HAW HAMBURG

AUSARBEITUNG
Tristan Ropers

Lamports Algorithmus für verteilten gegenseitigen Ausschluss

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Tristan Ropers

Lamports Algorithmus für verteilten gegenseitigen Ausschluss

Ausarbeitung eingereicht im Rahmen des Moduls "Verteilte Systeme"
im Studiengang *Bachelor of Science Technische Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Martin Becke

Eingereicht am: 31. März 2021

Tristan Ropers

Thema der Arbeit

Lamports Algorithmus für verteilten gegenseitigen Ausschluss

Stichworte

Gegenseitiger Ausschluss, Lamport, Verteilte Systeme

Kurzzusammenfassung

Das Ziel der vorliegenden Arbeit ist die Umsetzung des Lamport-Algorithmus [2] für wechselseitigen Ausschluss in einem verteilten System mit anschließender Bewertung der Leistungsfähigkeit des Algorithmus. Für die Umsetzung und Evlauierung des Algorithmus wurde eine RPC-Architektur entwickelt und der Lamport-Algorithmus in diese eingebettet. Im Anschluss wird die Umsetzung hinsichtlich ihrer Leistung evaluiert und die Ergebnisse diskutiert.

Tristan Ropers

Title of Thesis

Mutual exclusion in distributed systems using Lamports algorithm

Keywords

Mutual exclusion, Lamport, distributed systems

Abstract

The goal of this assignment is the implementatin of the lamports algorithm [2] for mutual exclusion in distributed systems with an evaluation of said algorithm. In order to achieve this, a RPC-architecture has been implemented in which the lamports algorithm was embedded. After that, the worked out solution will be tested regarding performance and discussed afterwards.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

In dieser Ausarbeitung geht es um die Umsetzung eines Algorithmus zum gegenseitigen Ausschlusses in einem verteilten System von Schweißrobotern. Das Problem des wechselseitigen Ausschlusses ist ein weit verbreitetes Problem der Informatik. In diesem konkreten Fall sollen mehrere Schweißroboter auf eine Ressource zugreifen können und dort einen Schweißpunkt setzen, was eine Regelung der Reihenfolge erfordert, da die Roboter nicht alle gleichzeitig auf die gegebene Ressource zugreifen können. Um diese Reihenfolge festzulegen wird Lamports Algorithmus [2] zum gegenseitigen Ausschluss in verteilten Systemen umgesetzt und für diesen Anwendungsfall evaluiert.

1.1 Anforderungsanalyse

1.1.1 Requirements

In der Aufgabenstellung sind Anforderungen an die RPC-Architektur formuliert. Zunächst werden diese analysiert und hier zusammengefasst. Die Anforderungen umfassen die Transparenzziele, Skalierung [5]) sowie Anforderungen an die Umsetzung und Architektur.
Es wurde ein hoher Grad an Transparenz gefordert. Die Transparenzziele umfassen die Access-Transparency, Location-Transparency, Relocation-Transparency, Migration-Transparency, Replication-Transparency, Concurrency-Transparency und Failure-Transparency [5]. Die Skalierung beschränkt sich administrativ auf einen Administrator, der auch gleichzeitig Benutzer des Systems ist sowie geographisch auf einen Rechner, auf dem alle Nodes [5] über das Loopback-Interface der Netzwerkkarte miteinander kommunizieren. Somit entfällt die Concurrency-Transparency, da nur ein Nutzer am System beteiligt ist sowie die Location-Transparency, Relocation- und Migration-Transparency, da alle Nodes über das Loopback-Interface kommunizieren, welches auf eine Netzwerkkarte beschränkt ist. Die Replication-Transparency ist in diesem Zusammenhang uninteressant, da jede Node

im System eindeutig identifizierbar ist,was Replikationen der Nodes für die Funktionsweise der verwendeten Algorithmen ausschließt.

Desweiteren ist ein Minimalset an Funktionen an die RPC-Architektur gefordert:

- void register(int id)
- void welding()
- void setStatus(int status)

Es soll kein zentrales System, bis auf Erfassung von Daten zu Experimentabläufen, am Gesamtsystem beteiligt sein. Alle beteiligten Nodes (Roboter) sollen sich auf einen Zyklus einigen, in dem immer drei Roboter nacheinander (Reihenfolge durch Lamport) schweißen. Insgesamt soll kein Roboter mehr als drei Schweißpuntke mehr als ein anderer gesetzt haben. Alle Roboter sollen am Ende eines Experiemntablaufs mindestens 20 Schweißpunkte gesetzt haben. In 99% soll ein Roboter nach einem Schweißvorgang weiterhin betriebsbereit sein. Im Umkehrschluss ist ein Roboter in 1% der Fälle nach einem Schweißvorgang nicht mehr betriebsbereit.
Zur Auswertung und Beobachtung des Ablaufs soll jeder Roboter seinen Ablauf loggen. Daraus ergeben sich folgende Anforderungen:

Requirement	Registrierung eines Nodes im System
Beschreibung	Ein Node kann sich im System mit allen anderen Nodes bekannt machen.
Eingaben	id: int; eindeutige ID für den Node
Ziel	Die Node hat sich mit allen anderen im System bekanntgemacht.
Vorbedingung	Die Node ist hochgefahren und betriebsbereit.
Nachbedingung	Die Node kennt alle anderen Nodes im System und alle anderen Nodes kennen die sich registrierende Node.

Tabelle 1.1: Requirement Registrierung

Requirement	Ein Roboter versucht zu schweißen
Beschreibung	Ein Roboter einer Node möchte Zugriff auf die Ressource haben und einen Schweißauftrag ausführen. Sobald die Ressource verfügbar ist, soll der Roboter seinen Schweißauftrag ausführen.
Eingaben	/
Ziel	Der Roboter hat einen Schweißvorgang durchgeführt.
Vorbedingung	Die Node sowie der Roboter der Node ist hochgefahren und betriebsbereit.
Nachbedingung	Der Roboter hat einen Schweißvorgang abgeschlossen und geht entweder in einen Fehlerzustand (in 1% der Fälle) oder bleibt betriebsbereit.

