Verteilte Systeme

VS Praktikum SoSe 2025

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

25. Juni 2025

Inhaltsverzeichnis

1 Einführung und Ziele **4**
1.1 Aufgabenstellung 4
1.2 Qualitätsziele 5
1.3 Stakeholder 7

2 Randbedingungen **8**
2.1 Technische Randbedingungen 8
2.2 Organisatorische Randbedingungen 8

3 Kontextabgrenzung **9**
3.1 Fachlicher Kontext 9
3.1.1 Akteure und Rollen 9
3.1.2 Fachliche Aufgaben (Use Cases) 10
3.1.3 Fachliche Randbedingungen 10
3.2 Technischer Kontext 11
3.2.1 Technische Anforderungen an die Middleware 11
3.3 Externe Schnittstellen 13

4 Lösungsstrategie **14**

5 Bausteinsicht **17**

6 Laufzeitsicht **19**
6.1 Szenario I 19

7 Verteilungssicht **20**

8 Konzepte **21**
8.1 Offenheit 21
8.2 Verteilungstranzparenzen 21
8.3 Kohärenz 21
8.4 Sicherheit (Safety) 21
8.5 Bedienoberfläche 21
8.6 Ablaufsteuerung 21
8.7 Ausnahme- und Fehlerbehandlung 22
8.8 Kommunikation 22
8.9 Konfiguration 22
8.10 Logging, Protokollierung 22
8.11 Plausibilisierung und Validierung 22
8.12 Sessionbehandlung 22
8.13 Skalierung 22

8.14 Verteilung 22

9 Entwurfsentscheidungen 23

10 Qualitätsszenarien 24

11 Risiken 25

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).
- 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
- Security Mechanismus, dass nicht akzeptierte Nachrichten verworfen werden. Feedback muss stattfinden.

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).Definiert aufgetretene Fehler werden kommuniziert.
Skalierbarkeit	Zusätzliche Nodes oder Komponenten sollen ohne Änderungen an der bestehenden Architektur integrierbar sein.	Es können bis zu 253 Nodes hinzugefügt und entfernt werden
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 zb 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. Die eigentliche Kommunikation bleibt versteckt.

Ziel	Beschreibung	Metrik
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**
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

- **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.
  - Es wird mit IPv4 kommuniziert.
- **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.
- **Sprachen:** Die Middleware muss in den Sprachen C und JAVA umgesetzt werden.

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**
Server (Applikation)	Ein einzelner Teilnehmer. Meldet sich eigenständig bei der Middleware mit seinen Diensten an.
Client (Applikation)	Ein einzelner Teilnehmer. Kann die Dienste der anderen Teilnehmern aufrufen.
Health-Observable (Applikation)	Teilnehmer, der Lebenszeichen schickt
Health-Observer (Applikation)	Teilnehmer, der bei ausbleibenden Lebenszeichen benachrichtigt wird

3.1.2 Fachliche Aufgaben (Use Cases)

ID	Name	Beschreibung
U1	Informationen weiterleiten	Die Middleware leitet RPCs von der Applikation an die entsprechenden Nodes weiter.
U2	Node registrieren	Ein Server registriert seine Dienste bei der Middleware mit einem eindeutigen Identifier. Dem Identifier wird dessen Socket-Adresse zugeordnet.
U3	Namen auflösen	Die Middleware ermittelt, welche Dienste bei welcher Socket-Adresse zu finden sind.
U4	Schnittstellen zur Applikation	Dienste werden der Applikation als Client-Stubs bereitgestellt. Die Applikation ruft die entfernten Dienste wie lokale Dienste auf. Interner Aufrufe werden vor der Applikation versteckt.
U5	Marshalling	Die Middleware definiert eine IDL. Diese muss dafür sorgen, dass die Datenstrukturen und Parameter der Funktionsaufrufe via RPC korrekt in ein übertragbares Format (Marshalling) umgewandelt und am Zielsystem wieder entpackt (Unmarshalling) werden.
U6	Transport	Die RPCs werden via UDP über die Netzwerkschicht transportiert. Das ist durch die Echtzeitanforderung bedingt. Kommunikation findet asynchron statt. Synchronizität wird durch mehrere asynchrone RPCs umgesetzt.
U7	Sicherheit	Versendete RPCs haben (zwischen Client und Server) Echtzeitanforderungen.
U8	Fehlertransparenz	Fehlerhafte Übertragung eines RPCs müssen ggf. der Applikation mitgeteilt werden.
U9	Watchdog	Für überwachte Health-Observables werden Watchdogs angelegt. Diese überwachen die Verfügbarkeit der Teilnehmer, wenn Lebenszeichen ausbleiben werden Health-Observer benachrichtigt.

