

Verteilte Systeme

VS Praktikum SoSe 2025

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

26. Mai 2025

Inhaltsverzeichnis

1	Einführung und Ziele	4
1.1	Aufgabenstellung	4
1.2	Qualitätsziele	5
1.3	Stakeholder	6
2	Randbedingungen	7
2.1	Technische Randbedingungen	7
2.2	Organisatorische Randbedingungen	7
3	Kontextabgrenzung	8
3.1	Fachlicher Kontext	8
3.1.1	Akteure und Rollen	8
3.1.2	Fachliche Aufgaben (Use Cases)	8
3.1.3	Fachliche Randbedingungen	8
3.2	Technischer Kontext	9
3.2.1	Technische Anforderungen an die Middleware	9
3.3	Externe Schnittstellen	10
4	Lösungsstrategie	11
5	Bausteinsicht	14
6	Laufzeitsicht	15
6.1	Szenario I	15
6.2	Szenario II	16
6.3	Szenario III	17
7	Verteilungssicht	18
8	Konzepte	19
8.1	Offenheit	19
8.2	Verteilungstranzparenzen	19
8.3	Kohärenz	19
8.4	Sicherheit (Safety)	19
8.5	Bedienoberfläche	19
8.6	Ablaufsteuerung	19
8.7	Ausnahme- und Fehlerbehandlung	20
8.8	Kommunikation	20
8.9	Konfiguration	20
8.10	Logging, Protokollierung	20
8.11	Plausibilisierung und Validierung	20

8.12 Sessionbehandlung	20
8.13 Skalierung	20
8.14 Verteilung	20
9 Entwurfsentscheidungen	21
10 Qualitätsszenarien	22
11 Risiken	23

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 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:

- Zuständig für die Verteilungstransparenz
- 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, und die Weiterleitung der Steuerbefehle.

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.	Es können beliebig viele Nodes hinzugefügt und entfernt werden (0 - 253)
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	Alle dem Netzwerk hinzugefügten Nodes können sich registrieren und anschließend miteinander kommunizieren	Das Namensregister ist im gesamten Netzwerk verfügbar
Offenheit	Zugänglichkeit Interoperabilität Portabilität	Einzelne Softwarekomponenten können, ausgetauscht oder portiert werden, um z.B. Standards bzgl. Kommunikation und Datenstruktur zu tauschen oder um die Plattform zu wechseln. Die Funktionalität bleibt gleich
Zugriffstransparenz	Die Umsetzung Kommunikation zwischen den Nodes ist für den Benutzer nicht erkennbar	Die Applikation kommuniziert über Namen
Lokalitäts-Transparenz	Die Netzwerk- und Softwarestruktur ist nach außen unsichtbar	Das Interface nach außen ist einheitlich und verschleiert die Implementierung

1.3 Stakeholder

Tabelle 1.2: Interessen der Stakeholder

Stakeholder	Interesse
Betreiber	<ul style="list-style-type: none">• Portabilität: Das System kann auf verschiedenen Plattformen betrieben werden.• Zuverlässigkeit: Die Middleware kann über den gesamten benötigten Zeitraum ohne Ausfälle genutzt werden
Entwicklerteam Middleware	<ul style="list-style-type: none">• Wartbarkeit• Portabilität: Das System kann auf verschiedenen Plattformen betrieben werden (z.B Testen)• Austauschbarkeit: Softwaremodule können ohne großen Aufwand ersetzt werden
Entwicklerteam Applikation	<ul style="list-style-type: none">• Middleware bietet vollständige Funktionalität• Middleware-Schnittstellen sind vollständig beschrieben.• Zuverlässigkeit und Reaktionszeit der von der Middleware bereitgestellten Kommunikationsdienste.
Professor	<ul style="list-style-type: none">• 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.
- **Anbindung:** Die Middleware muss in der Lage sein, mit der Anwendungsschicht und den System-/Netzwerkschichten zu interagieren.

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 Roboterservos.
Node (z. B. Servo(s))	Ein einzelner Akteur. Meldet sich eigenständig bei der Middleware an.

