Verteilte Systeme

VS Praktikum SoSe 2025

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

17. 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 Roboterarm 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 vorwärts
- Roboterarm zurück
- 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 Roboterarme oder Komponenten sollen ohne Änderungen an der bestehenden Architektur integrierbar sein.	Es können beliebig viele Roboterarme 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 Roboterarmsteuerung 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 Roboterarme 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 Roboterarms 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	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 Roboterarm 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 Applikation

Um die fachlichen Use Cases umzusetzen, muss die Applikation 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 ausgewählt für die Ansteuerung markiert
  - Die Auswahl wird als erfolgreich markiert
  - dem Nutzer werden die Auswahländerungen angezeigt und bekommt die erfolgreich 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
- **Neuen Motor erkennen U3:**
  - 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 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.
Sobald sie aktiv ist, erhält sie automatisch den zuletzt übermittelten Zustand.
Der Controller bleibt vollständig zustandslos[3]: Er speichert weder frühere Zustände noch verarbeitet er Eingaben oder beinhaltet 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: String[], 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` 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.[4] Gemäß dem Prinzip der *Separation of Concerns* ist das Model ausschließlich für Datenhaltung und Zustandsverwaltung zuständig und ist dabei losgelöst von Darstellung und Steuerung. Die nachfolgenden Funktionen sind bewusst im Model (StateService) verankert, da sie direkt die Verwaltung und den Zustand des Roboterarms betreffen.

Die Kapselung dieser Funktionen im Model stellt sicher, dass Zustandslogik, Validierung und Datenverarbeitung zentral gebündelt sind[5]. Die View bleibt vollständig abstrahiert und erhält ausschließlich aktualisierte Daten über den Controller.

4.2.1 StateService

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

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 Roboterarme ist nicht leer	Wählt einen Roboterarm aus der Liste der verfügbaren Roboterarme 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 Roboterarmverfügbarkeitsliste.
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 Roboterarms aktiv sind. Gibt `true` zurück, wenn der Roboterarm vollständig verfügbar ist.
checkServiceTimeouts()	keine	Überprüft alle 250ms ob services sich mindestens 1 mal pro Sekunde melden, löscht Einträge wenn diese kein Heartbeat mehr verschicken.
checkSubscriptionTimeouts()	keine	Überprüft alle 250ms of die subscription noch aktiv ist, ansonsten wird beim Watchdog erneut subscribed.
sendUpdate()	Eine Controller Instanz muss erstellt worden sein	Sendet alle 250ms periodisch Zustandsänderungen an den Controller

Tabelle 4.2: Funktionen der Komponente StateService

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, Roboterarmselektion und Gesundheitsinformationen. Alle domänenspezifischen Funktionen, die sich auf den Roboterarmzustand 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 Roboterarm ist im StateService ausgewählt und verfügbar	Führt eine Bewegungsanweisung in der angegebenen Richtung für den aktuell ausgewählten Roboterarm aus.
void setSelected(selected: String)	Der angegebene Roboterarmname existiert im StateService	Setzt den angegebenen Roboterarm 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 Roboterarms und führt kontextbezogene Bewegungsbefehle aus. Die Methode `setSelected()` speichert den ausgewählten Roboterarm 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 eines Aktuators verantwortlich ist. Sie kapselt die Logik zur Positionsveränderung eines Aktuators entlang einer vorgegebenen Richtung (`INCREASE` oder `DECREASE`). Die Methode `move()` setzt den neuen Positionswert und stellt sicher, dass nur der initialisierte Aktuator angesteuert wird. 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 Interne IO-Funktionen

Funktion	Voraussetzung	Semantik
void readButtons()	Eingabetasten sind verbunden	Liest den aktuellen Zustand aller Tasten aus.
bool isAnyButtonPressed(lastBtn: int)	Taste wurde zuvor gedrückt	Überprüft, ob dieselbe Taste weiterhin gedrückt wird.
void toggleState()	Zustand soll gewechselt werden	Wechselt den aktuellen Zustand. Kann erweitert werden, um zusätzliche Funktionen auszulösen.

