Verteilte Systeme

VS Praktikum SoSe 2025

Manh-An David Dao, Philipp Patt, Jannik Schön, Marc Siekmann

2. Juli 2025

Inhaltsverzeichnis

1 Einführung und Ziele

1.1 Aufgabenstellung

Es soll ein verteiltes Steuerungssystem gemäß den Prinzipien verteilter Systeme nach Tanenbaum & van Steen entworfen und implementiert werden. Über ein ITS-Board (STM32F4) sollen beliebig viele autonome Roboterarme innerhalb eines kontrollierten Areals (z.B. BT7 R7.65 – als realer Testbereich) sicher und effizient gesteuert werden. Die verteilte Architektur soll dabei explizit gewährleisten, dass durch Fehlverhalten der Software oder Architektur keine Gefahr für Anwesende entstehen kann. Daraus geht hervor, dass der Benutzer einen zu steuernden Roboterarm auswählt und diesen anschließend mit den folgenden Bewegungen steuern kann:

- Roboterarm hoch
- Roboterarm runter
- Roboterarm links
- Roboterarm rechts
- Greifer auf
- Greifer zu

1.2 Qualitätsziele

Tabelle 1.1: Qualitätsziele der Software Engineering

Ziel	Beschreibung	Metrik
Funktionalität	Der ausgewählte Roboterarm muss Steuerbefehle korrekt umsetzen	Alle Abnahmetests werden erfolgreich bestanden
Zuverlässigkeit	Fehler dürfen den Betrieb nicht gefährden. Fehlererkennung und -toleranz müssen integriert sein.	Das System ist über dem gesamten Abnahmezeitraum stabil (ca. 1,5 h)
Skalierbarkeit	Zusätzliche Roboter oder Komponenten sollen ohne Änderungen an der bestehenden Architektur integrierbar sein.	Es können beliebig viele Roboter hinzugefügt und entfernt werden (0 - 253)
Leistung	Reaktionszeiten auf Steuerbefehle und Ereignisse müssen innerhalb definierter Zeitgrenzen liegen.	max. 200 ms bis Befehlsausführung
Sicherheit (Safety)	Es bewegt sich immer genau ein Roboterarm. Sollte das System nicht wie gewünscht reagieren, wird ein sicherer Zustand erreicht	Reaktionszeit max. 250 ms, bis Roboterarm stoppt
Wartbarkeit	Der Code muss übersichtlich sein, gut dokumentiert sein und wenig Komplexität enthalten.	Zyklomatische Komplexität ≤ 10 und LOC ≤ 30 pro Methode/Funktion exklusive Kommentar
Benutzerfreundlichkeit	Bedienung ist intuitiv, sodass die Nutzer möglichst wenig Zeit mit der Einarbeitung in die Bedienung benötigen.	Keine Einweisung erforderlich
Anpassbarkeit	Neue Funktionen, Sensoren oder Roboterarme sollen ohne tiefgreifende Änderungen am System integriert werden können.	Erweiterungen können durch modulare Struktur und erweiterbare Schnittstellen einfach hinzugefügt werden.

Ziel	Beschreibung	Metrik
Kompatibilität	Das System soll mit verschiedenen Hard- und Softwareplattformen kompatibel sein.	Es unterstützt die Kommunikation mit Embedded-Systemen (z.B. Raspberry Pi).
Ressourcenteilung	Mehrere Roboterarme können über ein Netzwerk von einem ITS-Board gesteuert werden.	Alle hinzugefügten Roboterarme sind ansteuerbar und teilen die verfügbaren Rechenressourcen vom ITS-BRD effizient.
Offenheit	Zugänglichkeit Interoperabilität Portabilität	Einzelne Softwarekomponenten können, ausgetauscht oder portiert werden, um zb Standards bzgl. Kommunikation und Datenstruktur zu tauschen oder um die Plattform zu wechseln. Die Funktionalität bleibt gleich
Zugriffstransparenz	Die Umsetzung der Roboterarmsteuerung ist für den Benutzer nicht erkennbar und möglichst einfach gehalten	GUI zeigt nur Zustände des ausgewähltem Roboterarms an (z.B. Verfügbarkeit) und der Roboterarm ist über das ITS-Board so einfach wie möglich steuerbar
Lokalitäts-Transparenz	Die Netzwerk- und Softwarestruktur ist für den Benutzer unsichtbar	Aus der Sicht der Benutzer sind alle benötigten Ressourcen zu Steuerung direkt und lokal auf dem ITS-Board