3.1.3 Fachliche Randbedingungen

- Es können bis zu 253 Nodes mit unterschiedlichen IPv4-Adressen gleichzeitig betrieben werden.
- Es wird IPv4 gefordert, daher muss das TCP/IP-Modell für den Transport genutzt 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 Teilnehmer verteilt, die über ein lokales /24 Netzwerk miteinander verbunden sind. Die Middleware nutzt Netzwerkprotokolle, insbesondere 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.

3.2.1 Technische Anforderungen an die Middleware

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

- **Informationen weiterleiten U1:**
  - Schnittstelle zur Applikation zum entfernten Funktionsaufruf und dessen Parameter.
  - Einheitliche Übersetzung und Serialisierung.
  - Namensauflösung, ggf. aus Cache.
  - Transport der Nachricht über Netzwerk.
  - Deserialisierung der empfangenen Aufrufs.
  - Schnittstelle zur Applikation, um lokalen Funktionsaufruf auszuführen.
- **Node registrieren U2:**
  - Schnittstelle zur Applikation mit Gruppen Name, Funktionsname und seiner Socket-Adresse via RPC
  - Einheitliche Übersetzung und Serialisierung
  - Senden an bekannten Namensserver
  - Deserialisierung der empfangenen Aufrufs am Namensserver
  - Eintragen der Informationen
- **Namen auflösen U3:**
  - Senden der Anfrage mit Parameter (Gruppen Name, Funktionsname, eigener Gruppenname, Funktionsname der Antwort) via RPC
  - Einheitliche Übersetzung und Serialisierung
  - Senden an bekannten Namensserver
  - Warten auf Antwort
  - Deserialisierung der empfangenen Aufrufs am Namensserver
  - Auflösen der Socket-Adresse zum Funktionsnamen

  - Aufruf der Antwortsfunktion mit Socketadresse als Parameter
  - Einheitliche Übersetzung und Serialisierung
  - Senden an vorherigen Sender
  - Deserialisierung der empfangenen Aufrufs
  - Einsetzen der Socketadresse im Cache
- **Schnittstellen zur Applikation U4:**
  -
- **Marshalling U5:**
  -
- **Transport U6:**
  -
- **Sicherheit U7:**
  -
- **Fehlertransparenz U8:**
  -
- **Watchdog U9:**
  -
- **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.
- **Namen auflösen (U3):** Einrichtung einer hierarischen Namensauflösung, die in der Lage ist, die Adresse eines Dienstes (einer Funktion) auf einem Node zu verschicken. Diese werden in Gruppen aus Nodes sortiert.
- **Marshalling (U4):** Einführung einer IDL Definition, sodass die Sprachne C und JAVA unterstützt werden.
- **Fehlertoleranz und Sicherheit:** Mechanismen zur Behandlung von Kommunikationsausfällen oder Unterbrechungen. Mitteilung an die Applikation zur Fehlertransparenz.

3.3 Externe Schnittstellen

Die Middleware besitzt folgende 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
void register(char* name, char* socket)	**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.
int registerNewNode(String name, String socket)	Registriert neuen Empfänger in der Namensauflösung	Name ist noch nicht vorhanden; return 0 oder -1
void invoke(String name, String functionName, String[] argTypes, String[] args)	**Schnittstelle zur Applikation** um RPCs an entfernte Ziele zu senden.	IDL der Middleware kann den Typen Marshallable der Applikation serialisieren
int marshal(String functionName, String[] argTypes, String[] args, char* buffer, size bufSize)	Führt Marshalling durch	Marshallable und buffer darf nicht NULL sein. Nach: return 0 oder -1
int unmarshal(const char* buffer, String functionName, String[] argTypes, String[] args)	Führt Unmarshalling durch	Marshallable und buffer darf nicht NULL sein. Nach: return 0 oder -1
void updateAvailableNodes(List<Node> nodes) 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
void forwardCall(const char* target, String functionName, String[] argTypes, String[] args)	Leitet den ankommenden RPC an das Target weiter	Das Target muss existieren.
char* resolveName(const String* target)	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 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

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. 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

```
typedef enum {
    TYPE_INT,
    TYPE_FLOAT,
    TYPE_STRING
} ParamType;

typedef struct {
    ParamType type;
    union {
        int i;
        float f;
        char* s;
    } data;
} Param;

#define MAX_PARAMS 8

typedef struct {
    char function_name[64];
    int param_count;
    Param params[MAX_PARAMS];
} Marshallable;
```

Die Felder des Typs werden in folgender Tabelle erläutert:

Tabelle 4.3: Felder der Struktur `Marshallable`

Feld	**Typ**	**Beschreibung**
`function_name`	`char[64]`	Name der aufzurufenden Funktion.
`param_count`	`int`	Anzahl der übergebenen Parameter.
`params`	`Param[]`	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