3.1.2 Fachliche Aufgaben (Use Cases)

ID	Name	Beschreibung
U1	Node registrieren	Eine Node registriert sich bei der Middleware mit einem eindeutigen Identifier. Dem Identifier wird dessen IPv4 Adresse zugeordnet.
U2	Informationen weiterleiten	Die Middleware leitet RPCs von der Applikation an die entsprechenden Nodes weiter.
U3	Verfügbarkeit prüfen	Es wird geprüft, welche Nodes erreichbar sind.
U4	Verfügbarkeit propagieren	Die Middleware muss der Applikation mitteilen welche Nodes fehlerfrei erreichbar sind.
U5	Marshalling	Die Middleware muss dafür sorgen, dass die Datenstrukturen und Parameter der Funktionsaufrufe korrekt in ein übertragbares Format (Marshalling) umgewandelt und am Zielsystem wieder entpackt (Unmarshalling) werden.

3.1.3 Fachliche Randbedingungen

- Es können bis zu 253 Nodes mit unterschiedlichen IPv4-Adressen gleichzeitig betrieben werden.
- Jede Node ghat einen eindeutigen Namen.
- Die Middleware muss mit dem Ausfall oder der Nichterreichbarkeit einzelner Nodes robust umgehen können.

- Die Systemsicherheit hat höchste Priorität: Weiterleitungen an ungültige Nodes müssen verhindert werden. Es muss aber eine Fehlermeldung an den Administrator erfolgen.

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
- Implementierung einer Namensauflösung

3.2.1 Technische 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.
- **Weiterleitung (U2):** Marshalling der zu Versendeten Informationen, korrektes Verteilen an den entsprechenden Empfänger mit abschließenden Unmarshalling.
- **Verfügbarkeit prüfen (U3):** Regelmäßiges Abfragen der einzelnen Nodes auf Verfügbarkeit durch einen Heartbeat-Mechanismus **TODO: Prüfen, ob Watchdog tatsächlich Teil der Middleware, da Applikation im besten Falle erreichbar sein sollte. Watchdog sollte besser ausserhalb von Middleware und Applikation liegen. Aber auch mit und über Middleware kommunizieren.** Fehlerhafte Nodes oder Nichterreichbarkeit sollte der Applikation mitgeteilt werden.

- **Verfügbarkeit propagieren (U4):** Die interne Namensliste aller verfügbaren Nodes sollte der Applikation auf anfrage mitgeteilt werden. **TODO: Entweder Schnittstelle anbieten und Applikation entscheiden lassen, oder Schnittstelle der Applikation nutzen!?**
- **Marshalling (U5):** Einführung einer IDL Definition
- **Fehlertoleranz und Sicherheit:** Mechanismen zur Behandlung von Kommunikationsausfällen, erneutes Senden von Nachrichten und Schutz vor Fehlfunktionen einzelner Nodes. Mitteilung an die Applikation zur Fehlertransparenz.

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, sowie sich als Teilnehmer registrieren 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
<code>void register(char* name, char* socket)</code>	Schnittstelle zur Applikation für eine Node (Servo), um sich zu beim System zu registrieren. Identifier ist für jeden Servo einzigartig, socket ist die IPv4 Adresse plus Port der Applikation.	Jede Node (Servo) muss seinen eigenen Identifier und Socket vor der Registrierung kennen.
<code>int registerNewNode(String name, String socket)</code>	Registriert neuen Empfänger in der Namensauflösung	Name ist noch nicht vorhanden; return 0 oder -1
<code>void invoke(String name, String functionName, String[] argTypes, String[] args)</code>	Schnittstelle zur Applikation um RPCs an entfernte Ziele zu senden.	IDL der Middleware kann den Typen Marshallable der Applikation serialisieren
<code>int marshal(String functionName, String[] argTypes, String[] args, char* buffer, size bufSize)</code>	Führt Marshalling durch	Marshallable und buffer darf nicht NULL sein. Nach: return 0 oder -1
<code>int unmarshal(const char* buffer, String functionName, String[] argTypes, String[] args)</code>	Führt Unmarshalling durch	Marshallable und buffer darf nicht NULL sein. Nach: return 0 oder -1
<code>void updateAvailableNodes(List<Node> nodes)</code> TODO: Wie wird die Liste rausgegeben?	Liste mit allen verfügbaren Nodes, um diese der Applikation mitzuteilen.	Es existiert eine Liste die auch von der Applikation gelesen werden kann (Liste muss angepasst werden an die)