Tabelle 1.2: Requirement Schweißen (Welding)

Requirement	Bestimmung eines Zyklus
Beschreibung	Es muss sichergestellt sein, dass immer genau drei Roboter pro Zyklus einen Schweißauftrag ausführen.
Eingaben	/
Ziel	Ein Zyklus wird bestimmt und drei Roboter können schweißen.
Vorbedingung	Alle am Experiment teilnehmenden Nodes sowie dazugehörige Roboter sind hochgefahren und betriebsbereit.
Nachbedingung	Ein Zyklus wurde bestimmt und drei Nodes haben den Schweißauftrag erhalten.

Tabelle 1.3: Requirement Bestimmung eines Zyklus

Requirement	Logging der Prozessabläufe
Beschreibung	Jede Node muss seinen Zustand und die Abläufe loggen.
Eingaben	logText: String; Event oder Zustand
Ziel	Ein Event oder Zustand des Nodes wurde geloggt.
Vorbedingung	Die Node sowie der Roboter der Node ist hochgefahren und betriebsbereit.
Nachbedingung	Der Zustand der Node oder das Event wurden erfolgreich geloggt.

Tabelle 1.4: Requirement Logging

Requirement	Austauschbarkeit des Algorithmus
Beschreibung	Der Algorithmus für den gegenseitigen verteilten Ausschluss soll austauschbar als Bibliothek in die Architektur eingebunden werden.

Tabelle 1.5: Requirement Austauschbarkeit des Algorithmus

Requirement	Randbedingungen des Prozessablaufs
Beschreibung	• Nach dem Ablauf des Experiments sollen alle Roboter mindestens 20 Schweißpunkte gesetzt haben oder sich in einem Fehlerzustand befinden. • Kein Roboter darf mehr als drei Schweißpunkte als ein anderer gesetzt haben. • Die Anzahl der teilnehmenden Nodes (Roboter) liegt zwischen drei und 16.

Tabelle 1.6: Requirement Prozessbedingungen

1.1.2 UseCase

UseCase	Prozessablauf mit mindestens drei Robotern
ID	UC01
Beschreibung	Dieser UseCase schildert einen Prozessablauf von mindestens drei Robotern, wie er im Experiment durchgeführt wurde.
Vorbedingungen	Mindestens drei Roboter und der Logger werden hochgefahren.
Hauptszenario	1. Alle Roboter registrieren sich gegenseitig. 2. Ein Zyklus wird bestimmt. 3. Alle Roboter, welche im Zyklus ausgewählt wurden, schweißen nacheinander. 4. Ein neuer Zyklus wird bestimmt. 5. Wiederholen ab Punkt 2.
Alternative Szenarien	A1: 4. Mindestens ein Roboter ist in den Fehlerzustand gewechselt. 5. Versuch neuen Zyklus zu bestimmen und zu schweißen.
Fehlerszenarien	E1: 3. Schweißaufträge im Zyklus können nicht abgeschlossen werden. 4. System geht in Fehlerzustand.
Nachbedingung	Alle Roboter haben mindestens 20 Schweißpunkte gesetzt oder sind in einem Fehlerzustand.
Ergebnis	Das Experiment wurde erfolgreich beendet und alles wurde geloggt.

Tabelle 1.7: UseCase Prozessablauf mit mindestens drei Robotern

2 Design & Architektur

Zur Umsetzung der Anforderungen wurde eine RPC-Architektur entwickelt, die es ermöglicht den Prozess abzubilden. Im folgenden Kapitel wird diese Architektur vorgestellt sowie ihre Besonderheiten.

2.1 Systemkontext

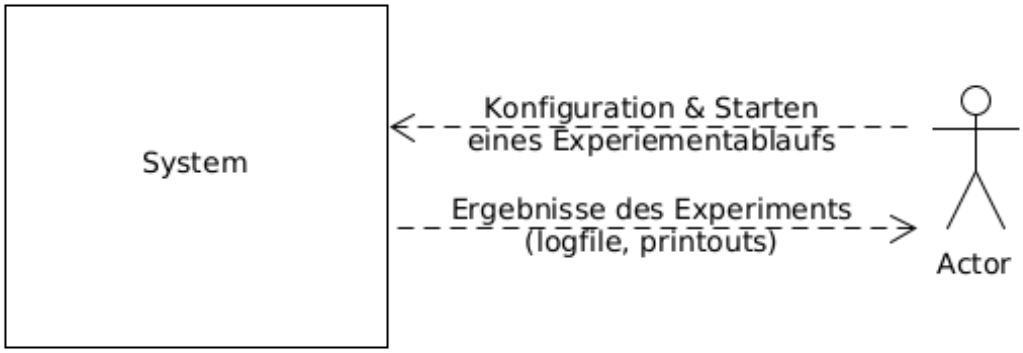


Abbildung 2.1: Systemkontext

Der Systemkontext des Gesamtsystems, wie in 2.1 zu sehen, umfasst den Nutzer, welcher gleichzeitig Administrator ist und das Gesamtsystem. Der Nutzer konfiguriert das System und startet im Anschluss die zuvor konfigurierte Konfiguration.

2.2 Lösungsansatz

Das System besteht aus einer RPC-Architektur und einer Robotersimulation. Die RPC-Architektur ermöglicht die Kommunikation und Steuerung der Abläufe der Roboter untereinander und dient damit als Basis der Architektur. Alle Nodes laufen auf einem Rechner, laufen also jeder auf einem Port. Um den Bereich abzustecken, in dem die Nodes laufen wird in der Konfiugration ein Portbereich mitgegeben. Dieser Bereich dient auch der Kommunikation der Roboter. Über den Portbereich lassen sich Nachrichten als Broadcast an jeden Port verschicken. Die Software wurde in der Programmiersprache „Kotlin“ implementiert (https://kotlinlang.org/), die auf Java basiert. Als Entwicklungsumgebung wurde IntellijIDEA Community 2020.3 (https://www.jetbrains.com/idea/) verwendet.

2.2.1 Registrierung eines Roboters

Die Roboter registrieren sich gegenseitig wenn das System hochgefahren wird. Sobald eine Node hochgefahren wird, wird ein Broadcast auf dem Portbereich getätigt, damit wird jeder Roboter mit jedem bekanntgemacht bevor die Wahl des Koordinators und Bestimmung des Zyklus losgeht. Somit hat man eine vollvermaschte Peer-To-Peer Topologie.