1.3 Stakeholder

Tabelle 1.2: Interessen der Stakeholder

Stakeholder	**Interesse**
Nutzer	• vollständige Funktionalität • Benutzerfreundlichkeit • Zuverlässigkeit: Das System kann über den gesamten benötigten Zeitraum ohne Ausfälle genutzt werden • Reaktionszeit: Die Robotersteuerung reagiert innerhalb eines definierten Zeitfensters • Sicherheit: Während des Betriebs kommen keine Personen durch Fehler des Systems zu schaden
Betreiber	• Portabilität: Das System kann auf verschiedenen Plattformen betrieben werden. • Zuverlässigkeit: Das System kann über den gesamten benötigten Zeitraum ohne Ausfälle genutzt werden • Sicherheit: Während des Betriebs kommen keine Personen durch Fehler des Systems zu schaden
Entwicklerteam	• Professor bzw. der „Kunde“ ist Mittwoch Nachmittag verfügbar. Bis dahin sind alle offenen Fragen zusammenzustellen. • Wartbarkeit • Portabilität: Das System kann auf verschiedenen Plattformen betrieben werden (z.B Testen) • Austauschbarkeit: Softwaremodule können ohne großen Aufwand ersetzt werden
Professor	• Zugang zu allen Arbeitsmitteln zwecks Bewertung und Kontrolle • Das Endprodukt besitzt alle geforderten Funktionalitäten

2 Randbedingungen

2.1 Technische Randbedingungen

Das verteilte Steuerungssystem unterliegt mehreren festgelegten technischen Rahmenbedingungen, die den Entwicklungs- und Implementierungsspielraum einschränken. Diese Bedingungen sind im Folgenden aufgeführt:

- **Hardwareplattform:** Das System basiert auf ein Raspberry Pi der den jeweiligen Roboterarm steuert. Eine Software auf dem Raspberry Pi stellt eine API zur Verfügung, sodass die Roboter sich ohne Einschränkungen bewegen können. Ein ITS-BRD dient als Steuerung/Administration der Roboterarme.
- **Programmiersprachen:**
  - **C:** Das ITS-Board wird mit C Programmiert.
  - **Java:** Die Roboterarme werden mit einem Javaprogramm angesteuert.
- **Netzwerkumgebung:** Es steht ein Adressenraum von 255 Adressen zur verfügung. Daraus folgt, dass nicht mehr als 254 Roboterarme eingesetzt werden können. Das ITS-Board ist ebenfalls ein Teilnehmer des Netzes.

2.2 Organisatorische Randbedingungen

- **Umgebung:** Der Abnahme bereich befindet sich im Raum BT7 R7.65. Die Steuerung und Navigation des Roboters müssen innerhalb der räumlich definierten Grenzen erfolgen. Die dort vorhandene LAN-Infrastruktur kann zur Netzwerkkommunikation verwendet werden.
- **Zeit:** Entwicklungszeitraum beträgt 12 Wochen.
- **Vorwissen:** Einige Konzepte und Herangehensweisen werden erst im Laufe der 12 Wochen gelernt.
- **Budget:** Es steht kein Budget zur Verfügung.

3 Kontextabgrenzung

Ziel dieses Kapitels ist es, das zu entwickelnde System innerhalb seines fachlichen und technischen Umfelds klar einzugrenzen. Dazu wird das System in Bezug auf seine Aufgaben (fachlicher Kontext), seine Einbettung in die bestehende technische Infrastruktur (technischer Kontext) sowie die definierten externen Schnittstellen beschrieben.

3.1 Fachlicher Kontext

Das verteilte System ermöglicht es, beliebig viele Roboterarme (zwischen 1 und 254) in einem Raum zu steuern. Die Kommunikation mit dem Nutzer sowie die Steuerung der Roboterarme erfolgen über ein ITS-Board. Zur Unterstützung des Nutzers wird auf einem Rechner eine Benutzeroberfläche (UI) bereitgestellt, die eine intuitive Orientierung und Bedienung der Roboterarme ermöglicht.

3.2 Technischer Kontext

Die Realisierung des Systems erfolgt innerhalb eines /24-Netzwerks. In diesem Netzwerk befinden sich das ITS-Board, sowie mehrere Roboterarme. Jeder Roboterarm wird von einem vorgeschalteten Raspberry Pi angesteuert, der einen eigenen UDP-Server zur Kommunikation bereitstellt. Die Benutzeroberfläche (UI) ist über das gleiche Netzwerk erreichbar und kommuniziert per WebSocket mit dem ITS-Board. Optional kann das System um externe Dienste wie die ICC-Cloud erweitert werden kann.

3.3 Externe Schnittstellen

- System: Der Benutzer kann über ein Bildschirm einzelne erreichbare Roboterarme erkennen. Die Auswahl und Steuern der einzelnen Roboterarme wird durch IO-Buttons realisiert.
- ITS-Board: Das ITS-Board kommuniziert über einen leichtgewichtigen UDP-Server mit einer eigenen IP-Adresse mit dem System.
- Roboterarm: Der Roboterarm wird über eine IP-Adresse angesprochen. Die Kommunikation erfolgt über einen vorgeschalteten Raspberry Pi, der einen eigenen UDP-Server und eine API zur Steuerung bereitstellt.
- ICC Cloud: Die ICC Cloud ist über eine IP-Adresse im Netzwerk erreichbar.

