Verteilte Systeme

VS Praktikum SoSe 2025

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

11. Juli 2025

Inhaltsverzeichnis

1 Einführung und Ziele

1.1 Aufgabenstellung

Es soll eine Applikation entwickelt werden, die beliebig hinzugefügte Roboterarme erkennt und ansteuern kann. Jeder Roboter besteht aus vier unabhängigen Motoren. Über ein ITS-Board (STM32F4) sollen die Roboterarme innerhalb eines kontrollierten Areals (z.B. BT7 R7.65 – als realer Testbereich) sicher gesteuert werden. Mögliches Fehlverhalten der Software oder Architektur soll keine Gefahr für Anwesende darstellen. Der Benutzer soll einen zu steuernden Roboterarm auswählen und diesen anschließend mit den folgenden Bewegungen steuern:

- 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 - 254)
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.

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.

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.
- **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

Die Applikation 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 einer Benutzeroberfläche (UI) bereitgestellt, die eine intuitive Orientierung und Bedienung der Roboterarme ermöglicht.

3.1.1 Use Cases

ID	**Name**	**Beschreibung**
U1	Roboterarm auswählen	Der Nutzer ist in der Lage einen vollständigen Roboterarm auswählen zu können. Der Nutzer hat jederzeit eine Übersicht, welche Roboterarm ausgewählt werden können und ob keiner auswählbar ist
U2	Bewegung auslösen	Der Nutzer kann die Bewegung des Roboterarms auslösen und bekommt Feedback.
U3	Bewegungsrichtung unterscheiden	Die vom Nutzer ausgelöste Bewegungsrichtung wird dem richtigen Motor zugewiesen
U4	Neuen Motor erkennen	Die einzelnen Motoren melden sich an. Sobald ein neuer vollständiger Roboterarm (4 Motoren) angesteuert werden kann, ist dieser auswählbar und steuerbar.

3.1.2 Fachliche Randbedingungen

- Es können bis zu 254 Roboterarme ausgewählt werden
- Jeder Roboterarme hat 4 Motoren, die unabhängig voneinander sind
- Teile der Applikation sind auf dem ITS-Board zu implementieren
- Die Applikation ist sicher. Personen dürfen nicht zu schaden kommen.

3.2 Technischer Kontext

Die Applikation besteht aus einem ITS-Board und jeder Roboterarm wird von einem vorgeschalteten Raspberry Pi angesteuert. Es existiert eine Benutzeroberfläche (UI). Optional kann die Applikation um externe Dienste, die auf anderen Plattformen, wie der ICC-Cloud erweitert werden.

3.2.1 Technische Anforderungen an die Middleware

Um die fachlichen Use Cases umzusetzen, muss die Middleware folgende technische Funktionen bereitstellen:

- **Roboterarm auswählen U1:**
  - Der Nutzer sucht durch eine Eingabe den gewünschten vollständigen Roboterarm aus
  - Die Applikation wertet die Auswahl aus, ob die Gruppe auswählbar ist
  - Die Gruppe aus Motoren wird als äusgewählt"für die Ansteuerung markiert
  - Die Auswahl wird als ërfolgreich"markiert
  - dem Nutzer werden die Auswahländerungen angezeigt und bekommt die ërfolgreich"Mitteilung
- **Bewegung auslösen U2:**
  - Der Nutzer betätigt eine Auswahl für die Bewegungsrichtung
  - Der Steuerungsbefehl wird inklusive Richtung der Applikation übergeben
  - Der Steuerungsbefehl wird übersetzt
  - Die Bewegung wird anhand einer Prozentualen Bewegung durchgeführt
  - Der erfolgreiche Befehl wird dem Nutzer mitgeteilt
- **Bewegungsrichtung unterscheiden U3:**
  - Der Steuerungsbefehl wird inklusive Richtung der Applikation übergeben
  - Anhand der Steuerungsrichtung wird entschieden welcher Motor angesprochen wird
  - Es wird überprüft, ob der Motor angesprochen werden kann/auswählbar ist
  - Der Steuerungsbefehl wird an den ausgewählten Roboterarm weitergegeben
- **Neuen Motor erkennen U4:**
  - Ein neuer Motor wird angeschlossen
  - Die Software des Motors sendet einen Anmelde-Anfrage mit dessen Zugehörigkeit und Bewegungsfunktion
  - Der neue Motor wird eingetragen

  - Es wird geprüft, ob durch den Motor eine kompletter Roboterarm nun auswählbar ist
  - dem Nutzer wird ggf. mitgeteilt, dass ein neuer Roboterarm auswählbar ist

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.