2.2.2 Bestimmung eines Zyklus

Da die Nodes im System unabhängig von einer zentralen Einheit (bis auf Erhebung von Experimentdaten) in einem Peer-To-Peer [5] Verbund existieren, muss zur Bestimmtung eines Zyklus eine Node die Rolle des Koordinators [5] übernehmen, welcher den Zyklus bestimmmt.
Zur Bestimmung des Koordinators wird zu Beginn des Experiments der Bully-Algorithmus [5] als Wahlalgorithmus verwendet, bei diesem wird über die ID der Roboter bestimmt wer als Koordinator gewählt wird.

2.2.3 Schweißvorgang (Welding)

Um den Schweißvorgang selbst zu simulieren wird beim welding() call in einer Node ein Thread gestartet, der eine konfigurierte Zeit wartet um die mechanische Bewegung und

das Schweißen der Roboter zu simulieren. Zusätzlich wird über die Konsole sowie in der Log-Datei eine Nachricht ausgegeben, welche auf den Schweißvorgang hinweist.

2.2.4 Logging der Prozessabläufe

Um das Logging der verschiedenen Abläufe und Zustände sicherzustellen wurde ein Logging-Server implementiert, der selbst über eine RPC Schnittstelle verfügt, auf dieser Log-Nachrichten der einzelnen Nodes übertragen werden können.

2.3 Architektur

2.3.1 Gesamtsystem

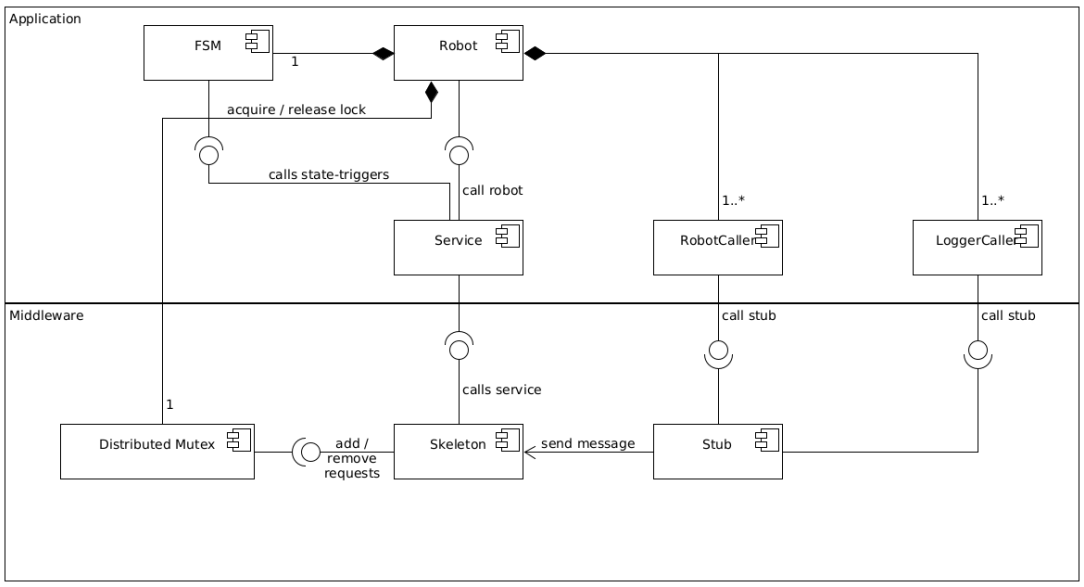


Abbildung 2.2: Komponentendiagramm Gesamtsystem

2.3.2 Robot

Eine zentrale Rolle in der Architektur spielt der Roboter, er beinhaltet Kommunikationselemente (Stubs [5] zu den anderen Robotern), eine Statemachine, eine Simulation des Schweißvorgangs (siehe 2.2) und eine Implementierung des verteilten gegenseitigen Ausschluss (Distributed Mutex 2.2).

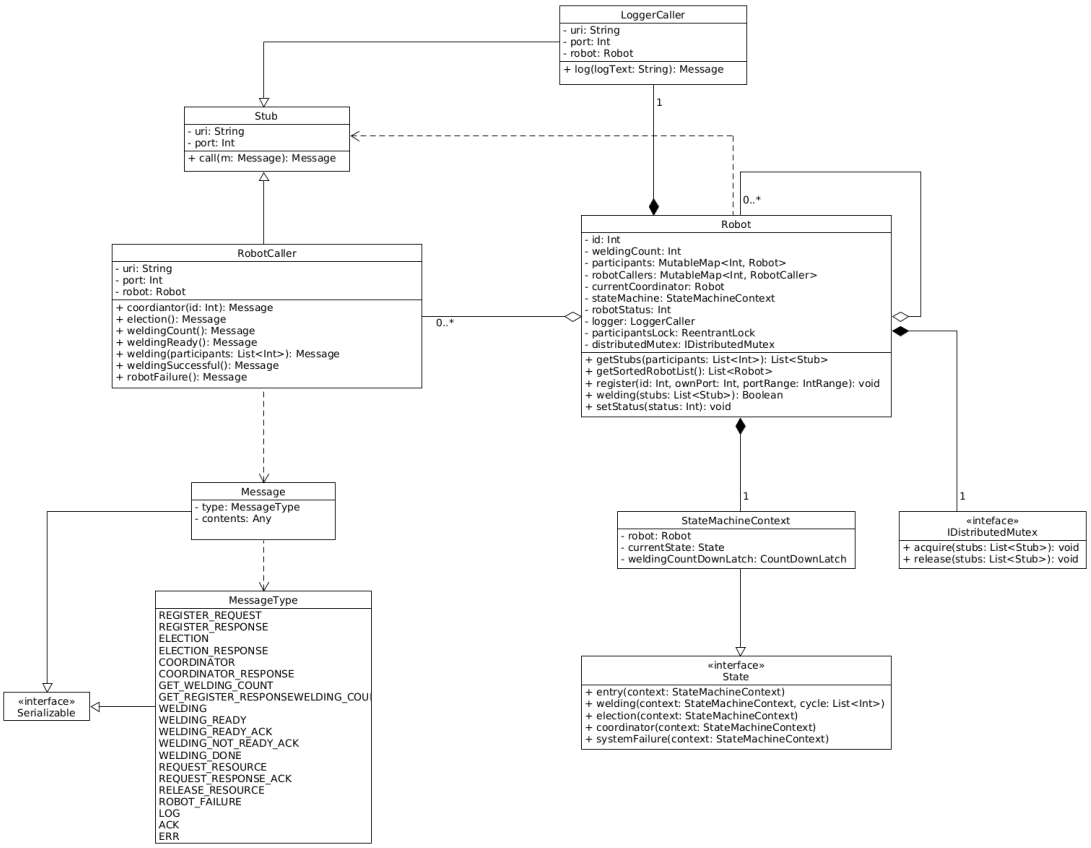


Abbildung 2.3: Klassendiagramm Robot

Im Klassendiagramm 2.3 zu sehen ist die Implementierung der Robot-Klasse. Wie vorher erwähnt sieht man hier die Bündelung der Funktionalitäten wie eine Instanz der FSM (StateMachineContext), die Robot- und LoggerCaller und den IDistributedMutex für den verteilten gegenseitigen Ausschluss.

