Analysedokument

(Veranstaltung Modellierung II, SoSe 2016)

Projekt:	Roboterbasiertes Personentransportsystem
Auftraggeber:	Prof. Holger Giese Hasso-Platter-Institut Prof.-Dr.-Helmert-Str. 2–3 14482 Potsdam
Auftragnehmer:	Modellierung II – Projektgruppe 5

Verantwortlichkeit	Name, Vorname	Datum
Ansprechpartner	Bischoff, Sebastian	23.05.2016
Bearbeitender	Sauder, Jonathan	23.05.2016
Bearbeitender	Lüpke, Fabian	23.05.2016
Bearbeitender	Hering, Jonas	23.05.2016
Bearbeitender	Braun, Jakob	23.05.2016
Bearbeitender	Cremerius, Jonas	23.05.2016
Bearbeitender	Wischner, Jakob	23.05.2016
Bearbeitender	Schwenkert, Daniel	23.05.2016
Bearbeitender	Jäkel, Dominik	23.05.2016
Bearbeitender	König, Bastian	23.05.2016

Inhaltsverzeichnis

1 Abstrakte Architektur

In diesem Dokument soll ein System mit autonom agierenden Robotern (*Robots*) entwickelt werden, welche von einem *Server* zu bestimmten Zielen (*Destinations*) in Form von Positionen (*Positions*) geschickt werden.

Dieser Entwurf basiert auf der in der Analyse erarbeiteten Spezifikation des Systems. Abbildung 1 zeigt das abstrakte Komponentendiagramm.

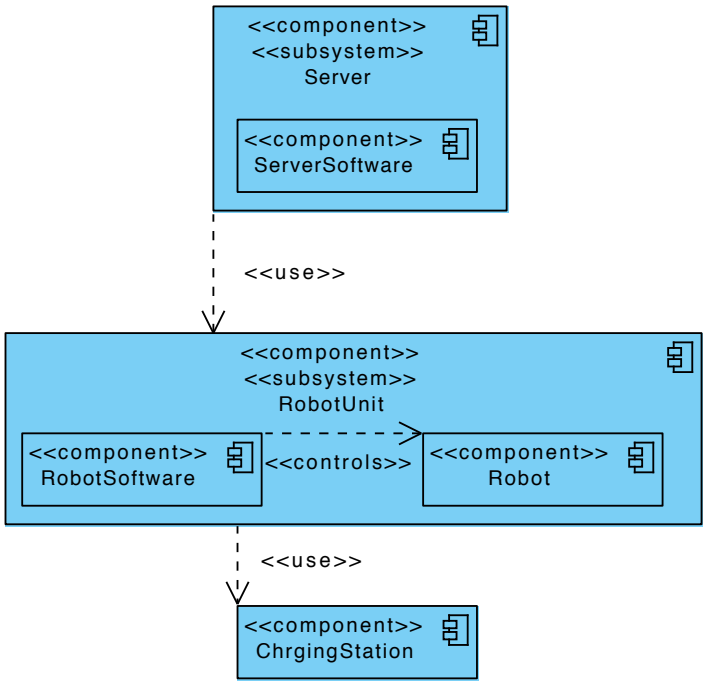

Abbildung 1: Abstraktes Komponentendiagramm

Da eine Komponente mit dem Namen *Robot* bereits vorhanden ist, nennen wir die *Robot*-Hauptkomponente *RobotUnit*. Dieser Name verdeutlicht, dass jeder physische Roboter im System durch diese Hauptkomponente repräsentiert wird.

In diesem Kapitel wird die im Rahmen der Analyse ermittelte Einteilung des Systems in Komponenten strukturiert dargestellt. Hierbei soll auch die Interaktion der Komponenten untereinander verdeutlicht werden.

1.1 Server

Der *Server* verwaltet die *Destinations* für die *RobotUnits* und behält einen ständigen Überblick über die Positionen der *RobotUnits*. Er ist für die effiziente Allokation der Aufträge für die *RobotUnits* zuständig.