4 Lösungsstrategie

Als grundlegendes Architekturprinzip wurde das Model-View-Controller-(MVC)-Muster eingesetzt, um eine klare Trennung zwischen Anwendungslogik (Model), Steuerung (Controller) und Darstellung (View) sicherzustellen [1]. Dieses Prinzip der *Separation of Concerns* bildete den Ausgangspunkt der Umsetzung und erleichterte den strukturierten Einstieg in die Entwicklung. Durch die konsequente Aufteilung in klar abgegrenzte Verantwortlichkeiten konnten die funktionalen Anforderungen frühzeitig in kleinere, handhabbare Arbeitspakete überführt werden. Dies förderte sowohl die Parallelisierung einzelner Entwicklungsaufgaben als auch die Wartbarkeit und Erweiterbarkeit der Lösung im weiteren Verlauf des Projekts.

[1] `https://github.com/scimbe/vs_script/blob/main/vs-script-first-v01.pdf` S.90-94

4.1 Controller

In der vorliegenden Architektur übernimmt der Controller nicht mehr die klassische Rolle eines Observers, sondern fungiert als schlanker, zustandsloser Vermittler zwischen Modell und View im Sinne des MVC-Musters.
Anstelle eines ereignisgesteuerten oder abfragenden (Pull-)Ansatzes sendet das Modell (z. B. der *StateService*) in regelmäßigen Intervallen aktiv (Push-basiert)[2] seinen aktuellen Zustand an den Controller.
Dieser verarbeitet die empfangenen Daten nicht weiter, sondern leitet sie unmittelbar und unverändert an die View weiter.
Dieses Verfahren gewährleistet durch fehlertolerantes Verhalten[3] eine konsistente Darstellung des Systemzustands auch dann, wenn die View zum Zeitpunkt der Aktualisierung nicht verfügbar ist oder erst verzögert gestartet wird. Sobald sie aktiv ist, erhält sie automatisch den zuletzt übermittelten Zustand.
Der Controller bleibt vollständig zustandslos[4]: Er speichert weder frühere Zustände noch verarbeitet er Eingaben oder Geschäftslogik.
Zudem erfolgt keine Rückrichtung von der View zum Modell.
Die klare Trennung der Verantwortlichkeiten sowie der rein gerichtete Datenfluss erhöhen die Robustheit der Anwendung, reduzieren potenzielle Fehlerquellen und fördern die Wiederverwendbarkeit der Komponenten.

Funktion	**Voraussetzung**	**Semantik**
void update(robots: Byte[], selected: int, error: bool, confirm: bool)	Modell sendet periodisch aktuellen Zustand	Controller leitet Daten unmittelbar an View weiter
void reportHealth(serviceName: String, subscription: String)	Gültiger Servicename und Abonnement vorhanden	Meldet den Gesundheitszustand eines Dienstes an das Modell zurück.

Tabelle 4.1: Funktionen des Controllers

[2] `https://github.com/scimbe/vs_script/blob/main/vs-script-first-v01.pdf` S.209-211
[3] `https://github.com/scimbe/vs_script/blob/main/vs-script-first-v01.pdf` Seite 62, 200
[4] `https://github.com/scimbe/vs_script/blob/main/vs-script-first-v01.pdf` siehe S. 57-58,89,137

4.2 Model

Im Rahmen der gewählten MVC-Architektur übernimmt das *Model* die zentrale Rolle bei der Verwaltung des Anwendungszustands und der Geschäftslogik.[5]. Gemäß dem Prinzip der *Separation of Concerns* ist das Model ausschließlich für Datenhaltung, Zustandsverwaltung und domänenspezifische Logik zuständig – losgelöst von Darstellung und Steuerung. Die nachfolgenden Funktionen sind bewusst im Model (StateService) verankert, da sie direkt die Verwaltung und den Zustand der Roboter betreffen.