4 Lösungsstrategie

4.1 Controller

Funktion	**Voraussetzung**	**Semantik**
void update(AvailableRobots: String[], SelectedRobotIdx: int, Error: bool, Confirmation: bool)	Modell hat Änderungen	Benachrichtigt den Controller, der die View aktualisiert.

Tabelle 4.1: Funktionen des Controllers

4.2 Model

4.2.1 StateService

Funktion	**Voraussetzung**	**Semantik**
void select(SelectDirection selectDirection)	Liste der verfügbaren Roboter ist nicht leer	Wählt einen Roboter aus der Liste der verfügbaren Roboter aus und aktualisiert den Status.
void select(int sd)	sd ist ein gültiger Index für die SelectDirection-Enum	Ruft die Auswahl eines Roboters basierend auf dem Richtungsindex auf.
void setError(boolean error, boolean confirm)	Änderung des Fehler- oder Bestätigungsstatus notwendig	Setzt den Fehler- und Bestätigungsstatus und benachrichtigt den Controller.
void register(String motorName)	Motorname im richtigen Format (z.B. R1A1)	Registriert den Motor und fügt den zugehörigen Roboter der Verfügbarkeitsliste hinzu, falls dieser vollständig ist.
void subscribe()	keine	Placeholder-Methode, aktuell ohne Implementierung.
String getSelected()	Ein Roboter ist ausgewählt (selectedRobot > 0)	Gibt den Namen des aktuell ausgewählten Roboters zurück.
private void sendUpdate()	Änderungen im Status des StateService	Benachrichtigt den Controller über die aktuellen Verfügbarkeiten und Zustände.
boolean isAvailable()	keine	Prüft, ob alle vier Aktoren (A1, A2, A3, A4) des Roboters aktiviert sind. Liefert `true`, wenn der Roboter vollständig verfügbar ist.

Tabelle 4.2: Funktionen der Komponente StateService inklusive Robot-Klasse

4.2.2 MoveAdapter

Funktion	**Voraussetzung**	**Semantik**
void move(int md)	md ist ein gültiger Index der RobotDirection-Enum	Ruft die Bewegungsfunktion basierend auf dem Richtungsindex auf.
void move(RobotDirection rd)	Ein Roboter muss im StateService ausgewählt sein	Führt die Bewegung des Roboters in die angegebene Richtung aus. Bei Erfolg wird der Fehlerstatus im StateService zurückgesetzt. Bei Fehler wird der Fehlerstatus im StateService gesetzt.

Tabelle 4.3: Funktionen des MoveAdapter

4.2.3 ActuatorController

Funktion	**Voraussetzung**	**Semantik**
void move(int md)	md ist ein gültiger Index der Direction-Enum	Wandelt den Index in eine Richtung um und ruft die entsprechende Bewegungsfunktion auf.
void move(Direction direction)	Der Aktuator ist initialisiert	Bewegt den Aktuator in die angegebene Richtung (INCREASE oder DECREASE) und setzt den neuen Wert.
private void applyValue()	Ein gültiger Aktuatorname muss gesetzt sein	Setzt den aktuellen Wert des Aktuators physisch am Roboterarm und gibt den Zustand in der Konsole aus.
int getValue()	keine	Gibt den aktuellen Positionswert des Aktuators zurück.

Tabelle 4.4: Funktionen der Komponente ActuatorController

4.3 View

Die View besteht aus den unabhängigen Blöcken IO und UI. Die UI bietet eine Softwareschnittstelle an, die IO keine.

4.3.1 IO Funktionen

Funktion	**Voraussetzung**	**Semantik**
void readInputs()	Eingabe durch den Benutzer erfolgt	Überprüft die Benutzereingaben und löst entsprechende Aktionen aus.
int initIO()	IO-Hardware verfügbar	Initialisiert und überprüft die IO-Hardwareschnittstellen und gibt einen Fehlercode bei Problemen zurück.

Tabelle 4.5: IO Funktionen

4.3.2 UI Funktionen

Funktion	**Voraussetzung**	**Semantik**
void updateView(AvailableRobots: String[],SelectedRobotIdx: int, Error: bool, Confirmation: bool)	Gültige Modell-Daten vorhanden	View-Schnittstelle. Aktualisiert die UI mit den neuesten Roboter-Daten und Statusinformationen (Fehler, Bestätigung).

Tabelle 4.6: UI Funktionen

5 Bausteinsicht

Um ein besseres Verständnis über die Struktur des Systems zu bekommen, nutzen wir die Bausteinsicht. Sie hilft dabei ein gemeinsames Verständnis des Systems innerhalb des Teams zu bekommen. Die zur Zerlegung benutzte Dekompistionsstrategie ist funktional.