1.1.1 ServerSoftware

Die Komponente *ServerSoftware* ist die Verwaltungslogik des *Servers*. Sie greift auf die serverseitigen Subsysteme zu und stellt die zentrale Anlaufstelle für alle *RobotUnits* dar.

Bei einer Anfrage ermittelt sie die passende *RobotUnit* und sendet ihr die Position des Ziels.

1.2 RobotUnit

Die Komponente *RobotUnit* sublimiert die *RobotSoftware* und -*Hardware* (Komponente *Robot*) als eine Oberkomponente. Sie vereint alle Hard- und Softwareinterfaces dieser Komponenten und leitet die ein- und ausgehenden Nachrichten der *RobotSoftware* an den *Server* weiter. Sie dient somit als übergeordnete Schnittstelle für die Kommunikation.

1.2.1 RobotSoftware

Die *RobotSoftware* steuert den *Robot* an und verwertet seine Sensordaten. Dazu gehört unter anderem die Fahrlogik und die Verwaltung der *Battery*, damit angenommene Aufträge auch komplett ausgeführt werden können und die *RobotUnit* rechtzeitig die *ChargingStation* erreicht.

1.2.2 Robot

Die Komponente *Robot* steht für alle hardwaretechnischen Spezifikationen des *Robots*. Dazu gehören die Fahreinheit und die interne Sensorik so wie die *Battery*.

2 Interaktion der Komponenten

Auf Basis der Use Cases aus dem Analysedokument wird in diesem Kapitel die Interaktion der einzelnen Komponenten aus Kapitel 1 betrachtet. Dabei liegt der Fokus vor allem auf der Interaktion zwischen dem *Server* und der *RobotUnit*. Die Use Cases innerhalb der Komponenten werden in Kapitel 8 näher ausgeführt.

Interaktion bei Ausführung von 4: Choose Robot

Die Auswahl eines *Robots* läuft wie in Abbildung 2 beschrieben folgendermaßen ab: Bei einer neu eingehenden *Destination* sendet der *Server* mit `getTaskRating(task)` Anfragen an alle zur Verfügung stehenden *RobotUnits* (im Diagramm sind beispielhaft zwei angeführt, der Aufruf findet asynchron statt). Die *RobotUnit* führt dann *Read Sensor* (Use Case 2) aus. Dabei sammelt sie alle notwendigen Informationen von ihren Hardwareschnittstellen, wie Ladestand und Nähe zum Ziel, die der *Server* benötigt, um den bestmöglichen *Robot* für das Ziel auszuwählen. Der *Robot* wartet auf das Zusammentragen der Daten, also das Abschließen des internen Loop-Prozesses, bis er eine Nachricht mit den Informationen an den *Server* zurücksendet; erst dann kann der *Server* auf Grund der übermittelten Daten beurteilen, welcher *Robot* am besten dazu geeignet ist, die *Destination* zu erreichen und entsendet ihn zu den Zielkoordinaten (*assign Task* – Use Case 5). Bei dem darauffolgenden Use Cases *DriveToDestination* ist der *Server* hingegen nicht beteiligt. Bei dem Use Case *Charging* kommt es ausschließlich zur Wechselwirkung mit der *ChargingStation*-Komponente.