Die Kapselung dieser Funktionen im Model stellt sicher, dass Zustandslogik, Validierung und Datenverarbeitung zentral gebündelt sind[6]. Die View bleibt vollständig abstrahiert und erhält ausschließlich aktualisierte Daten über den Controller. Da der Datenfluss unidirektional vom Modell über den Controller zur View erfolgt, wird eine klare Trennung der Verantwortlichkeiten sichergestellt.

4.2.1 StateService

Funktion	**Voraussetzung**	**Semantik**
void setError(error: boolean, confirm: boolean)	Änderung des Fehler- oder Bestätigungsstatus notwendig	Setzt den Fehler- und Bestätigungsstatus und benachrichtigt den Controller.
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 registerActuator(actuatorName: String, isAlive: boolean)	Motorname im richtigen Format (z. B. R1A1)	Registriert den Motor und ergänzt ggf. die Roboterverfügbarkeitsliste.
void reportHealth(serviceName: String, subscription: String)	Gültiger Servicename und Abonnement vorhanden	Meldet den Gesundheitszustand eines Dienstes an das Modell zurück.
boolean isAvailable()	keine	Prüft, ob alle vier Aktoren (A1–A4) des Roboters aktiv sind. Gibt `true` zurück, wenn der Roboter vollständig verfügbar ist.

Tabelle 4.2: Funktionen der Komponente StateService

[5] `https://github.com/scimbe/vs_script/blob/main/vs-script-first-v01.pdf` S.90-94
[6] `https://github.com/scimbe/vs_script/blob/main/vs-script-first-v01.pdf` S.91 und S.137

Der `StateService` bildet das zentrale Zustandsmodell der Anwendung und ist als Teil der Model-Komponente im Sinne des MVC-Musters zu verstehen. Er verwaltet den internen Systemzustand wie Fehlerstatus, Aktorverfügbarkeit, Roboterselektion und Gesundheitsinformationen. Alle domänenspezifischen Funktionen, die sich auf den Roboterzustand und dessen Verwaltung beziehen, sind hier gebündelt. Somit kapselt der `StateService` nicht nur die Daten, sondern auch die zugehörige Logik zur Interpretation und Aktualisierung ein zentrales Merkmal von Modellkomponenten in verteilter Anwendungsarchitektur.

4.2.2 MoveAdapter

Funktion	**Voraussetzung**	**Semantik**
void move(robotDirection: RobotDirection)	Ein Roboter ist im StateService ausgewählt und verfügbar	Führt eine Bewegungsanweisung in der angegebenen Richtung für den aktuell ausgewählten Roboter aus.
void setSelected(selected: String)	Der angegebene Robotername existiert im StateService	Setzt den angegebenen Roboter als aktuell ausgewählten und initialisiert ggf. die Bewegung. Fehler werden im StateService registriert.

Tabelle 4.3: Funktionen des MoveAdapter

Der `MoveAdapter` ist eine zustandsbehaftete Komponente innerhalb des Modells. Er verwaltet die Auswahl eines Roboters und führt kontextbezogene Bewegungsbefehle aus. Die Methode `setSelected()` speichert den ausgewählten Roboter intern, während `move()` Bewegungen für genau diesen ausführt. Durch die lokale Zustandsführung wird die Trennung von Logik, Steuerung und Darstellung gestärkt und die Konsistenz der Bewegungsoperationen sichergestellt.

4.2.3 ActuatorController

Funktion	Voraussetzung	Semantik
void move(ActuatorDirection actuatorDirection)	Der Aktuator ist initialisiert	Bewegt den Aktuator in die angegebene Richtung (INCREASE oder DECREASE) und setzt den neuen Wert.