Tabelle 4.5: Interne IO-Funktionen

4.3.2 UI Funktionen

Funktion	**Voraussetzung**	**Semantik**
void updateView(robots: String[],selected: int, error: bool, confirm: bool)	Gültige Modell-Daten vorhanden	View-Schnittstelle. Aktualisiert die UI mit den neuesten Roboterarm-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. Ein HTTP-Server stellt die statischen HTML-Inhalte klassisch im Client-Server-Modell bereit. Für die Aktualisierung des UI-Zustands kommt zusätzlich eine WebSocket-Verbindung zum Einsatz, über die neue Modellinformationen entsprechend unserer definierten Datenflussrichtung direkt vom Server an den Browser übertragen werden. Auf diese Weise wird die Oberfläche automatisch an neue Zustände angepasst, ohne dass wiederholte Client-Anfragen notwendig sind.

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

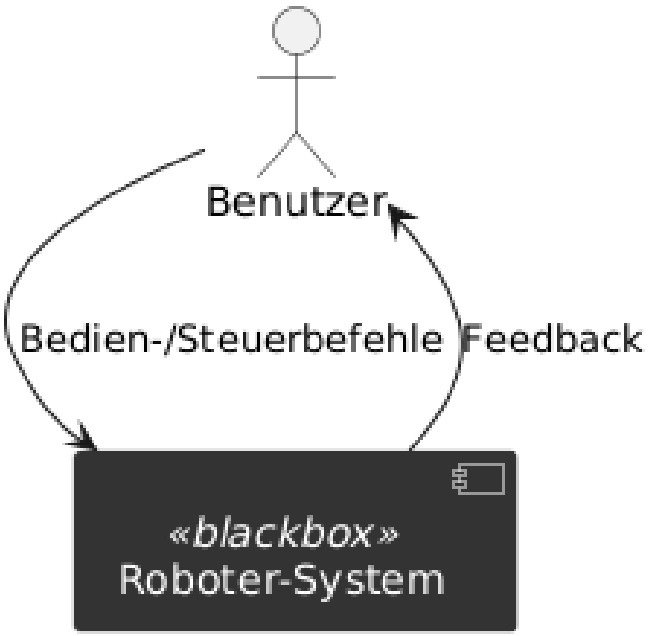

Abbildung 5.1: Bausteinsicht Level 1

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

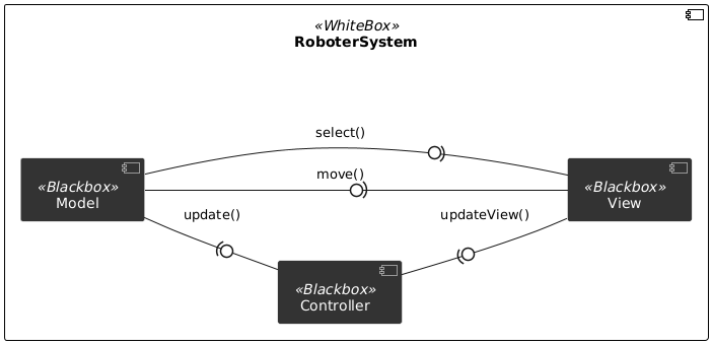

Abbildung 5.2: Bausteinsicht Level 2

Komponente	**Beschreibung**
Model	Beinhaltet die Geschäftslogik. Sendet periodisch Zustandsinformationen an den Controller.
Controller	Der Controller leitet die Informationen vom StateService 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

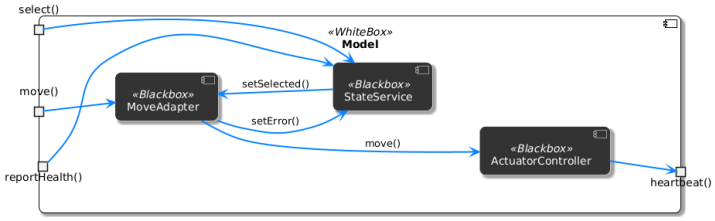

Abbildung 5.3: Bausteinsicht Level 3 Model

Komponente	**Beschreibung**
StateService	Speichert die Zustandsänderungen, informiert periodisch den Controller. Informiert den MoveAdapter über den derzeitigen selektierten Roboterarm. Durch die reporthealth() Methode, werden die Zustände der verschiedenen Aktoren überwacht. Darauf aufbauend wird die Liste der verfügbaren Roboter aufgebaut.
MoveAdapter	Empfängt einen Steuerungsbefehl vom View, übersetzt diesen um den passenden Aktuator anzusprechen.
ActuatorController	Empfängt einen Steuerwert, überprüft die Gültigkeit und setzt diesen mithilfe der ICadsRoboticArm API. Sendet einen heartbeat an den Watchdog.