5.1 Bausteinsicht Level 1

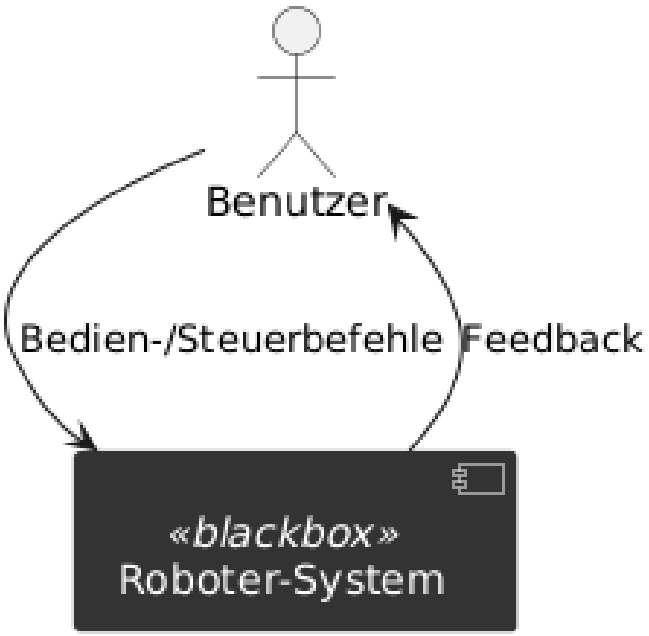

Abbildung 5.1: Bausteinsicht Level 1

Komponente	**Beschreibung**
Roboter-System	Gesamtsystem, das alle internen Steuer-, Safety- und Kommunikationsfunktionen kapselt. Empfängt Bedien-/Steuerbefehle vom Benutzer, verarbeitet sie und liefert Feedback.

Tabelle 5.1: Bausteinsicht Level 1

5.2 Bausteinsicht Level 2

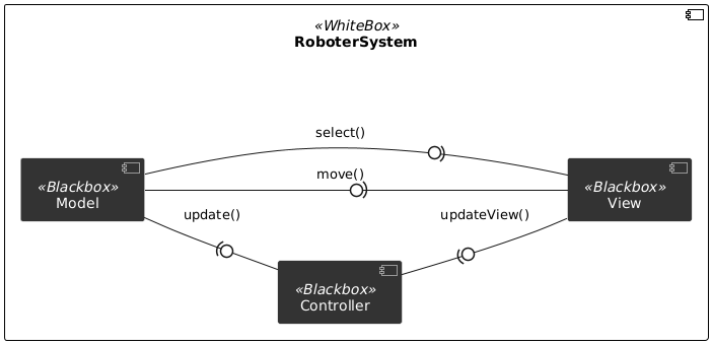

Abbildung 5.2: Bausteinsicht Level 2

Komponente	**Beschreibung**
Model	Beinhaltet die Geschäftslogik. Sendet Zustandsänderungen an den Controller.
Controller	Der Controller fungiert als Observer vom Model und gibt Zustandsänderungen des Models an die View weiter.
View	Bietet Interaktionsmöglichkeiten für den Nutzer und stellt dem Benutzer Informationen dar.

Tabelle 5.2: Bausteinsicht Level 2

5.3 Bausteinsicht Level 3 Model

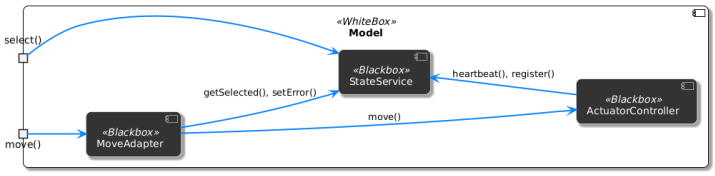

Abbildung 5.3: Bausteinsicht Level 3 Model

Komponente	**Beschreibung**
StateService	Speichert den ausgewählten Roboter und die verfügbaren Roboter. Bei Zustandsänderungen informiert er den Controller.
MoveAdapter	Empfängt einen Steuerungsbefehl vom View, übersetzt diesen mithilfe des StateServices, um den passenden Aktuator anzusprechen.
ActuatorController	Empfängt einen Steuerwert, überprüft die Gültigkeit und setzt diesen mithilfe der ICadsRoboticArm API.

Tabelle 5.3: Bausteinsicht Level 3

5.4 Bausteinsicht Level 3 View

Komponente	**Beschreibung**
IO	Leitet die Eimgaben des Benutzers an den MoveAdapter weiter.
UI	Stellt dem Nutzer dar welche Roboter verfügbar sind, welcher ausgewählt ist und informiert über Fehler.