2.3.3 Distributed Mutex

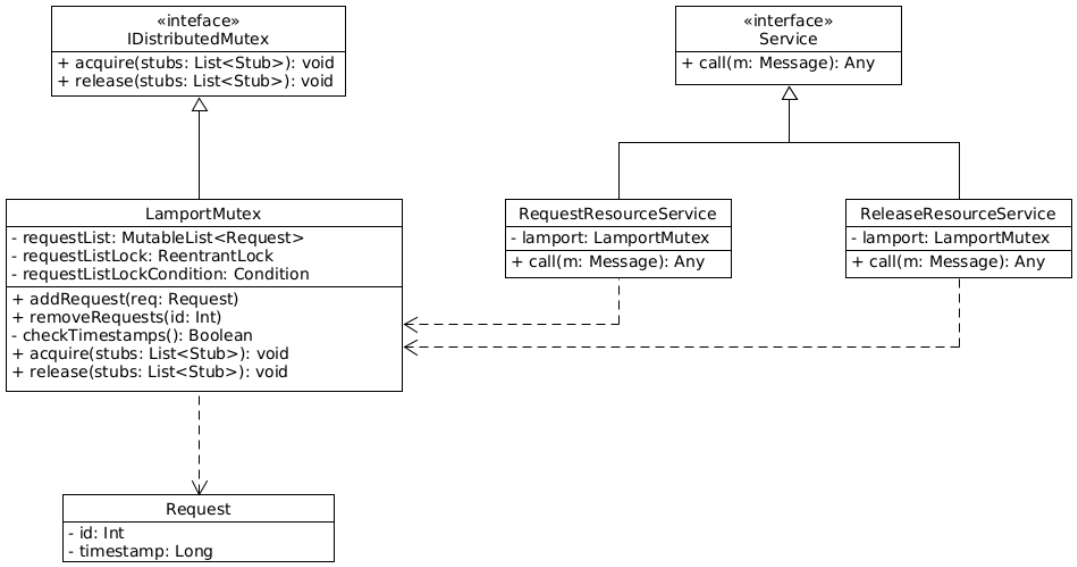


Abbildung 2.4: Klassendiagramm LamportMutex

In 2.4 zu sehen ist das Klassendiagramm des LamportMutex, der den Lamport-Algorithmus implementiert. Damit die Austauschbarkeit des verwendeten Algorithmus (siehe 1.5) sichergestellt ist, wird das IDistributedMutex als Interface definiert und wie in 2.3 zu sehen im Robot mit der Implementierung des LamportMutex instanziiert. Die beiden Services RequestResourceService und ReleaseResourceService sind Bestandteile der RPC-Architektur, über diese andere Roboter die Ressource akquirieren (RequestResourceService) oder freigeben (ReleaseResourceService) können.

2.3.4 FSM

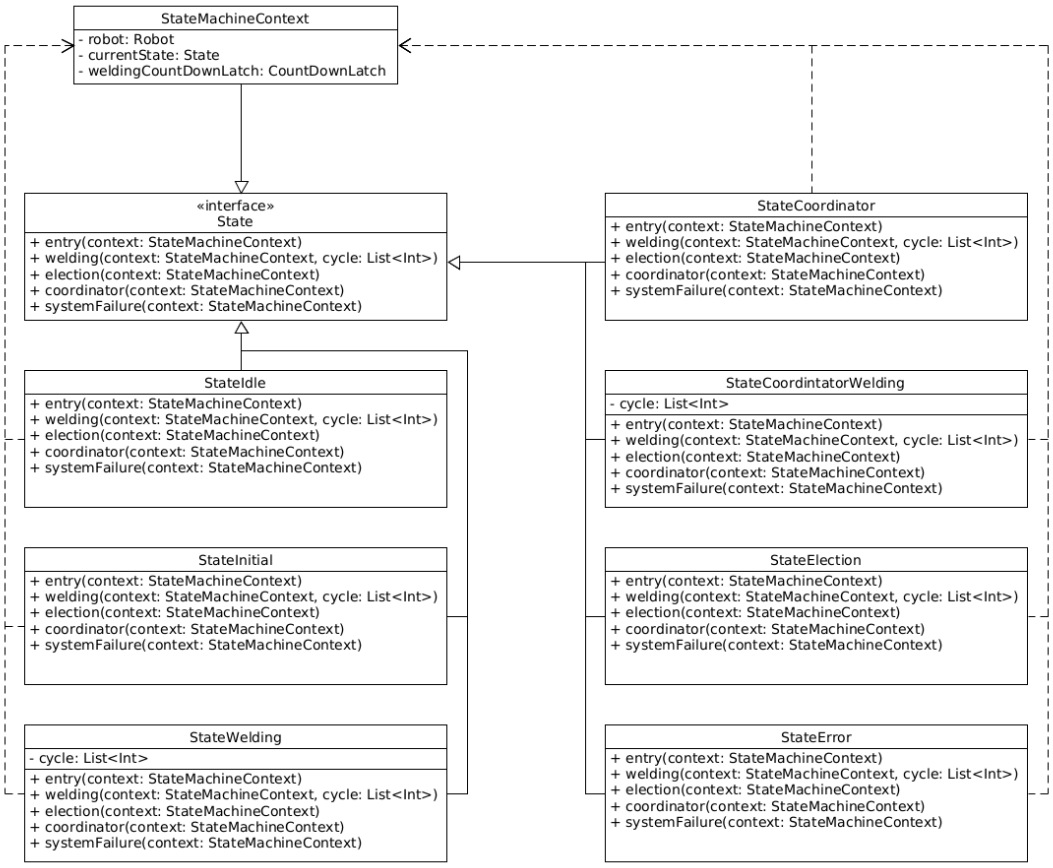


Abbildung 2.5: Klassendiagramm Statemachine

In Diagramm 2.5 ist das Klassendiagramm der FSM abgebildet. Für die Implementierung der FSM wurde das State Pattern [1] verwendet. Dieses ermöglicht eine übersichtliche und erweiterbare Implementierung einer FSM, da jeder Zustand als Klasse implementiert ist und vom Interface „State“ (siehe 2.5) erbt. Zum Hinzufügen von Zuständen muss eine neue Klasse implementiert werden, die von State erbt. Der StateMachineContext wird vom Roboter als Instanz gehalten und enthält zur Laufzeit immer den aktuellen Zustand der FSM.