Tabelle 5.3: Bausteinsicht Level 3

5.4 Bausteinsicht Level 3 View

Komponente	**Beschreibung**
IO	Leitet die Steuerungsbefehle des Benutzers an den MoveAdapter weiter. Leitet Selektierungsbefehle an den StateService.
UI	Stellt dem Nutzer dar welche Roboter verfügbar sind, welcher ausgewählt ist und gibt Rückmeldungen (Fehler, Bestätigungen).

Tabelle 5.4: Bausteinsicht Level 3: View

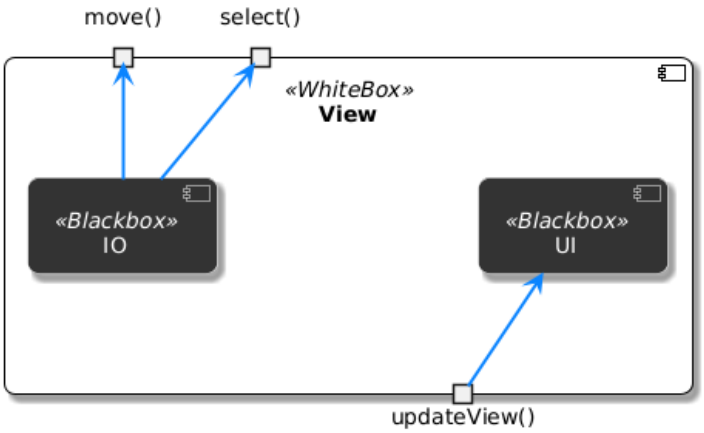

Abbildung 5.4: Bausteinsicht Level 3 View

5.5 Bausteinsicht Level 3 Controller

Komponente	**Beschreibung**
Controller	Leitet Nachrichten des StateService an das View weiter.

