Verteilte Systeme

VS Praktikum SoSe 2025

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

18. Mai 2025

Inhaltsverzeichnis

1 Einführung und Ziele

Diese Dokumentation beschreibt Entwurf und Kontext einer Middleware, die im Rahmen des Praktikums „Verteilte Systeme SoSe 2025“ entwickelt wird. Ziel des Praktikums ist der Entwurf und die Implementierung eines verteilten Steuerungssystems für autonome Roboterarme. Die Middleware bildet dabei eine zentrale Komponente, die die Komplexität der Verteilung für die Anwendungsebene abstrahieren soll.

1.1 Aufgabenstellung

Die Hauptaufgabe der zu entwickelnden Middleware besteht darin, die Kommunikation und Koordination zwischen den verteilten Komponenten des Steuerungssystems zu ermöglichen. Sie soll eine Abstraktionsschicht zwischen der eigentlichen Anwendungslogik (Roboterarmsteuerung) und den darunterliegenden Betriebssystem- und Netzwerkdiensten bilden. Gemäß den Prinzipien verteilter Systeme nach Tanenbaum & van Steen, die auch im zugrundeliegenden Skript referenziert werden, verfolgt die Middleware das Ziel, die Verteilung weitestgehend zu verbergen (Transparenz). Zu den Kernfunktionen, die eine Middleware typischerweise bereitstellt und die hier adressiert werden sollen, gehören:

- Bereitstellung der Kommunikation mittels standardisierter Protokolle. Dies wird (asynchrone) Kommunikation-Pattern umfassen, wie z.B. Remote Procedure Calls (RPC) oder Nachrichtenübertragung (Message-Oriented Middleware, MOM).
- Unterstützung zur Konvertierung von Datenformaten zwischen Systemen.
- Bereitstellung von Protokollen zur Namensauflösung (Service Discovery), um Ressourcen einfach zu identifizieren und zu referenzieren.
- Mechanismen der Skalierbarkeit, wie z.B. Replikation

Die Middleware dient als Vermittler, der zwischen der Applikation und der Runtime/OS Schicht Daten- und Informationsaustausch ermöglicht.

Zusammengefasst soll die Middleware Steuerung der einzelnen Nodes (Servos) eines jeden im System befindlichen Roboterarms ermöglichen. Dazu gehören die Erkennung jeder Node, die Auswahl eines zu steuernden Verbund an Nodes eines dazugehörigen Roboterarms, sowie die Ansteuerung selbst.

1.2 Qualitätsziele

Tabelle 1.1: Qualitätsziele der Software Engineering

Ziel	Beschreibung	Metrik
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 Nodes oder Komponenten sollen ohne Änderungen an der bestehenden Architektur integrierbar sein. Die Middleware muss Mechanismen (z.B. Replikation, Caching) bieten, um mit zunehmender Anzahl von Teilnehmern oder der Datenmenge zurechtzukommen.	Es können beliebig viele Roboter hinzugefügt und entfernt werden (0 - 253)
Sicherheit (Safety)	Es bewegt sich immer genau ein Node. Dieser darf sich nur Bewegen solange dieser Node erreichbar ist. Sollte das System nicht wie gewünscht reagieren, wird ein sicherer Zustand erreicht.	Reaktionszeit max. 250 ms, bis Node 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
Ressourcenteilung	Mehrere Nodes eines Verbunds (Roboterarm) können über einem Netzwerk von einem ITS-Board gesteuert werden	Alle dem Netzwerk hinzugefügten Nodes können sich registrieren und dessen Verbund beliebig angesteuert werden

Ziel	Beschreibung	Metrik
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 Nodesteuerung ist für den Benutzer nicht erkennbar	Nach aussen werden die genutzten Kommunikationsprotokolle verschleiert und durch einen Namen ersetzt. Der Name zeigt einen Verbund an.
Lokalitäts-Transparenz	Die Netzwerk- und Softwarestruktur ist nach außen unsichtbar	Das Interface nach außen ist eineiheitlich und verschleiert die Implementierung