Tabelle 5.4: Bausteinsicht Level 3

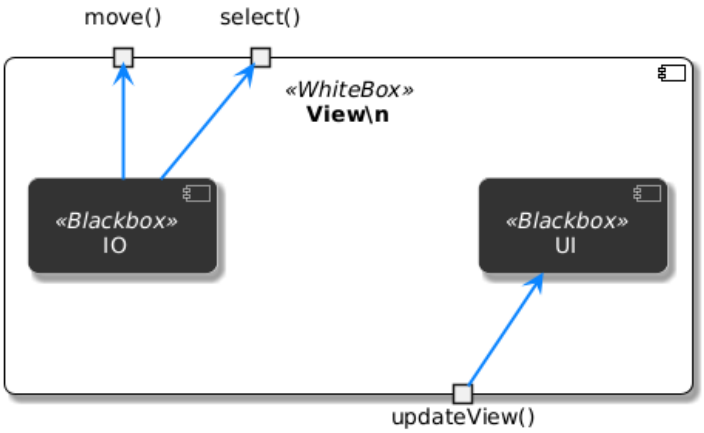

Abbildung 5.4: Bausteinsicht Level 3 View

6 Laufzeitsicht

6.1 Sequenzdiagramme

Szenario I: Auswählen eines Roboters

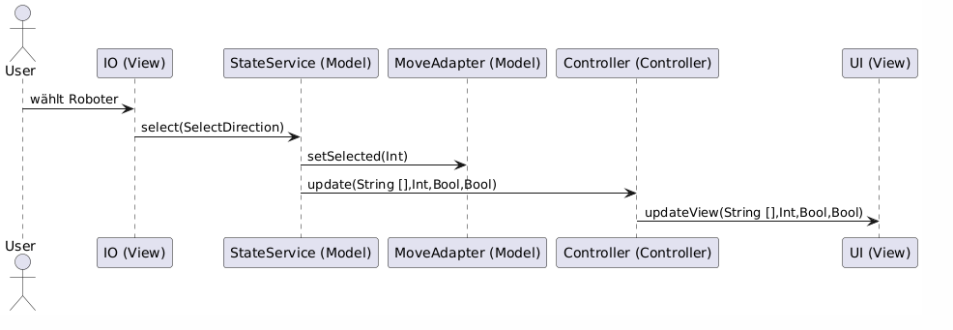

Abbildung 6.1: Auswahl des Roboters

Szenario II: Bewegungsbefehl über GUI

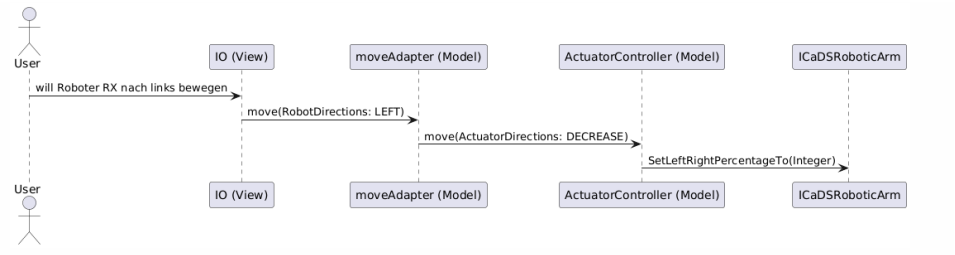

Abbildung 6.2: Bewegungsbefehl über GUI

6.2 FSM-Diagramme

FSM I: IO

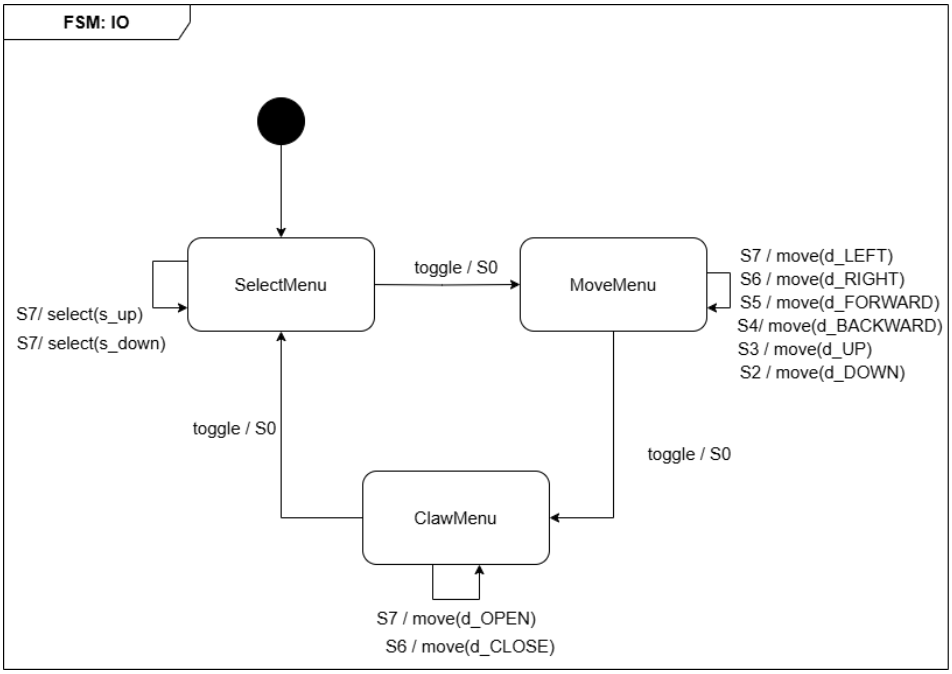

Abbildung 6.3: IO States Diagramm

6.3 Aktivitätsdiagramme

7 Verteilungssicht

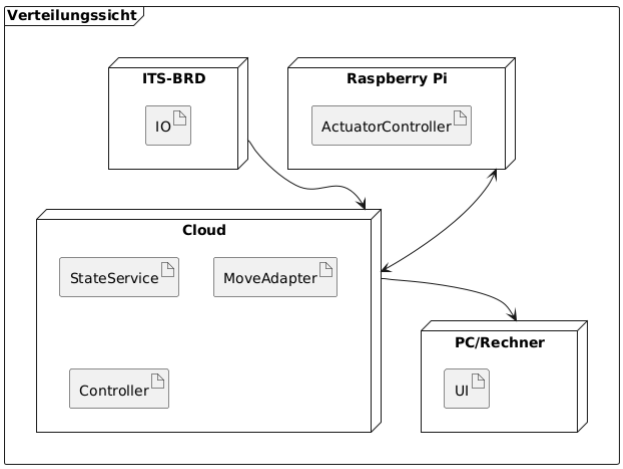

Abbildung 7.1: Deployment Diagramm

7.1 Begründung

Die Verteilung der Software-Bausteine auf verschiedene Hardware-Komponenten (PC, ITS-BRD, Raspberry Pi, Cloud) wurde gewählt, um folgende Systemanforderungen zu erfüllen:

- **Echtzeitnahe Verarbeitung:** Die Verwendung von UDP ermöglicht eine schnelle, verbindungslose Übertragung von Sensor- und Steuerdaten mit minimaler Latenz.
- **Lastverteilung:** Durch die Verlagerung von Steuerungs- und Zustandsdiensten in die Cloud wird die Last von den lokalen Geräten genommen und die Systemarchitektur bleibt flexibel und erweiterbar.
- **Hardware-Nähe:** Die IO-Module sowie der ActuatorController laufen auf Geräten in unmittelbarer Nähe zu den physischen Sensoren und Aktoren, um direkte Ansteuerung ohne zusätzliche Kommunikationsverzögerung zu ermöglichen.