Tabelle 4.4: Funktionen der Komponente ActuatorController

Der `ActuatorController` ist eine funktionale Komponente innerhalb des Anwendungskerns, die für die konkrete Steuerung einzelner Aktoren verantwortlich ist. Sie kapselt die Logik zur Positionsveränderung eines Aktors entlang einer vorgegebenen Richtung (`INCREASE` oder `DECREASE`). Die Methode `move()` setzt den neuen Positionswert und stellt sicher, dass nur initialisierte Aktoren angesteuert werden. Damit ergänzt der `ActuatorController` die Bewegungslogik auf unterer Ebene und unterstützt die feingranulare Steuerung innerhalb der Gesamtarchitektur.

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(robotBitmap: Byte[],selected: int, error: bool, confirm: bool)	Gültige Modell-Daten vorhanden	View-Schnittstelle. Aktualisiert die UI mit den neuesten Roboter-Daten und Statusinformationen (Fehler, Bestätigung).

Tabelle 4.6: UI Funktionen

Die UI-Funktionen übernehmen im Rahmen des MVC-Musters die rein visuelle Darstellung des aktuellen Modellzustands. Die zentrale Methode `updateView()` wird vom Controller innerhalb der Anwendungsschicht aufgerufen, sobald neue Zustandsinformationen aus dem Modell vorliegen. Die Benutzeroberfläche wird in einem HTML-basierten Webbrowser dargestellt. Diese Gestaltung folgt dem Prinzip der Client-Server-Architektur: Der Browser fungiert als leichtgewichtiger Client, der ausschließlich für die Präsentation zuständig ist, während Geschäftslogik und Zustandsverwaltung vollständig in der Anwendung verbleiben.[7] Dadurch wird eine plattformunabhängige und benutzerfreundliche Bedienoberfläche ermöglicht.[8]

[7] `https://github.com/scimbe/vs_script/blob/main/vs-script-first-v01.pdf` S.82 f.
[8] `https://github.com/scimbe/vs_script/blob/main/vs-script-first-v01.pdf` S.23