Beschreibung

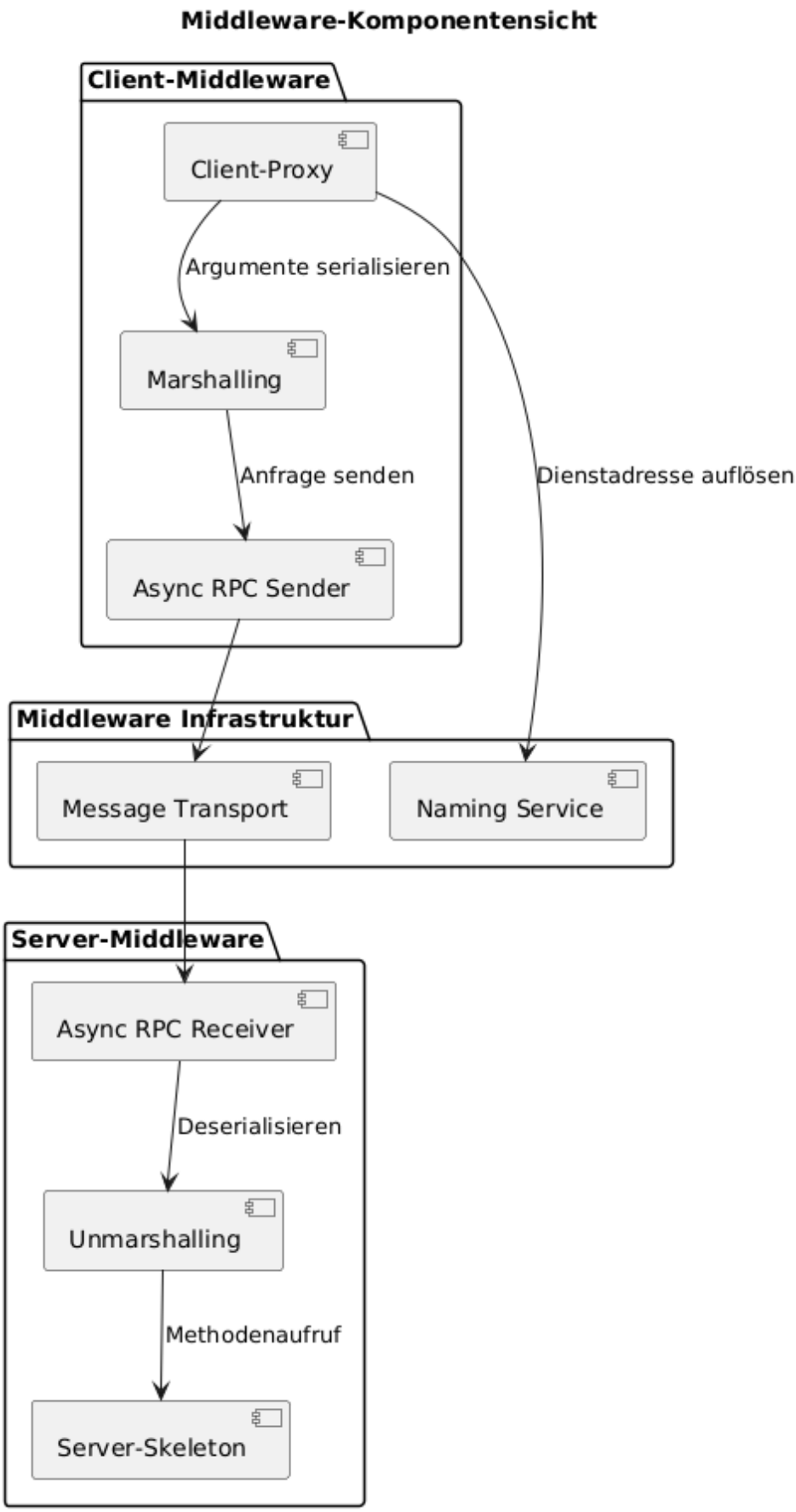

Abbildung 5.1: Komponenten der Middleware

6 Laufzeitsicht

6.1 Szenario I

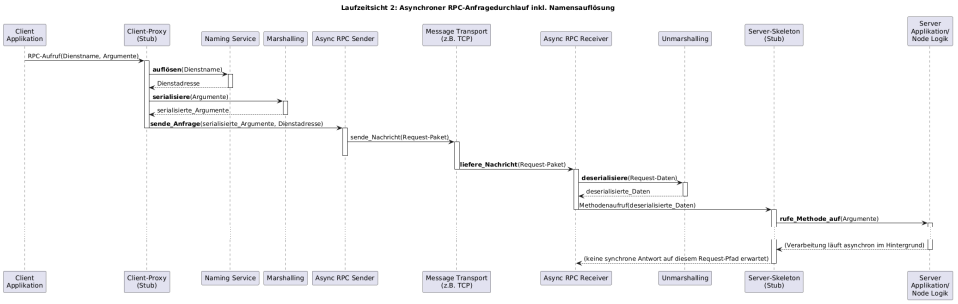

Abbildung 6.1: Asynchroner RPC-Anfragedurchlauf inkl. Namensauflösung

Beschreibung

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