1.3 Stakeholder

Tabelle 1.2: Interessen der Stakeholder

Stakeholder	**Interesse**
Nutzer	• Indirekte Stakeholder, deren Erfahrung stark von der Leistung und Zuverlässigkeit des Gesamtsystems, einschließlich der Middleware, abhängt.
Betreiber	• Portabilität: Das System kann auf verschiedenen Plattformen betrieben werden. • Zuverlässigkeit: Die Middleware kann über den gesamten benötigten Zeitraum ohne Ausfälle genutzt werden
Entwicklerteam Middleware	• Wartbarkeit • Portabilität: Das System kann auf verschiedenen Plattformen betrieben werden (z.B Testen) • Austauschbarkeit: Softwaremodule können ohne großen Aufwand ersetzt werden
Entwicklerteam Applikation	• Middleware bietet vollständige Funktionalität • Middleware-Schnittstellen sind vollständig beschrieben • Zuverlässigkeit und Reaktionszeit der von der Middleware bereitgestellten Kommunikationsdienste.
Professor	• Zugang zu allen Arbeitsmitteln zwecks Bewertung und Kontrolle • Das Endprodukt besitzt alle geforderten Funktionalitäten

2 Randbedingungen

Dieses Kapitel beschreibt die Rahmenbedingungen, unter denen die Middleware entworfen und implementiert werden muss.

2.1 Technische Randbedingungen

TODO Zeichnung

- **Betriebsumgebung:** Die Middleware muss auf einer heterogenen Umgebung aus verschiedenen Hardware-Plattformen und Betriebssystemen laufen können. Im spezifischen Projektkontext umfasst dies mindestens ein ITS-Board (STM32F4) und mehrere Raspberry Pis.
- **Netzwerk:**
  - Die Kommunikation findet innerhalb eines begrenzten Netzwerks statt, im Projektkontext ein /24 Netzwerk. Die Middleware baut auf den grundlegenden Kommunikationsdiensten des Betriebssystems und Netzwerks auf.
- **Hardware:** Die Middleware muss auf der Hardware laufen, die für das Steuerungssystem verwendet wird: ITS-Board und Raspberry Pis. Dies impliziert möglicherweise Beschränkungen hinsichtlich Rechenleistung, Speicher und verfügbaren Bibliotheken im Vergleich zu leistungsstarken Servern.
- **Anbindung:** Die Middleware muss in der Lage sein, sich mit der Anwendungsschicht darüber und den System-/Netzwerkschichten darunter zu integrieren.

2.2 Organisatorische Randbedingungen

- **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

3.1 Fachlicher Kontext

3.1.1 Akteure und Rollen

Akteur / Rolle	**Beschreibung**
Applikation	Externes Steuerungssystem, das über eine Schnittstelle Befehle an die Middleware sendet, z. B. zur Auswahl oder Bewegung eines Roboterarms.
Node (z. B. Servo)	Ein einzelner Aktor, der physisch mit einem Roboterarm verbunden ist. Meldet sich eigenständig bei der Middleware an und gehört zu einer logischen Gruppe.
Node-Gruppe	Eine logische Einheit mehrerer Nodes, die gemeinsam einen vollständigen Roboterarm darstellen. Nur eine Gruppe kann gleichzeitig aktiv sein.

3.1.2 Fachliche Aufgaben (Use Cases)

ID	**Name**	**Beschreibung**
U1	Node registrieren	Eine Node registriert sich bei der Middleware mit einem eindeutigen Identifier und einer Funktionsbeschreibung. Sie wird anschließend einer Node-Gruppe (Roboterarm) zugeordnet.
U2	Node-Gruppe auswählen	Die Applikation wählt eine existierende Node-Gruppe aus, die als Ziel für Steuerungsbefehle verwendet werden soll.
U3	Node-Gruppe bewegen	Die Middleware empfängt Bewegungsbefehle von der Applikation und leitet diese an entsprechende Nodes der aktiven Node-Gruppe weiter.
U4	Verfügbarkeit prüfen	Die Middleware prüft, ob alle Nodes einer gewählten Gruppe erreichbar und einsatzbereit sind, bevor Steuerungsbefehle ausgeführt werden.
U5	Verfügbarkeit propagieren	Die Middleware muss der Applikation in regelmäßigen Abständen mitteilen welche Node-Gruppen fehlerfrei erreichbar sind.