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

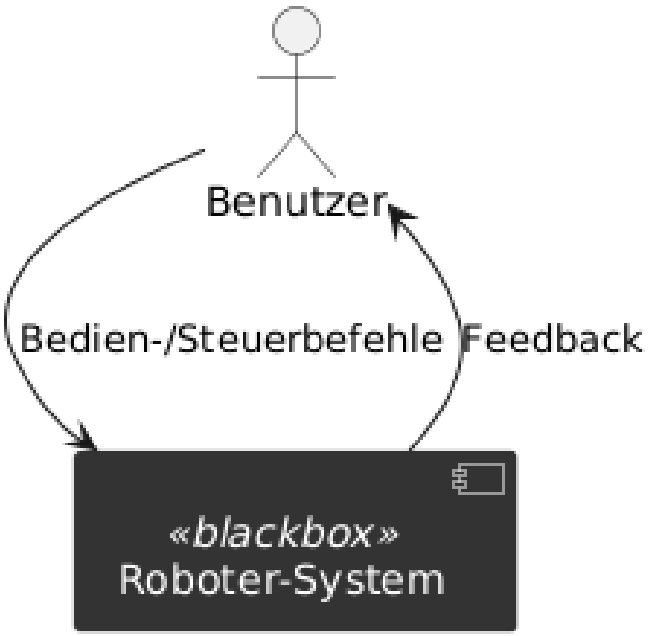

Abbildung 5.1: Bausteinsicht Level 1

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

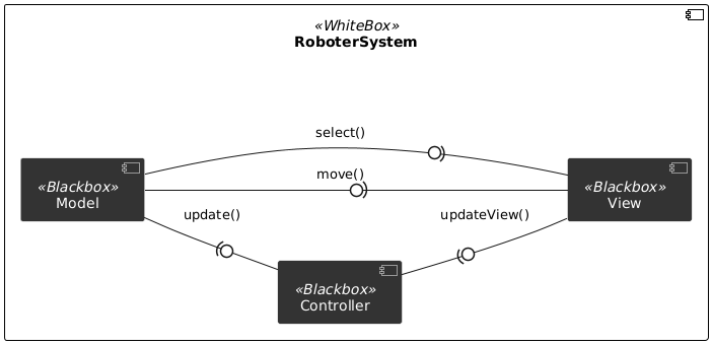

Abbildung 5.2: Bausteinsicht Level 2

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

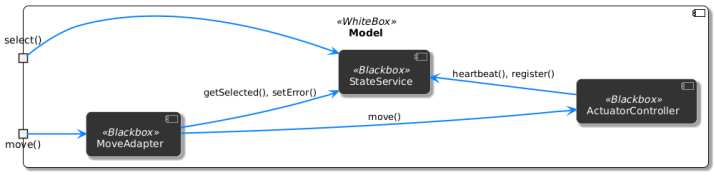

Abbildung 5.3: Bausteinsicht Level 3 Model

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

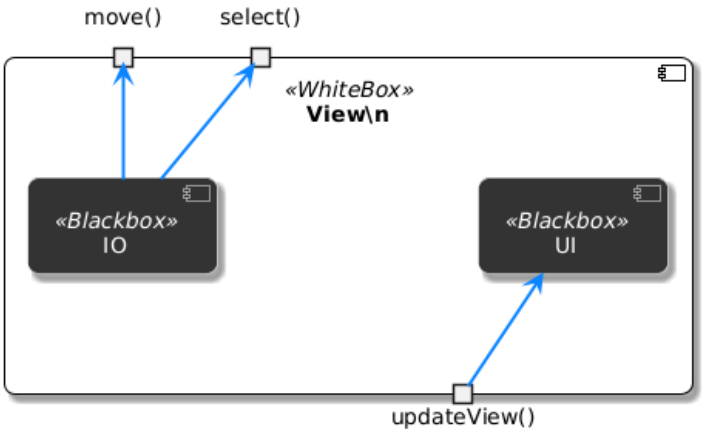

Abbildung 5.4: Bausteinsicht Level 3 View

6 Laufzeitsicht

6.1 Sequenzdiagramme

Szenario I: Auswählen eines Roboters

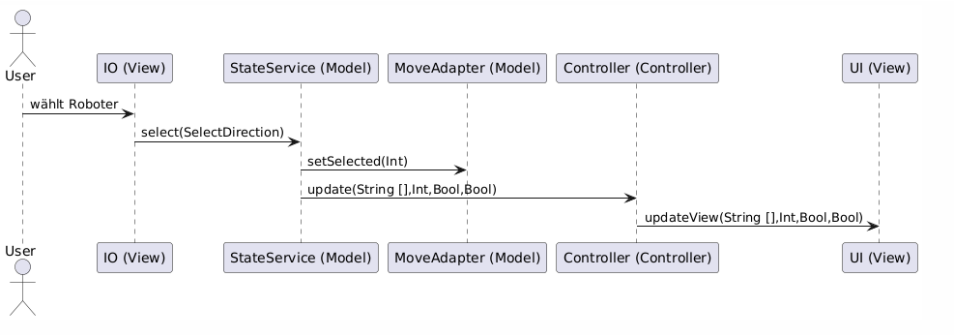

Abbildung 6.1: Auswahl des Roboters

Szenario II: Bewegungsbefehl über GUI

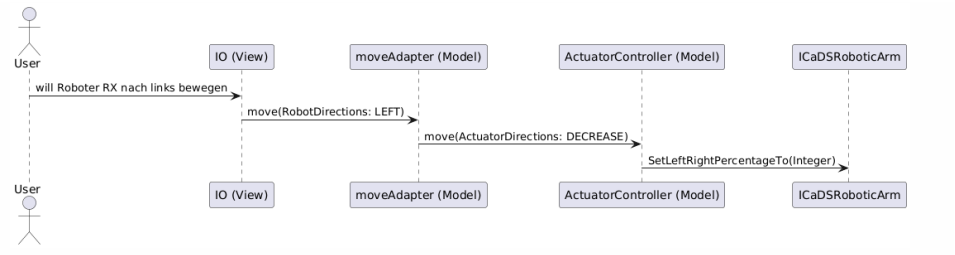

Abbildung 6.2: Bewegungsbefehl über GUI

6.2 FSM-Diagramme

FSM I: IO

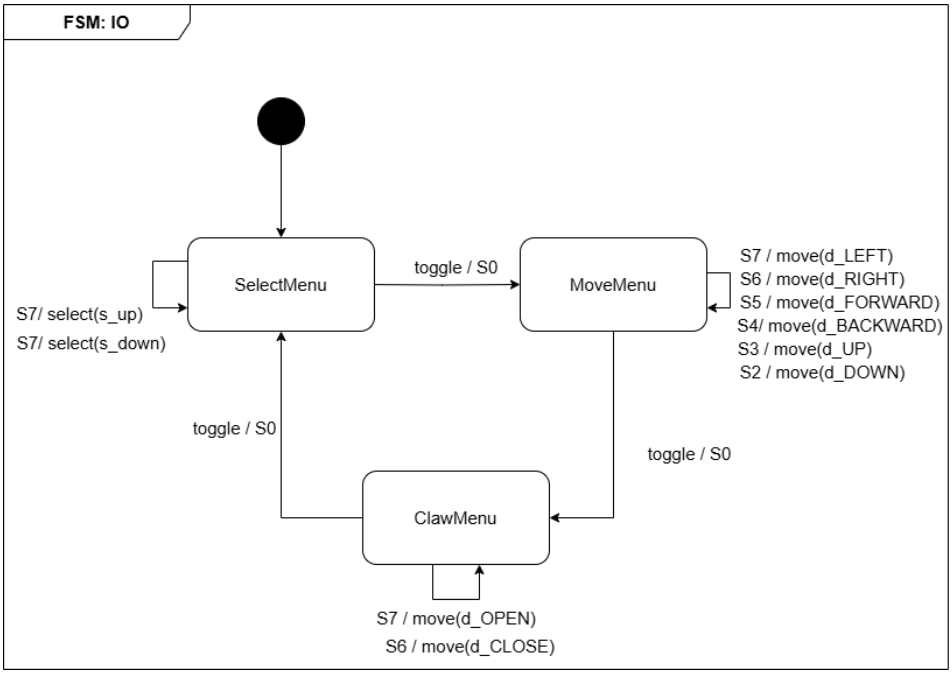