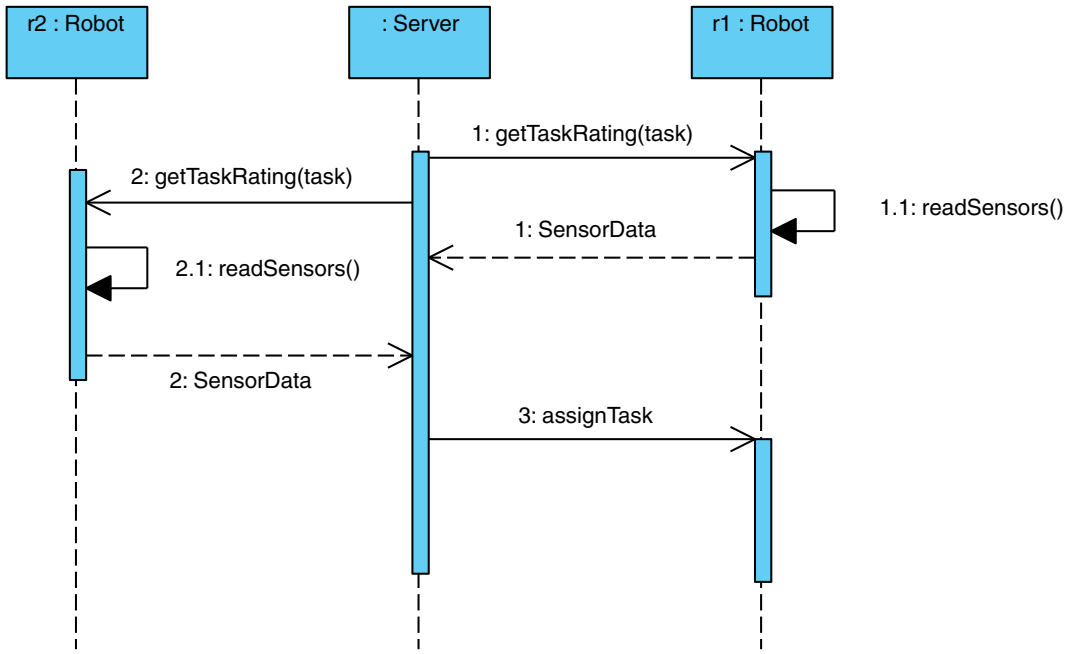

Abbildung 2: *ChooseRobot*-Sequenzdiagramm

3 Komponentenschnittstellen

Die in Abbildung 3 dargestellten Datentypen werden von den Schnittstellen der eingeführten Komponenten verwendet.

Die Aufgabe des Datentyps *SensorData* ist primär, die Koordination mit dem Server zu unterstützen, um festzustellen, inwieweit eine Zielposition gut erreicht werden kann. Dazu besitzt er als Attribute die Orientierungsrichtung im Koordinatensystem, den Batteriestatus und zuletzt Attribute der Datentypen *Position* und *Destination*. *Position* besitzt wiederum die Koordinaten x und y, die einen beliebigen Punkt im Bereich des Einsatzgebietes des Roboters darstellen. Um Redundanz zu vermeiden sind *Destination* und *Position* über keine zusätzliche Klassendiagrammbeziehung verknüpft. Zielpunkte erben von *Position* und existieren zur Spezifikation, um welche Art Zielpunkt es sich handelt: Also zum Beispiel eine allgemeine *Destination* oder den *Charger*.

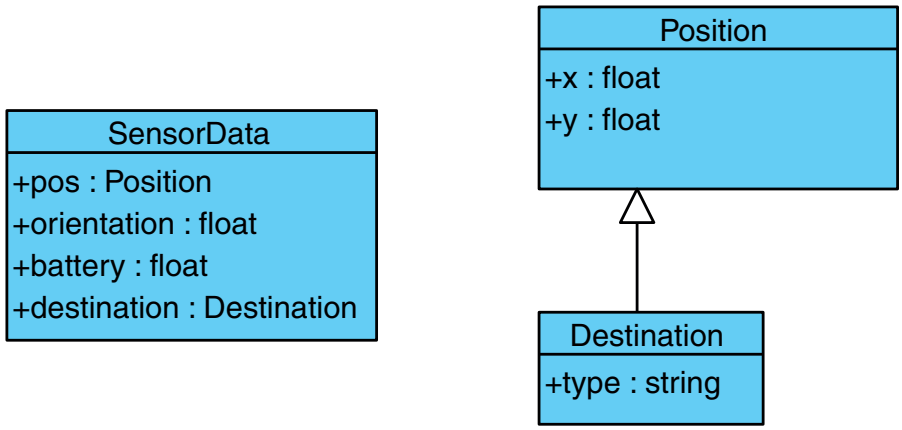

Abbildung 3: Datentypen, die in Komponentenschnittstellen verwendet werden

4 Konkrete Architektur

Abbildung 4 zeigt die Komponenten *ServerSoftware* und *RobotSoftware* in einem Komponentendiagramm.