2.3.5 Middleware

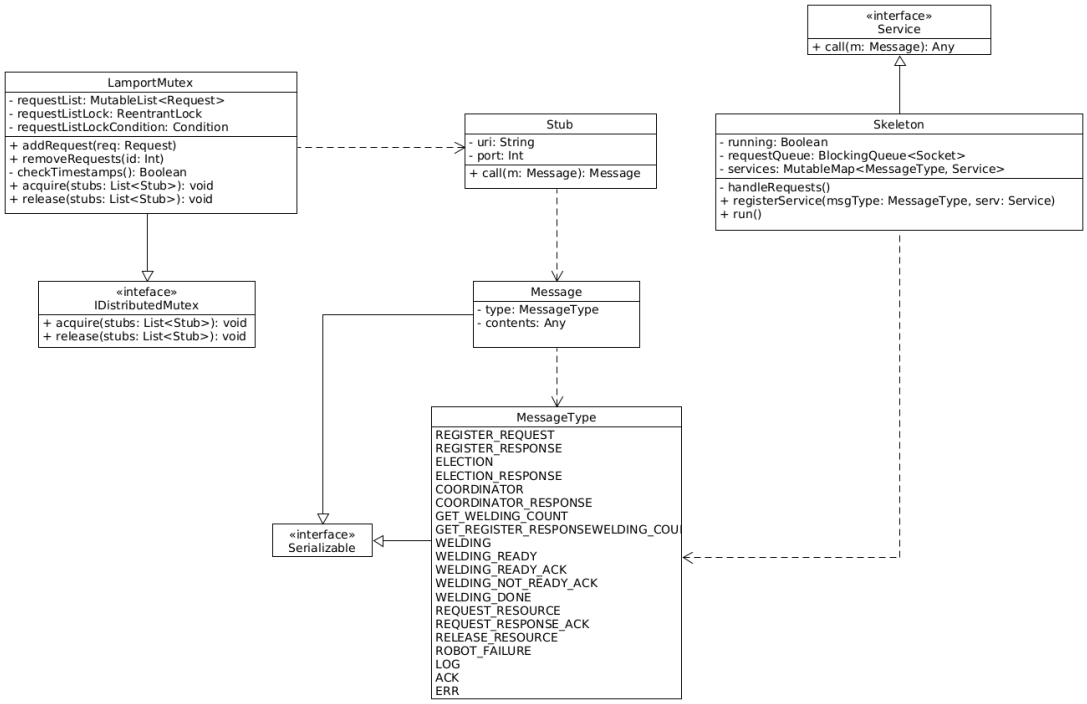


Abbildung 2.6: Klassendiagramm Middleware

Das Klassendiagramm 2.6 zeigt die Middleware mit RPC-Lösung. Stub und Skeleton arbeiten mit Streams (ObjectInputStream [3], ObjectOutputStream [4]) zur Datenübertragung. Diese ermöglichen eine flexible RPC-Lösung, da auf ein externes Datenformat (z.B. json) für das Marshalling [5] verzichtet werden kann. Über die Streams lassen sich ganze Java-Objekte transferieren, in diesem Fall „Message“ Objekte (siehe 2.6). Eine Message beinhaltet den MessageType und den eigentlichen Inhalt (Content) der Nachricht, worüber sich alle Dienste der RPC-Lösung abbilden lassen.

2.4 Prozesse und Abläufe

2.4.1 StateMachine

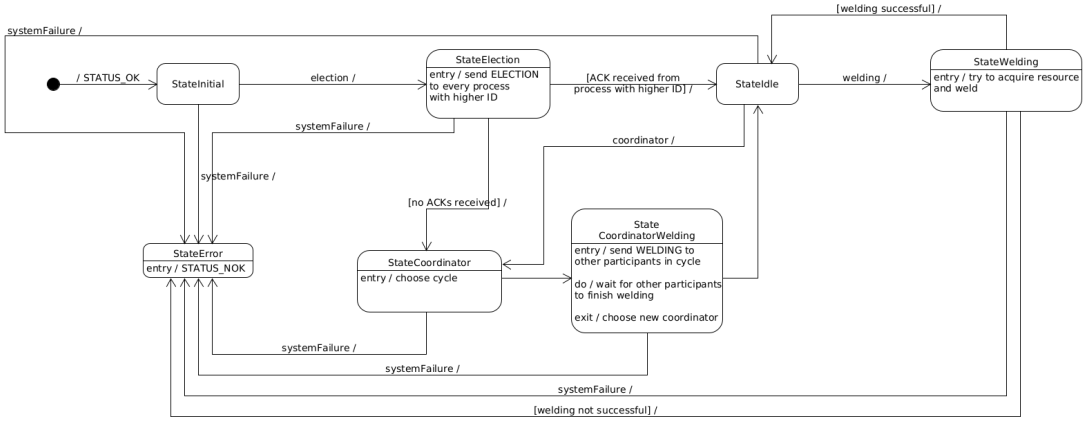


Abbildung 2.7: Zustandsdiagramm FSM

Die Statemachine, abgebildet in 2.7, zeigt alle Zustände des Nodes und Roboters. In den beiden States „StateCoordinator“ und „StateCoordinatorWelding“ ist die Logik des Koordinators implementiert. Dadruch lassen sich Zyklen bestimmen und diese auch nacheinander ausführen. Durch die Statemachine ist damit Anforderung 1.3 erfüllt.