Abbildung 6.3: IO States Diagramm

6.3 Aktivitätsdiagramme

7 Verteilungssicht

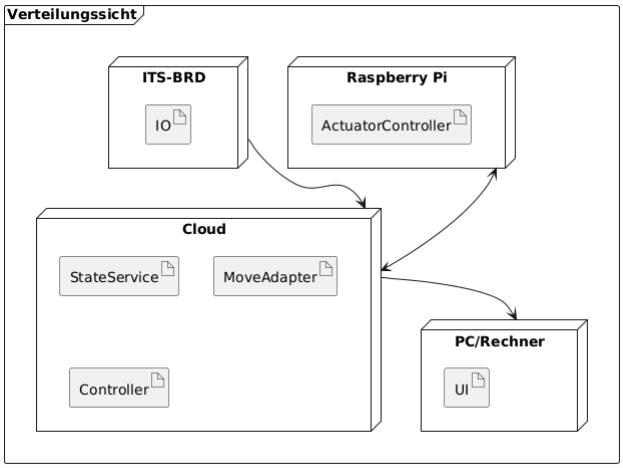

Abbildung 7.1: Deployment Diagramm

7.1 Begründung

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

7.2 Qualitäts- und Leistungsmerkmale

Merkmal	Beschreibung

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 Sicherheit (Safety)

Für jede Betätigung des Nutzer wird nun ein Bewegungsbefehl an den Roboter weitergegeben. Sobald der Nutzer die Taste loslässt, hört der Roboter auf sich zu bewegen.

8.2 Bedienoberfläche

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

8.3 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.

8.4 Ausnahme- und Fehlerbehandlung

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**
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

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