Funktion	Beschreibung	Vor-Nachbedingungen
<code>void forwardCall(const char* target, String functionName, String[] argTypes, String[] args)</code>	Leitet den ankommenden RPC an das Target weiter	Das Target muss existieren.
<code>char* resolveName(const String* target)</code>	Löst den eindeutigen Namen zu einem Socket (IPv4:Port) auf.	Vorbedingung: target ungleich NULL; Rückgabe: String mit Socket-Adresse oder NULL bei Fehler.
TODO: CHECK wegen Watchdog <code>void heartbeat(Identifier ident)</code>	Check ob Node verfügbar	Vor: Node ist registriert; Nach: Timeout-Zähler zurückgesetzt
<code>void unregister(Identifier ident)</code>	Entfernt eine Node aus dem System, zb wenn nicht mehr erreichbar anhand seines Identifiers	Vor: Node ist registriert; Nach: Node ist aus Liste entfernt

Tabelle 4.2: Lösungsstrategien Ziele

Ziel	Strategie
Kommunikation	Asynchrones Remote Procedure Call (RPC) als Kommunikationsmechanismus. Registrierungsvorgänge werden persistent gespeichert, während Steuerbefehle transient und zeitkritisch behandelt werden. Kommunikation zwischen entfernten Zielen mithilfe von UDP/TCP nach vorherigem Marshalling.
Namensauflösung	Flacher Namensraum. Der Name besteht aus eindeutiger Node-ID, und IPv4 Socketinformationen.
Fehlerbehandlung	Logging, Timeouts und Wiederholungen von Weiterleitungen. Erkannte Fehler werden der Applikation mitgeteilt TODO: Funktion bestimmen
Marshalling	Verwendung eines einheitlichen Marshallable-Typs und einer IDL zur Beschreibung und Serialisierung komplexer Funktionsaufrufe.
Transparenz	Middleware abstrahiert Verbindungsdetails, Adressierung und Datenformate für die Applikation vollständig (Standort-, Zugriffstransparenz).
Sicherheit / Safety	TODO: Watchdog

Beispiel-Struktur des Marshallable-Typs

Der Typ `Marshallable` dient als generischer Container für entfernte Funktionsaufrufe. Er enthält den Funktionsnamen und eine Liste typisierter Parameter.

Listing 4.1: Definition des `Marshallable`-Typs

```
1  typedef enum {
2      TYPE_INT,
3      TYPE_FLOAT,
4      TYPE_STRING
5  } ParamType;
6
7  typedef struct {
8      ParamType type;
9      union {
10         int i;
11         float f;
12         char* s;
13     } data;
14  } Param;
15
16  #define MAX_PARAMS 8
17
18  typedef struct {
19      char function_name[64];
20      int param_count;
21      Param params[MAX_PARAMS];
22  } Marshallable;
```

Die Felder des Typs werden in folgender Tabelle erläutert:

Tabelle 4.3: Felder der Struktur `Marshallable`

Feld	Typ	Beschreibung
<code>function_name</code>	<code>char[64]</code>	Name der aufzurufenden Funktion.
<code>param_count</code>	<code>int</code>	Anzahl der übergebenen Parameter.
<code>params</code>	<code>Param[]</code>	Liste der Parameter inklusive Typinformationen.

Die Middleware kennt die Struktur von `Marshallable` und kann sie serialisieren. Die genauen Typinformationen der Parameter (z.B. Reihenfolge und Datentypen) sind über eine IDL definiert, die separat gepflegt wird.

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 Verteilungstransparenzen

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 verfügbar" erkennbar. 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 Benutzeroberfläche, die Workflow- oder Geschäftsprozesssteuerung 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