2.4.2 Registrierung

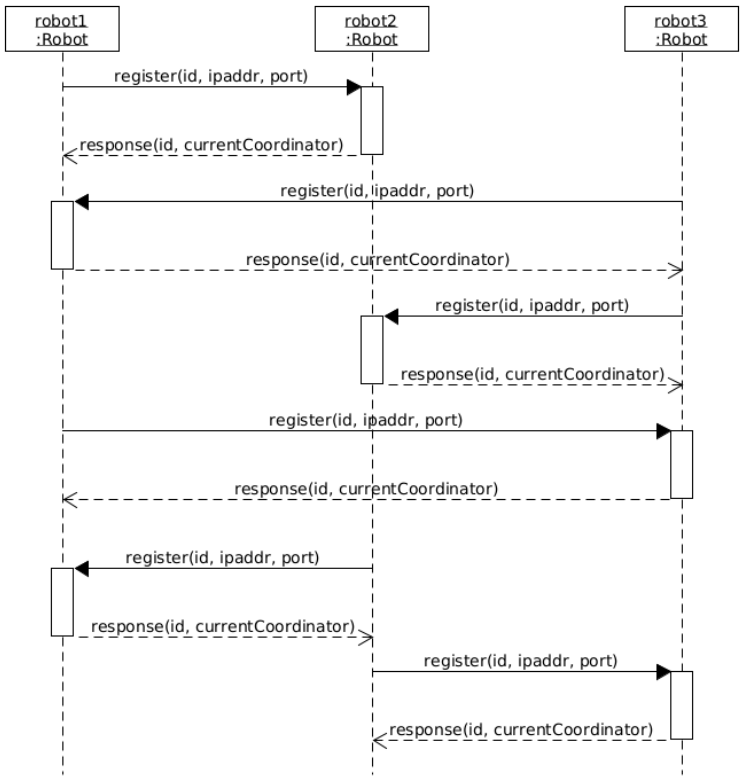


Abbildung 2.8: Sequenzdiagramm Registrierung

Im Sequenzdiagramm 2.8 ist der Registrierungsprozess von drei Robotern abgebildet. Jeder Roboter schickt allen anderen in der definierten Portrange (siehe in Kapitel 2.2) einen Registrierungsrequest mit seiner eigenen ID, seiner IP-Adresse und dem Port, auf dem er läuft. Als Antwort erhält er die ID des Gegenübers und kann diese bei sich in der Teilnehmerliste eintragen. Außerdem erhält er in der Antwort, falls bereits bestimmt, den aktuellen Koordinator im System. Wenn sich ein Roboter später im System registriert, während zum Beispiel bereits ein Zyklus bearbeitet wird, trägt dieser den aktuellen Koordinator bei sich ein und wartet bis er in einem Zyklus ausgewählt wird oder selber Koordinator wird.

2.4.3 Election

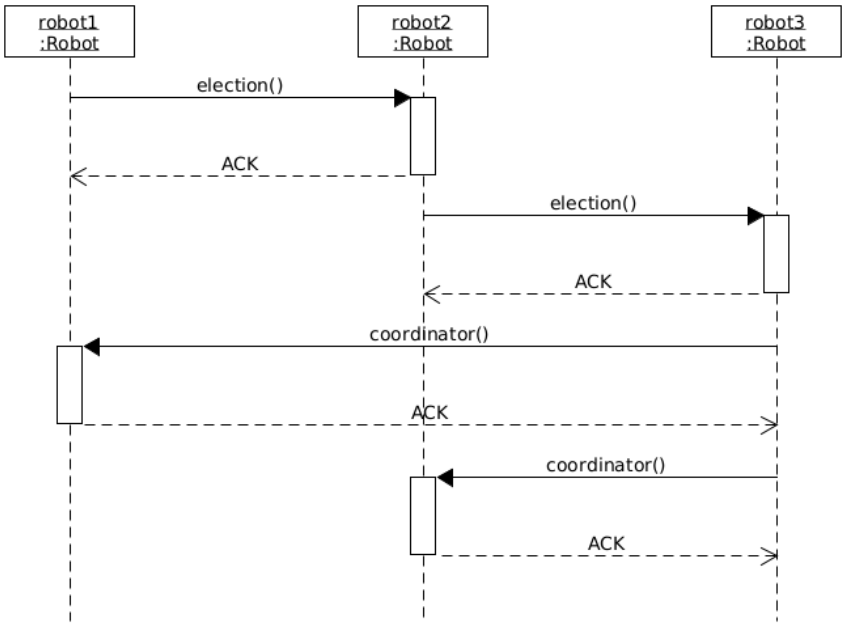


Abbildung 2.9: Sequenzdiagramm Election

In Abbildung 2.9 ist die Wahl des Koordinators mit dem Bully-Algorithmus abgebildet. Wenn ein Experimentablauf gestartet wird und sich genug Roboter registriert haben, wird eine Wahl mit dem Bully-Algorithmus angestoßen. Dadurch wird der erste Koordinator im System bestimmt, welcher einen Zyklus auswählt.