3.1.3 Fachliche Randbedingungen

- Es können bis zu 253 Nodes gleichzeitig betrieben werden.
- Jede Node gehört genau einer Node-Gruppe an.
- Steuerungsbefehle dürfen nur an Gruppen gesendet werden, die als aktiv markiert sind.
- Die Middleware muss mit dem Ausfall oder der Nichterreichbarkeit einzelner Nodes robust umgehen können.
- Die Systemsicherheit hat höchste Priorität: Bewegungen fehlerhafter Gruppen müssen verhindert werden.

3.2 Technischer Kontext

Die Middleware positioniert sich technisch als Vermittlungsschicht zwischen der Anwendungsebene und den darunterliegenden Betriebssystem- und Kommunikationsdiensten. Sie ist über mehrere Maschinen verteilt, die über ein lokales /24 Netzwerk miteinander verbunden sind. Die Middleware nutzt Netzwerkprotokolle, insbesondere TCP und UDP, sowie Betriebssystemfunktionen, um eine zuverlässige Kommunikation zu gewährleisten. Gleichzeitig bietet sie den Anwendungen eine abstrahierte, einheitliche Schnittstelle, die die Heterogenität der zugrundeliegenden Systeme verbirgt. Die Middleware stellt darüber hinaus eigene Kommunikationsprotokolle bereit, die speziell auf die Anforderungen der verteilten Steuerung und Koordination der Roboterarme zugeschnitten sind. Hierzu zählen Mechanismen zur Nachrichtenvermittlung, Statusüberwachung und Fehlerbehandlung, um eine hohe Zuverlässigkeit und Fehlertoleranz sicherzustellen. Zusammenfassend erfüllt die Middleware folgende technische Aufgaben:

- Verbergen der Heterogenität der zugrundeliegenden Hardware und Betriebssysteme
- Bereitstellung einer einheitlichen Kommunikationsschnittstelle für die darüber liegenden Anwendungen
- Sicherstellung der zuverlässigen und performanten Kommunikation zwischen verteilten Komponenten
- Implementierung von Mechanismen zur Erkennung und Behandlung von Fehlerzuständen im Netzwerk und auf den Nodes

3.2.1 Funktionale Anforderungen an die Middleware

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

- **Registrierungsmanagement (U1):** Verwaltung und Verarbeitung von Node-Registrierungen mit eindeutigen Identifikatoren und Funktionsbeschreibungen. Persistenz der Node-Informationen und Zuordnung zu Node-Gruppen.
- **Gruppenverwaltung (U2):** Auswahl, Wechsel und Markierung aktiver Node-Gruppen zur Steuerung. Verwaltung des Gruppenstatus.

- **Steuerungsweiterleitung (U3):** Empfang von Steuerbefehlen von der Applikation und Weiterleitung dieser Befehle an die Nodes der aktiven Gruppe.
- **Verfügbarkeitsprüfung (U4):** Überwachung des Status und der Erreichbarkeit aller Nodes innerhalb einer Gruppe, inklusive Fehlererkennung und Meldung an die Applikation.
- **Verfügbare Nodegruppen propagieren (U5):** Mitteilung an Applikation, welche Node-Gruppen verfügbar und ansteuerbar sind. TODO: Entweder Schnittstelle anbieten und Applikation entscheiden lassen, oder Schnittstelle der Applikation nutzen!?
- **Fehlertoleranz und Sicherheit:** Mechanismen zur Behandlung von Kommunikationsausfällen, erneutes Senden von Nachrichten und Schutz vor Fehlfunktionen einzelner Nodes.