- **Flexibilität:** Die Nutzung von RPC ermöglicht modulare, verteilte Anwendungsarchitekturen, bei denen die Services flexibel auf verschiedenen Knoten bereitgestellt werden können.

7.2 Qualitäts- und Leistungsmerkmale

Merkmal	**Beschreibung**
Latenz	Geringe Latenz durch den Einsatz des verbindungslosen UDP-Protokolls.
Verfügbarkeit	Die Cloud-Dienste können redundant und hochverfügbar ausgelegt werden.
Zuverlässigkeit	UDP bietet keine Garantie für die Paketübertragung. Fehlerbehandlung und Wiederholungsmechanismen müssen von der Anwendung sichergestellt werden.
Echtzeitfähigkeit	Die Übertragung über UDP ermöglicht schnelle Reaktionszeiten bei der Steuerung, ist jedoch aufgrund der Verbindungsfreiheit potentiell paketverlustanfällig.
Skalierbarkeit	Die Cloud-Komponenten sind horizontal skalierbar und können bei steigendem Datenaufkommen erweitert werden.

Tabelle 7.1: Qualitäts- und Leistungsmerkmale der Verteilungssicht

7.3 Zuordnung von Bausteinen zu Infrastruktur

Software-Baustein	**Hardware-Komponente**
UI	PC/Rechner
IO	ITS-BRD
ActuatorController	Raspberry Pi
StateService	Cloud
MoveAdapter	Cloud
Controller	Cloud

Tabelle 7.2: Zuordnung der Software-Bausteine zu den Infrastrukturkomponenten

8 Konzepte

8.1 Offenheit

8.2 Verteilungstranzparenzen

8.3 Kohärenz

Kohärenz wird dadurch sichergestellt, dass die Steuerung alle Roboter ausschliesslich über das ITS-Board zulässig ist.