2.4.4 Schweißzyklus

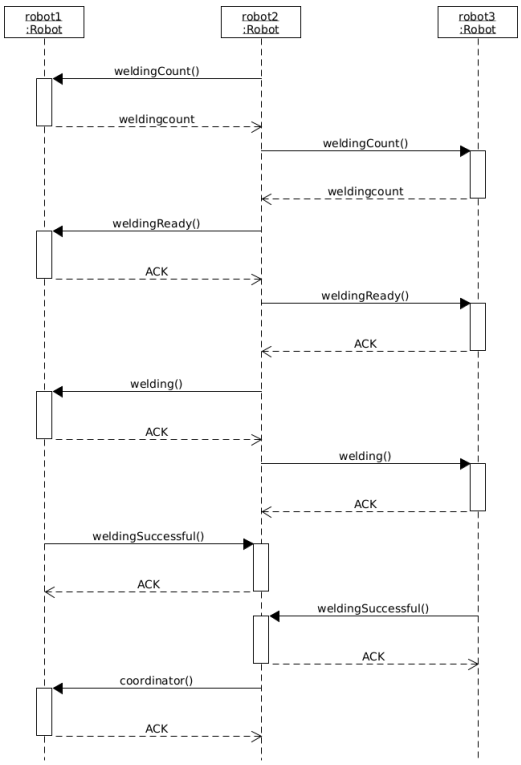


Abbildung 2.10: Sequenzdiagramm Schweißzyklus

Das Sequenzdiagramm 2.10 zeigt einen Schweißzyklus mit drei Robotern. Roboter Nr. 2 (robot2) ist der aktuell ausgewählte Koordinator. Zunächst holt der Koordinator sich die Anzahl abgeschlossenen Schweißaufträge aller anderen Roboter im System um zu bestimmen, welche Roboter im Zyklus schweißen sollen. Die zwei Roboter mit der niedrigsten Zahl an abgeschlossenen Schweißaufträgen werden als Teilnehmer im Zyklus gewählt. Dadurch wird sichergestellt, dass kein Roboter mehr als drei Mal so oft geschweißt hat wie ein anderer im System (siehe Anforderung 1.6). Wenn dies geschehen ist, wird gefragt ob alle Teilnehmer des Zyklus bereit zum schweißen sind. Wenn alle Teilnehmer ihre Bereitschaft bestätigt haben, gibt der Koordinator den anderen Robotern und sich selbst die Anweisung zu schweißen. Die Reihenfolge der Schweißvorgänge wird durch den Lamport-Algorithmus festgelegt. Während des Schweißvorgangs wird parallel die Zykluszeit vom Koordinator überwacht, sollte diese überschritten werden, wird das System in einen Fehlerzustand versetzt (siehe UseCase 1.7).

2.4.5 Roboterausfall

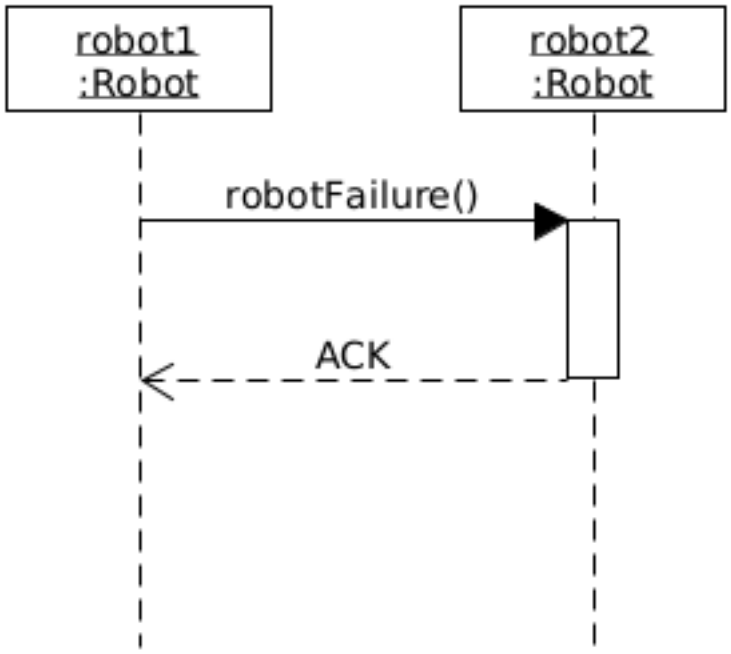


Abbildung 2.11: Sequenzdiagramm Roboterausfall

Wenn ein Roboter nach einem Schweißauftrag in einen Fehlerzustand wechselt, sendet er an alle anderen Teilnehmer eine Nachricht, dass der Roboter nicht mehr betriebsbereit ist (siehe 2.11). Die Roboter, die diese Nachricht erhalten haben löschen den Roboter aus ihrer Teilnehmerliste und wird für den weiteren Experimentverlauf nicht mehr berücksichtigt.

3 Evaluation

Um die Leistung des Algorithmus und der RPC-Lösung zu bewerten wurden mehrere Experimentdurchläufe mit verschiedenen Roboterzahlen und Zykluszeiten durchgeführt.

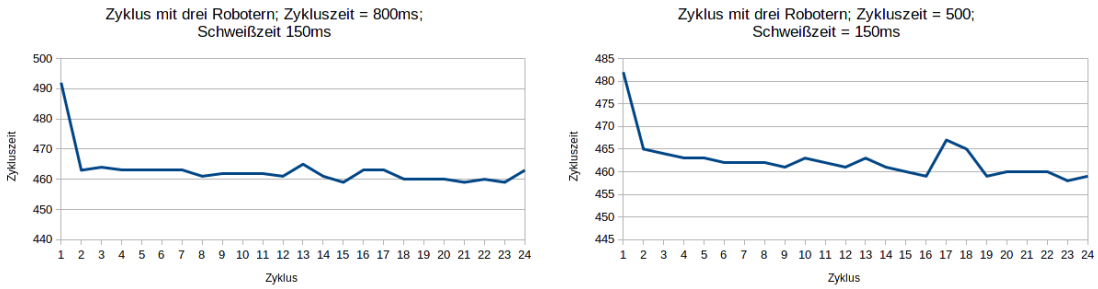
Daten zum Testsystem:

- Prozessor: AMD Ryzen 7 2700X Eight-Core Processor mit 8 Kernen, 16 Threads und 3.7GHz pro Kern
- RAM: 16 GB Corsair DDR4 mit 2666 Mhz Taktrate

Jeder Experimentdurchlauf wurde pro Roboteranzahl mit zwei Zykluszeiten getestet, 500 Millisekunden und 800 Millisekunden. Die Schweißzeit pro Roboter wurde in der Simulation bei 150 ms angesetzt.