3.3 Externe Schnittstellen

Die Middleware besitzt zwei Schnittstellen:

- **Schnittstelle zur Anwendungsebene:** Bietet eine einheitliche API, über die Anwendungen (z. B. auf ITS-Board oder Raspberry Pis) verteilte Dienste nutzen, Nachrichten senden und empfangen können – unabhängig von Netzwerkdetails oder physikalischer Verteilung.
- **Schnittstelle zu System- und Netzwerkschichten:** Nutzt Betriebssystem- und Netzwerkdienste (TCP/IP, IPv4) zur Nachrichtenübermittlung und Ressourcenverwaltung. Diese Schnittstelle ist für die Anwendungen verborgen.

4 Lösungsstrategie

TODO: Weitere Funktionen (atomar))

Tabelle 4.1: Funktionsbeschreibungen

Funktion	Beschreibung	Vor-Nachbedingungen
void register(Identifier ident, enum functionality)	Schnittstelle für eine Node (Servo), um sich zu beim System zu registrieren. Identifier ist für jeden Servo einzigartig, functionality ist ein enum mit Aufgabenbereich an einem Roboterarm.	Jede Node (Servo) muss seinen eigenen Identifier und Funktionalität vor der Registrierung kennen.
Node createNewNode (Identifier ident, enum functionality)	Übersetzt externe Node zu internen Node Code	
void updateAvailable-Nodes(List<Node> nodes)	Liste mit allen verfügbaren Nodes, um diese der Applikation mitzuteilen.	
void heartbeat(Identifier ident)	Check ob Node verfügbar	Vor: Node ist registriert; Nach: Timeout-Zähler zurückgesetzt
void unregister(Identifier ident)	Entfernt eine Node aus dem System, zb wenn nicht mehr erreichbar anhand seines Identifiers	Vor: Node ist registriert; Nach: Node ist aus Liste entfernt
void choose(NodeGrp nodeGrpIdent)	Schnittstelle für Applikation, um Node Gruppe zu wechseln	
void markActive(NodeGrp nodeGrpIdent)	Node Gruppe wird als aktiv makiert	Vor: Applikation hat wechsel beauftragt; Nach: Node Gruppe wird als aktiv markiert, alle Steuerbefehle beziehen sich auf diese Gruppe

Funktion	Beschreibung	Vor-Nachbedingungen
void move(Direction dir)	Schnittstelle für Applikation um Node zu bewegen	Vor: Node Gruppe ist ausgewählt und verfügbar; Nach: Bewegung wird ausgeführt
Node readFromNode-List(Direction dir)	Gibt aktiven Node zurück mit passender Funktion	
void isActive(Direction dir)	Schnittstelle für Applikation um Node zu bewegen	Vor: Node Gruppe ist ausgewählt und verfügbar; Nach: Bewegung wird ausgeführt
void moveNode(Identifier ident)	ruft entsprechende Node Schnittstelle auf	

TODO: Identifier muss ausgearbeitet werden (Allgemein Service Discovery und Speicherung) sowie Rückgabetyp Node

Tabelle 4.2: Lösungstrategien Ziele

Ziel	Strategie
Kommunikation	Asynchrones Remote Procedure Call (RPC) als Kommunikationsmechanismus. Registrierungsvorgänge werden persistent gespeichert, während Steuerbefehle transient und zeitkritisch behandelt werden.
Service-Discovery	Flacher Namensraum mit Gruppierung der Nodes. Der Name besteht aus eindeutiger Node-ID, Funktion und Zugehörigkeit zum Roboterarm-Verbund.
Namensauflösung	IPv4 Adresse und dazugehöriger Node wird in einen Namen übersetzt, der die Node zu einem Verbund zugehörig macht.
Fehlerbehandlung	

5 Bausteinsicht

6 Laufzeitsicht

6.1 Szenario I

Anschalten der Roboter-Software

6.2 Szenario II

Steuerbefehl über GUI-Software an einen ausgewählten, verfügbaren Roboter

6.3 Szenario III

Fehlerfall: Verbindung zu Roboter fällt aus

7 Verteilungssicht

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

Tabelle 9.1: Zentrale Entwurfsentscheidungen

10 Qualitätsszenarien

11 Risiken