8.4 Sicherheit (Safety)

Watchdogs auf dem Raspberry Pi des Roboterarms überwacht die eigene Verfügbarkeit. Wenn sich der Roboterarm durch einen vorherigen Aufruf vom ITS-Board bewegt, muss eine Verbindung zum ITS-Board bestehen. Bricht diese ab, oder das Netzwerk ist zu stark ausgelastet, muss der Roboterarm sofort anhalten. Wenn das ITS-Board einen Abbruch der Verbindung feststellt ist der Roboterarm als "nicht verfügbarërkennbar. Security ist keine Anforderung an das System.

8.5 Bedienoberfläche

Die GUI wird textuell auf dem Display des ITS-Boards dargestellt. Es wird auf Einfachheit und Eindeutigkeit geachtet.

8.6 Ablaufsteuerung

Ablaufsteuerung von IT-Systemen bezieht sich sowohl auf die an der (grafischen) Oberfläche sichtbaren Abläufe als auch auf die Steuerung der Hintergrundaktivitäten. Zur Ablaufsteuerung gehört daher unter anderem die Steuerung der Benutzungsoberfläche, die Workflow- oder Geschäftsprozessteuerung sowie Steuerung von Batchabläufen.

8.7 Ausnahme- und Fehlerbehandlung

8.8 Kommunikation

Für die Kommunikation wird das TCP-Protokoll genutzt. Die Steuerung der Roboterarme wird RPC durchgeführt.

8.9 Konfiguration

Die Raspberry Pi's und das ITS-Board bekommen eine feste IP-Adresse in einer bestehenden Netzwerkumgebung. Es existiert kein DHCP. Jeder Roboterarm hat eine eigene Hardgecodete Konfiguration, die dem ITS-Board mitgeteilt wird.

8.10 Logging, Protokollierung

Das Logging findet auf dem Raspberry Pi statt.

8.11 Plausibilisierung und Validierung

Wo und wie plausibilisieren und validieren Sie (Eingabe-)daten, etwa Benutzereingaben?

8.12 Sessionbehandlung

Sofern ein Roboterarm sich beim ITS-Board erfolgreich registriert hat, kann dieser, wenn erreichbar, angesteuert werden. Dafür wird eine TCP-Verbindung aufgebaut und danach mit RPC kommuniziert. Ein Watchdog auf beiden Seiten überwacht die Verbindung.

8.13 Skalierung

Dies Skalierung ist durch das /24 Netz begrenzt. Ansonsten können sich die Roboterarme beliebig mit ihren Kenndaten bei dem ITS-Board anmelden.

8.14 Verteilung

Das ITS-Board ruft per RPC die API der Middleware auf, diese wird dann in eine Bewegung des Roboterarms übersetzt. Die Nachrichten werden asynchron ausgetauscht. Die Anmeldung des Roboters erfolgt ebenfalls durch einen RPC aufruf. am ITS-Board.

9 Entwurfsentscheidungen

Entscheidung	Alternativen	Begründung	Woche
Client-Server	Schichten-Modell, Service-Orientiert, Ereignisgesteuert,Microservice, P2P, Serverless	Bi-Direktionale Kommunikation via RPC. Komplexität vermeiden (Microservices, Serverless)	2
RPC Kommunikation	MQTT, REST...	Funktionaler Ansatz; Es muss nur eine API bereitgestellt werden, sher effizient, kleiner Overhead, asynchrone Kommunikation möglich	2
Roboter-Software		Möglichkeit eine API bereitzustellen; Flexibilität, um Befehle zu verarbeiten (Sprache etc.); Erweiterbare Funktionalität	2
Logik liegt auf Raspberry Pi des zu steuernden R-Arms	Auf ITS-Board, Externer Server	Leistungsstark, kein Single Point Of Failure, Selbstverwaltung, erweiterte Skalierbarkeit	2
TCP	UDP	verbindungsorientiert, damit Daten vollständig sind	2
direkte Kopplung	Indirekte Kopplung,Losgekoppelte Kopplung, Strukturelle Kopplung	Verbindung ausschliesslich zu gesteuertem Roboter	2