3.1 Experimentdurchläufe

3.1.1 Experimentdurchlauf mit drei Robotern



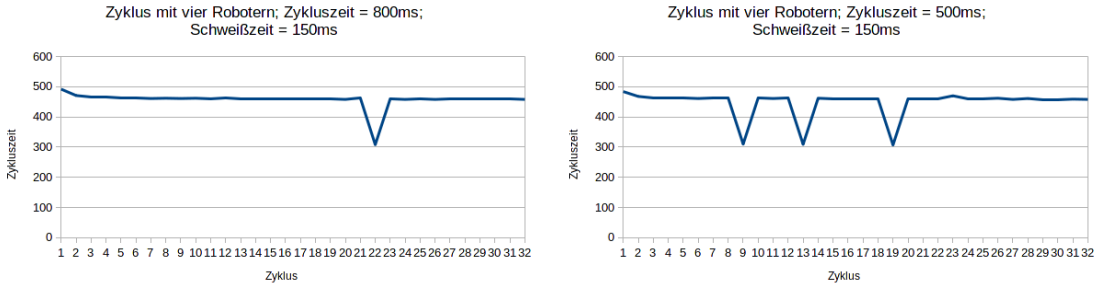
(a) Durchlauf mit drei Robotern und 800ms Zykluszeit

(b) Durchlauf mit drei Robotern und 500ms Zykluszeit

Abbildung 3.1: Experimentdurchlauf mit drei Robotern

Im ersten Durchlauf 3.1 mit jeweils 800ms 3.1a und 500ms 3.1b Zykluszeit ist zu erkennen, dass der erste Zyklus länger braucht als die darauf folgenden. Dies liegt daran, dass am Anfang durch die Registrierung der Roboter und der Wahl des ersten Koordinators mehr Nachrichten verarbeitet werden müssen als später im Durchlauf. Die Zykluszeit pendelt sich nach dem ersten Zyklus bei einem Durchschnittswert von 461ms ein. Wenn man die Schweißzeit pro Roboter rausrechnet (150ms pro Roboter, also 450ms pro Zyklus) kommt man auf 11ms reine Bearbeitungszeit des Algorithmus.

3.1.2 Experimentdurchlauf mit vier Robotern



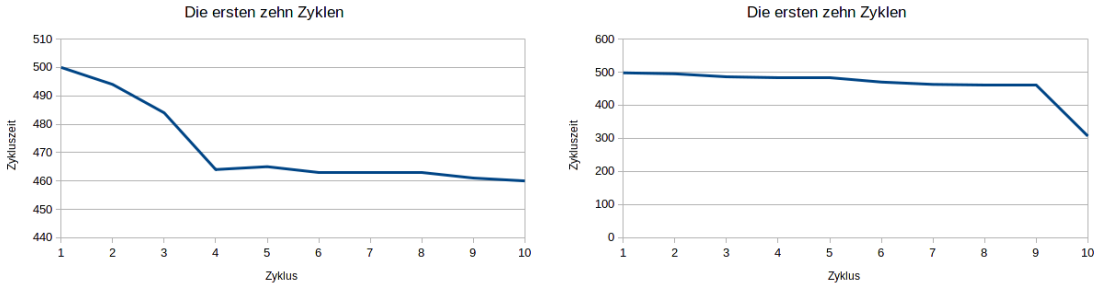
(a) Durchlauf mit vier Robotern und 800ms Zykluszeit

(b) Durchlauf mit vier Robotern und 500ms Zykluszeit

Abbildung 3.2: Experimentdurchlauf mit vier Robotern

Ein Experimentdurchlauf mit vier Robotern 3.2 unterscheidet sich nicht von einem mit drei Robotern hinsichtlich Zykluszeiten. Eine Auffälligkeit existiert jedoch in Form von auffällig kurzen Zykluszeiten (siehe Zyklus 22 in 3.2a oder Zyklus 9, 13 und 20 in 3.2b. Dies ist darauf zurückzuführen, dass alle Roboter gleichzeitig versuchen per Lamport Zugriff auf die Ressource zu erhalten. Wenn nun ein ein Prozess bei allen anderen den Zugriff erbittet und niemand in der Zeit selbst einen Zugriff auf die Ressource erbittet, greift er auf die Ressource zu, obwohl ein anderer Prozess mit höherer ID und gleichem Timestamp eher Zugriff auf die Ressource erhalten würde. Lamport erwartet allerdings eine klare Reihenfolge der Events die in dem System auftreten [2] (Erbitten der Ressource, Freigabe der Ressource).

3.1.3 Auswirkung der Skalierung auf Zykluszeit



(a) Die ersten 10 Zyklen eines Durchlaufs mit 8 Robotern und 500ms Zykluszeit

(b) Die ersten 10 Zyklen eines Durchlaufs mit 16 Robotern und 500ms Zykluszeit

Abbildung 3.3: Experimentdurchlauf mit 8 und 16 Robotern

In 3.3 zu erkennen hat die unterschiedliche Problemgröße der Durchläufe einen Einfluss auf die Zykluszeiten am Anfang des Experiments. Während bei 8 Robotern die Zykluszeit bereits ab dem 4. Zyklus (3.3a) sich auf dem Durchschnittswert von 461ms einpendelt braucht es bei 16 Robotern 10 Zyklen (3.3b). Die mit jedem zusätzlich teilnehmenden Roboter steigende Nachrichtenkomplexität sorgt für diese anfangs höheren Zykluszeiten. Es melden sich mehr Roboter im System an demnach steigt auch die Zahl der Registrierungsnachrichten.

4 Fazit

Die Umsetzung des Lamort Algorithmus für gegenseitigen Ausschluss in verteilten Systemen war erfolgreich. Mit der in dieser Arbeit beschriebenen Lösung ließ sich der Algorithmus in eine selbst entwickelte RPC-Lösung integrieren und am beschriebenen Use-Case demonstrieren. Die in Kapitel 1.1.1 entwickelten Anforderungen konnten umgesetzt werden, sowie die Design-Goals eines verteilten Systems [5] insofern sie nicht durch den Rahmen der Aufgabenstellung bereits eingeschränkt wurden (siehe 1.1.1). Eine Ausnahme stellt das in Kapitel 3.1.2 erläuterte Fehlverhalten der hier erarbeiteten Lösung dar.

Literaturverzeichnis

[1] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John M.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. – ISBN 0-201-63361-2

[2] Lamport, Leslie: Time, Clocks, and the Ordering of Events in a Distributed System. In: *Communications of the ACM* (1978)

[3] Oracle: *Class ObjectInputStream*. 2020. – URL https://docs.oracle.com/javase/7/docs/api/java/io/ObjectInputStream.html. – Zugriffsdatum: 30.03.2021

[4] Oracle: *Class ObjectOutputStream*. 2020. – URL https://docs.oracle.com/javase/7/docs/api/java/io/ObjectOutputStream.html. – Zugriffsdatum: 30.03.2021

[5] Tanenbaum, Andrew S. ; Steen, Marten van: *Distributed Systems 3rd edition*. Maarten van Steen, 2018. – URL https://www.distributed-systems.net/index.php/books/ds3/. – ISBN 978-90-815406-2-9

A Anhang

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original