen

4.3 Kein Protobuf weil

5 Systemanalyse

5.1 Systemkontextdiagramm

5.2 Schnittstellenanalyse

5.3 C++ Referenzsystem

6 Systementwurf

6.1 Änderungen bedingt durch Rust

7 Implementierung

8 Auswertung

9 Zusammenfassung und Fazit

Literatur

- [1] Jim Blandy. Why Rust? Trustworthy, Concurrent System Programming. Englisch. 2015. URL: <http://www.oreilly.com/programming/free/files/why-rust.pdf> (besucht am 01.06.2017).
- [2] Wikipedia contributors. Foreign function interface — Wikipedia, The Free Encyclopedia. [Online; accessed 14-February-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Foreign_function_interface&oldid=825105351.
- [3] fgilcher. Subreddit Rust. fgilcher kommentiert. Englisch. 3. Nov. 2017. URL: https://www.reddit.com/r/rust/comments/7amv58/just_started_learning_rust_and_was_wondering_does/dpb9qew/ (besucht am 01.06.2017).
- [4] Sebastian Grüner. “C ist eine feindselige Sprache”. Der Mitbegründer des Gnome-Projekts Federico Mena Quintero. Deutsch. 22. Juni 2017. URL: <https://www.golem.de/news/rust-c-ist-eine-feindselige-sprache-1707-129196.html> (besucht am 14.02.2018).
- [5] Jason Orendorff Jim Blandy. Programming Rust. Fast, Safe Systems Development. O'Reilly Media, Dez. 2017. ISBN: 1491927283.
- [6] Felix von Leitner. Fefes Blog. D soll Teil von gcc werden. Deutsch. 22. Juni 2017. URL: <https://blog.fefe.de/?ts=a7b51cac> (besucht am 14.02.2018).
- [7] Llogiq. Llogiq on stuff. Rust Performance Pitfalls. Englisch. URL: <https://llogiq.github.io/2017/06/01/perf-pitfalls.html> (besucht am 01.06.2017).
- [8] Federico Mena Quintero. Replacing C library code with Rust. What I learned with librsvg. Englisch. URL: <https://people.gnome.org/~federico/blog/docs/fmq-porting-c-to-rust.pdf> (besucht am 14.02.2018).

Glossar

Foreign Function Interface Beschreibt den Mechanismus wie ein Programm das in einer Programmiersprache geschrieben ist, Funktionen aufrufen kann, die einer einer anderen Programmiersprache geschrieben wurden. [\[2\]](#) . [3](#)

Abbildungsverzeichnis

2.1 “Hello World” in Rust	2
-------------------------------------	---