Tabelle 9.1: Zentrale Entwurfsentscheidungen

10 Qualitätsszenarien

- Bezug auf section 1.2 - weniger wichtige Requirements müssen genannt werden

10.1 Quality Requirements Overview

-

10.1.1 Ziele für Software Engineering

Ziel	**Beschreibung**	**Metrik**
Funktionalität		Alle Abnahmetests werden erfolgreich bestanden
Zuverlässigkeit	Teilausfälle dürfen den Gesamtsystembetrieb nicht gefährden. Fehlererkennung und -toleranz müssen integriert sein.	Das System ist über dem gesamten Abnahmezeitraum stabil (ca. 1,5 h)
Skalierbarkeit	Zusätzliche Roboter oder Komponenten sollen ohne Änderungen an der bestehenden Architektur integrierbar sein.	Es können beliebig viele Roboter hinzugefügt und entfernt werden (0 - N)
Leistung	Reaktionszeiten auf Steuerbefehle und Ereignisse müssen innerhalb definierter Zeitgrenzen liegen.	Reaktionszeit max. 250 ms
Sicherheit (Safety)	Es bewegt sich immer genau ein Roboterarm. Sollte das System nicht wie gewünscht reagieren, wird ein sicherer Zustand erreicht Das System darf unter keinen Umständen eine Gefährdung für Personen/Gegenstände darstellen. Bei Fehlern muss sofort ein sicherer Zustand erreicht werden (z.B. Notstopp).	Reaktionszeit max. 250 ms, bis Roboterarm stoppt
Wartbarkeit	Der Code muss modular, gut dokumentiert und testbar sein. Fehlerdiagnose und Protokollierung sollen integriert sein.	
Portabilität	Die Software soll ohne großen Aufwand auf vergleichbaren Embedded-Systemen lauffähig sein.	
Benutzerfreundlichkeit	Konfiguration und Überwachung müssen intuitiv bedienbar und gut visualisiert sein.	Keine Einweisung erforderlich
Anpassbarkeit	Neue Funktionen, Sensoren oder Regeln sollen ohne tiefgreifende Änderungen am System integrierbar sein.	
Kompatibilität	Das System soll mit bestehenden Standards und Protokollen kommunizieren können.	

Tabelle 10.1: Qualitätsziele der Software Engineering

10.1.2 Ziele der Verteilte Systeme

Ziel	**Beschreibung**	**Metrik**
Ressourcenteilung	...	
Offenheit	...	
Skalierbarkeit	...	siehe Tabelle 10.3
Verteilung Transparenz	...	siehe Tabelle 10.4

Tabelle 10.2: Qualitätsziele der Verteilten Systeme

Skalierbarkeit

Ziel	**Metrik**	**Metrik**
Vertikale Skalierung	...	
Horizontale Skalierung	...	
Räumliche Skalierbarkeit		1
Funktionale Skalierbarkeit	...	
Administrative-Skalierbarkeit		1

Tabelle 10.3: Skalierbarkeit von verteilten Systemen

Verteilungs-Transparenzen

Ziel	**Beschreibung**	**Metrik**
Zugriffstransparenz	...	
Lokalitäts-Transparenz	...	
Migrationstransparenz	...	
Replikationstransparenz	...	
Fehlertransparenz	...	
Ortstransparenz	..	
Skalierbarkeits-Transparenz	...	

Tabelle 10.4: Verteilungs-Transparenzen

10.2 Bewertungsszenarien

ID	Context / Background	Source / Stimulus	Metric / Acceptance Criteria
QS-1	Gesamtsystem betriebsbereit. Der Roboter befindet sich im Ruhezustand (Stop).	Bediener sendet Bewegungsbefehl und das Stromkabel des ITS Board wird gezogen	Roboter geht innerhalb von 250 ms in den Ruhezustand.
QS-2	Gesamtsystem betriebsbereit. Der Roboter befindet sich im Ruhezustand.	Bediener sendet Bewegungsbefehl und das Stromkabel des Raspberry Pi wird herausgezogen.	Roboter innerhalb von 250 ms in den Ruhezustand.

Tabelle 10.5: Bewertungsszenarien nach q42-Modell

11 Risiken

Beispieleintrag für ein Glossarverweis: Eine Liste identifizierter technischer Risiken oder technischer Schulden, nach Priorität geordnet!

11.1 Ziel des chapters

Frühzeitige Identifikation und Dokumentation technischer Risiken
Aufzeigen von bewusst eingegangenen technischen Schulden
Unterstützung bei Risikomanagement und fundierter Entscheidungsfindung

11.2 Technische Risiken

Fehlende Erfahrungen, Komplexität und Externe Abhängigkeiten

11.3 Technische Schulden

Hardcodierung und Fehlende Test

Form:
Eine Liste der jeweiligen Risiken und Schulden. Zusätzlich sollten vorransgehensweise und lösungswege hinzugefügt werden. Wie ist man mit den Risiken umgegangen? Warum nimmt man die Schuld in kauf?

12 Glossar

Begriff	**Definition Erklährung**
GUI-Software	Überbegriff für Steuerungssoftware, die mit Hilfe des LCD Displays dem Nutzer die Steuerung der Roboter ermöglicht. Es handlet sich hierbei um ein Programm/Executable
Roboter-Software	Überbegriff für Software, die für die direkte Steuerung eines Roboters zuständig ist. Es handlet sich hierbei um ein Programm/Executable
Roboter/Roboterarm	Hardware, umfasst den Arm selbst sowie das entsprechende RasberryPi

Tabelle 12.1: Glossar