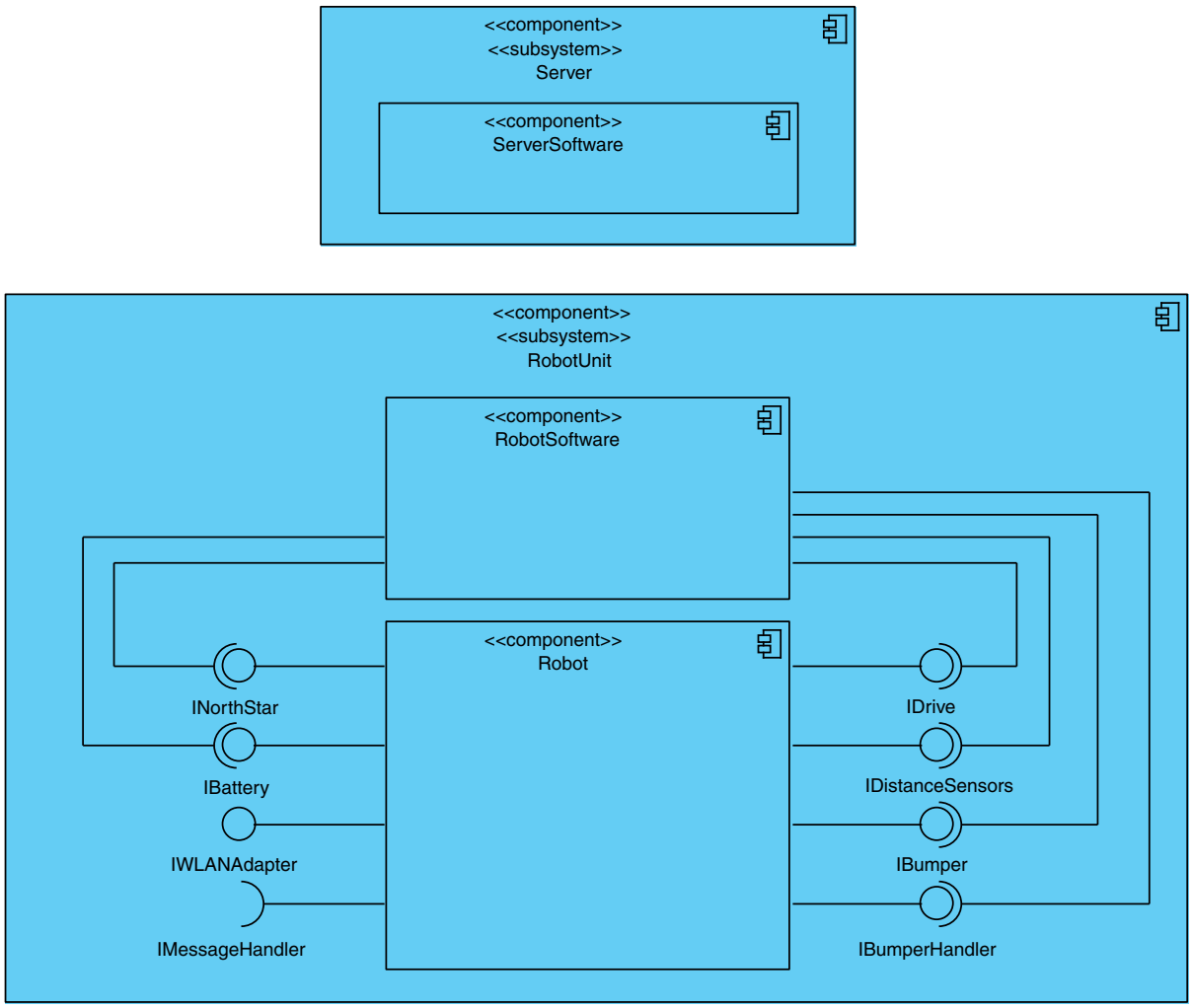

Abbildung 4: Komponentendiagramm mit den Komponenteninterfaces

4.1 ServerSoftware

Die *ServerSoftware* tätigt in der aktuellen nullten Ausbaustufe direkt Aufrufe an den *Robot*, um den besten *Robot* für eine *Destination* zu ermitteln und einem *Robot* eine *Destination* zuzuweisen.

4.2 RobotSoftware

RobotSoftware ist für die Steuerung der *RobotUnit* zuständig. Die von der abstrakten Hardware des *Robot* angebotenen, vorgegebenen Interfaces *INorthStar, IBattery, IDrive, IDistanceSensors, IBumper* sowie *IBumperHandler* werden dabei von der *Robot*-Komponente für die konkrete Steuerung der *RobotUnit* genutzt.

5 Komponenten

5.1 Komponente ServerSoftware

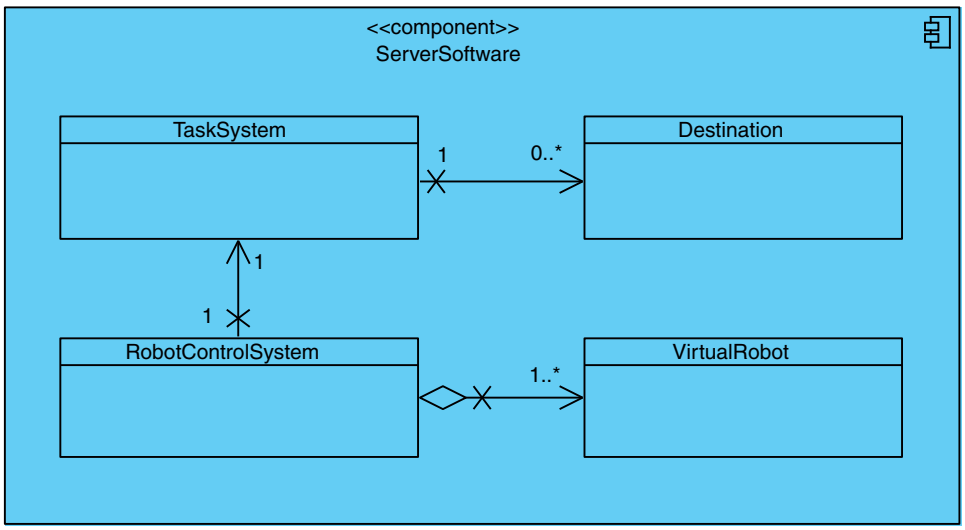

Abbildung 5: *ServerSoftware*-Komponentendiagramm

Abbildung 5 zeigt ein Komponentendiagramm von *ServerSoftware*. Diese Komponente umfasst 4 Klassen: *TaskSystem*, *Destination*, *RobotControlSystem* und *VirtualRobot*.

Das *TaskSystem* verwaltet die *Tasks*, in der nullten Ausbaustufe fallen dabei allerdings noch nicht viele Aufgaben an, da es keine Priorisierungen, unterschiedliche Arten von *Tasks* oder dergleichen gibt. Die Klasse *Destination* ist die Klasse der Aufgaben, die vom *TaskSystem* verwaltet werden. Das *RobotControlSystem* verteilt die *Tasks* auf die *Robots*. Dabei findet die Auswahl anhand der zurückgegebenen *SensorData* der einzelnen *Robots* statt. Bei *VirtualRobot* handelt es sich um eine Kapselung der Kommunikation mit den *Robots*. Hier werden alle von der Komponente *Robot* bereitgestellten *Interfaces* implementiert.