Tabelle 5.5: Bausteinsicht Level 3: Controller

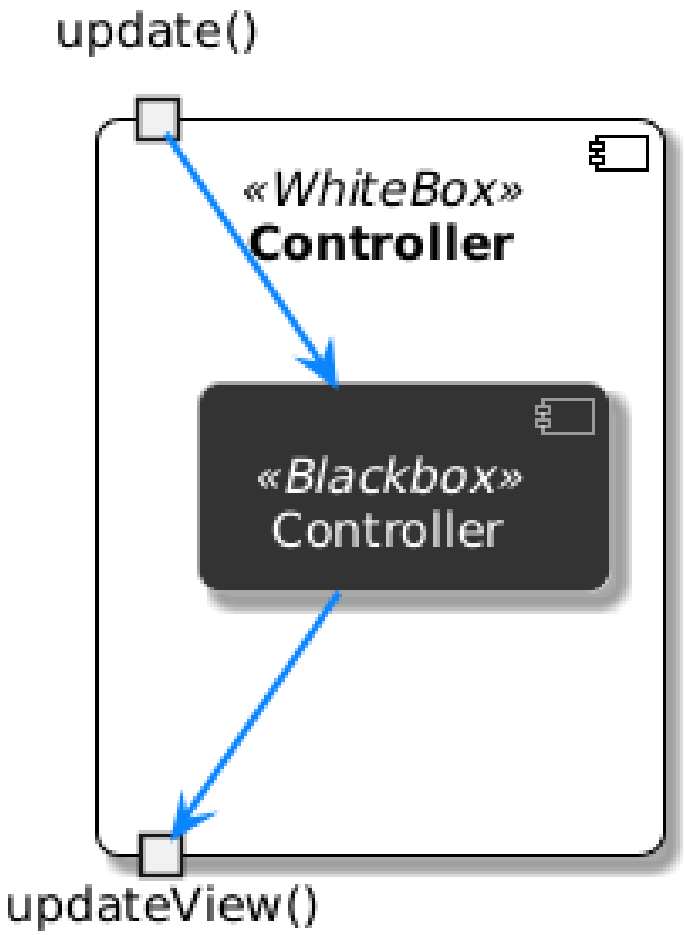

Abbildung 5.5: Bausteinsicht Level 3 Controller

6 Laufzeitsicht

6.1 Sequenzdiagramme

Szenario I: Auswählen eines Roboterarms

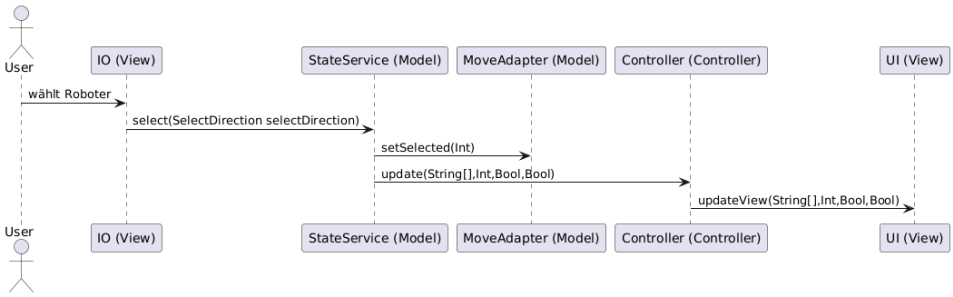

Abbildung 6.1: Auswahl des Roboterarms

Szenario II: Bewegungsbefehl über GUI

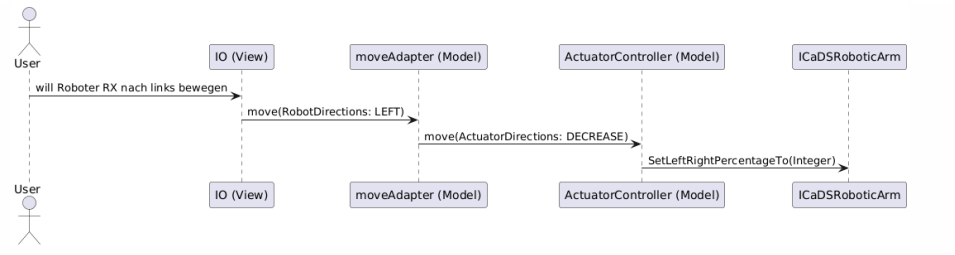

Abbildung 6.2: Bewegungsbefehl über GUI

Szenario III: Registrierung von Motoren

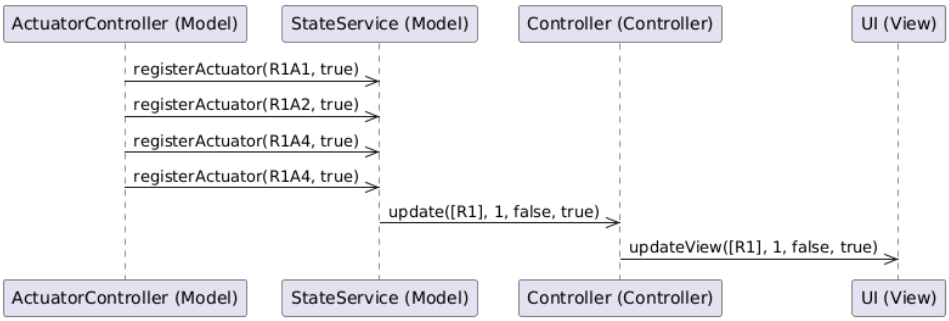

Abbildung 6.3: Registrierung von Motoren

6.2 FSM-Diagramme

FSM I: IO

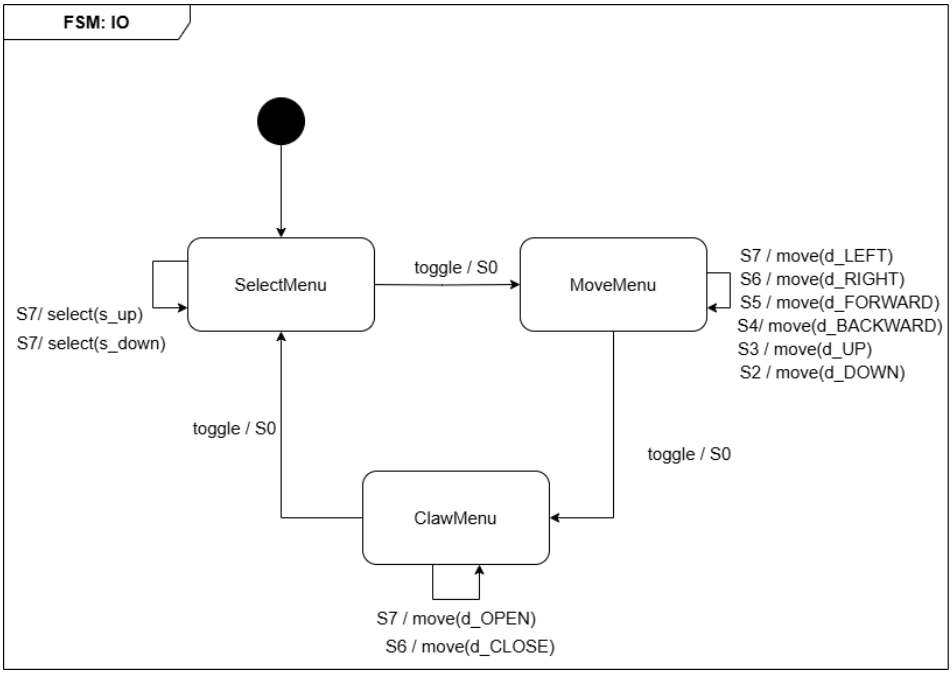

Abbildung 6.4: IO States Diagramm

7 Verteilungssicht

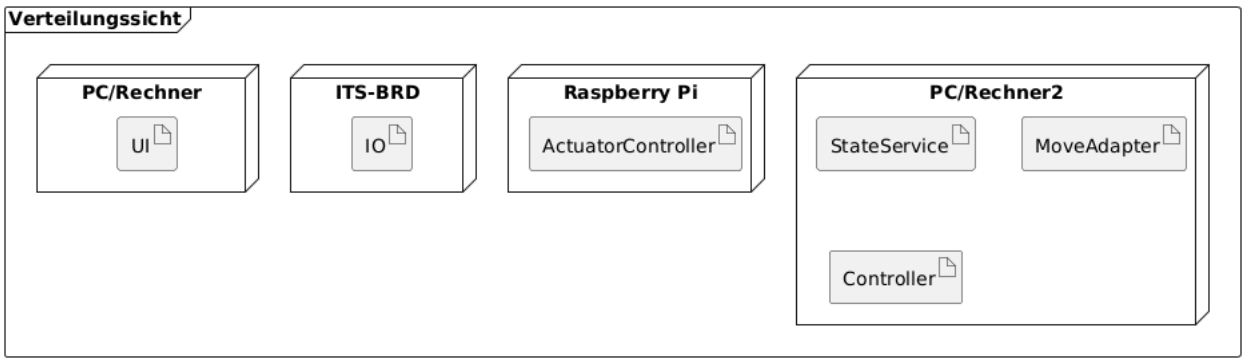

Abbildung 7.1: Verteilungssicht Applikation

8 Konzepte

8.1 Sicherheit (Safety)

Für jede Tastenbetätigung des Nutzers wird nun ein Bewegungsbefehl an den Roboterarm weitergegeben. Dabei ist auch das gedrückthalten der Taste nur ein Bewegungsbefehl. Gleichzeitiges betätigen der Taster funktioniert nicht.

8.2 Bedienoberfläche

Die Bedienung wird textuell auf dem Display des ITS-Boards dargestellt. Es wird auf Einfachheit und Eindeutigkeit geachtet. Außerdem werden die verfügbaren, sowie der ausgewählte Roboterarm in einem Webbrowser dargestellt. Auch werden Fehler, sowie Bestätigungen visuell in dem Webbrowser dargestellt.

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.

9 Qualitätsszenarien

9.1 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	Roboterarm geht innerhalb von 250 ms in den Ruhezustand.
QS-2	Gesamtsystem betriebsbereit. Der Roboterarm befindet sich im Ruhezustand.	Bediener sendet Bewegungsbefehl und das Stromkabel des Raspberry Pi wird herausgezogen.	Roboterarm innerhalb von 250 ms in den Ruhezustand.

Tabelle 9.1: Bewertungsszenarien nach q42-Modell

9.1.1 End-to-End Tests

Test-ID	Erwartung	Ergebnis	
1	IO -> ActuatorController	Ein Bewegungsbefehl nach links hat eine Bewegung des Roboterarms nach links zur Folge	Bestanden
2	IO -> ActuatorController	Ein Bewegungsbefehl nach rechts hat eine Bewegung des Roboterarms nach rechts zur Folge	Bestanden
3	IO -> ActuatorController	Ein Bewegungsbefehl nach oben hat eine Bewegung des Roboterarms nach oben zur Folge	Bestanden
4	IO -> ActuatorController	Ein Bewegungsbefehl nach unten hat eine Bewegung des Roboterarms nach unten zur Folge	Bestanden
5	IO -> ActuatorController	Ein Bewegungsbefehl nach vorne hat eine Bewegung des Roboterarms nach vorne zur Folge	Bestanden
6	IO -> ActuatorController	Ein Bewegungsbefehl nach hinten hat eine Bewegung des Roboterarms nach hinten zur Folge	Bestanden
7	IO -> ActuatorController	Ein Bewegungsbefehl des schließens hat eine anteilige Schließung der Klaue des Roboterarms zur Folge	Bestanden
8	IO -> ActuatorController	Ein Bewegungsbefehl des öffnens hat eine anteilige Öffnung der Klaue des Roboterarms zur Folge	Bestanden
9	IO -> UI	Wenn der Selektionsbefehl nach oben ausgeführt wird, muss in der UI der Roboter ein Schritt weiter oben selektiert werden	Bestanden
10	IO -> UI	Wenn der Selektionsbefehl nach unten ausgeführt wird, muss in der UI der Roboter ein Schritt weiter unten selektiert werden	Bestanden
11	ActuatorController -> UI	Wenn ein ActuatorController beendet wird oder kein Netzwerkzugriff mehr hat, soll der zugehörige Roboter aus der UI entfernt werden	Bestanden
12	ActuatorController -> UI	Wenn ein ActuatorController gestartet wird, oder er wieder einen Netzwerkverbindung bekommt und dadaurch ein Roboter komplettiert wird, muss dieser in der UI angezeigt werden	Bestanden

Tabelle 9.2: End-to-End Testfälle

9.1.2 Abnahmetests

Die folgenden Abnahmetests dienen dem Nachweis, dass die in Abschnitt 1.1 definierten Qualitätsziele erreicht werden. Jeder Test prüft eine oder mehrere Anforderungen an Funktionalität, Zuverlässigkeit, Leistung oder Sicherheit.

Test-ID	**Testname**	**Beschreibung**	**Erwartetes Ergebnis / Erfolgskriterium**
AT-01	Heartbeat bei Verbindungsabbruch	Simuliere Verbindungsabbruch (Ethernet Kabel trennen). Der Heartbeat-Mechanismus muss korrekt auslösen.	Roboterarm stoppt unter 250ms
AT-02	Horizontale Transport Einzelroboter, Hindernis	Ein Roboterarm bewegt ein Objekt(bsp. kleines stück Pappe, was den Greifer nicht behidnert) horizontal von links nach rechts.	Objekt erreicht geplantes Ziel, keine Kollision mit Hindernis.
AT-03	Vertikale Transport Höhenunterschied (20cm)	Objekt wird von unten nach oben über 20cm transportiert und sicher abgelegt.	Objekt liegt stabil, kein Herunterfallen, exakte Position.
AT-04	Switch zwischen Roboterarmen	Zwischen zwei Robotern wird gewechselt, ohne dass der inaktive Roboter sich bewegt.	Der neue Roboter akzeptiert RPC-Aufrufe, der vorherige ist still. Kein ungewolltes Bewegen.
AT-05	Horizontale Transport mit zwei Robotern + Switch	Zwei Roboter führen gemeinsam eine horizontale Bewegung aus (wie AT-02), aber mit einem Mid-Weg-Switch von Roboter A auf B.	Objekt bleibt stabil, Bewegung ohne Unterbrechung, reibungsloser Übergang, keine doppelte Bewegung.
AT-06	Heartbeat-Abbruch beim Roboterwechsel	Beim Wechsel von Roboter A auf B wird gezielt der Heartbeat von A abgebrochen.	Keinerlei störung, Bewegung auf Roboter B sind ohne Probleme weiter ausführbar
AT-07	Transport unter Netzwerklast	Das System wird während der Bewegung mit zusätzlichem Netzwerkverkehr belastet (z.B. durch künstliche UDP-Flut oder parallele Netzwerkstreams).	Bewegung wird trotz hoher Netzwerklast korrekt ausgeführt, keine Verzögerung, kein Paketverlust mit Auswirkung auf die Bewegung.

Tabelle 9.3: Abnahmetests zur Absicherung der Qualitätsanforderungen

10 Risiken

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

10.1 Ziel des chapters

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

10.2 Technische Risiken

Fehlende Erfahrungen, Komplexität und Externe Abhängigkeiten

10.3 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?

11 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 11.1: Glossar