5.2 Komponente RobotSoftware

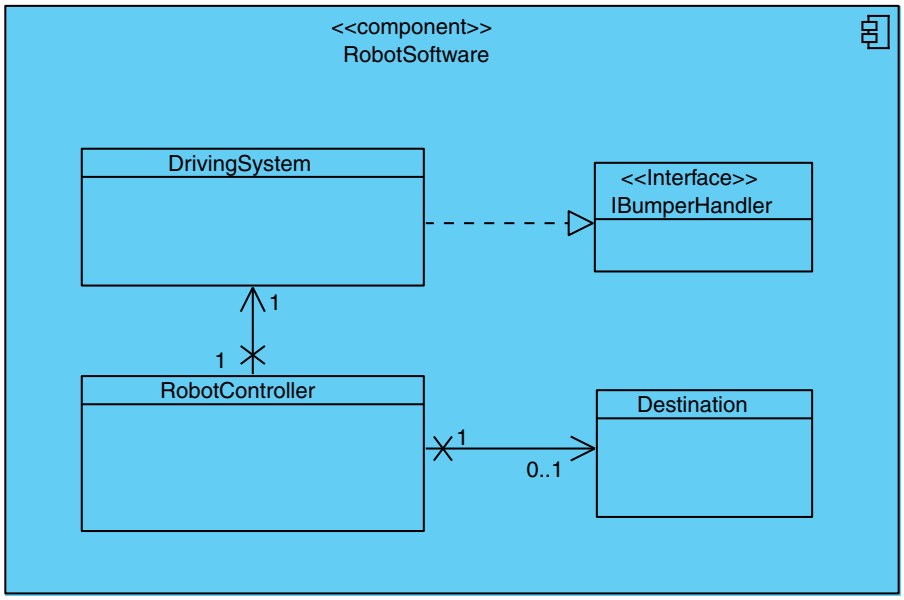

Abbildung 6: *RobotSoftware*-Komponentendiagramm

In Abbildung 6 ist das Komponentendiagramm der Komponente *RobotSoftware* dargestellt. Sie enthält 3 Klassen: *DrivingSystem*, *RobotController* und *Destination*.

Das *DrivingSystem* stellt eine Abstraktion der Hardware dar und wird dazu genutzt, Ziele anzufahren und dabei, falls nötig, Hindernisse zu umfahren. Dazu greift es auf die von der Hardware bereitgestellten *Interfaces* zurück. Um auf Kollisionen reagieren zu können, implementiert das *DrivingSystem* die Schnittstelle *IBumperHandler* Der *RobotController* stellt dem Server das *Interfaces ISensorData* zur Verfügung und verwaltet den gerade zu absolvierenden *Task* in Form einer *Destination*. Zur Messwertübermittlung greift er zum einen auf das *DrivingSystem* und zum anderen auf das von der Hardware angebotenen *Interface IBattery* zu.

6 Paketstruktur

Dieser Abschnitt beschreibt die strukturelle Gliederung des Projektes in einem Paketdiagramm, dargestellt in Abbildung 7.

Wir betrachten die Pakete *Robot* und *Server* getrennt, da es eine physikalische Trennung zwischen den Geräten gibt, auf denen die jeweiligen Pakete vorhanden sein müssen.

Im *Common*-Paket sind alle Klassen enthalten, die sowohl vom *Robot*- als auch vom *Server*-Paket genutzt werden. So sind in dieser Iterationsstufe lediglich die Datentypen *Position* und die davon erbende *Destination* enthalten. Eine Möglichkeit zur Unterscheidung von *Destinations* ist wichtig, da zwischen vom *Server* zugeteilten *Destinations* und vom *Robot* selbst zugeteilten Ladestationen als Ziel unterschieden werden muss. Jegliche Klassen und Interfaces die für die Kommunikation zuständig gewesen wären, wurden in dieser Iterationsstufe gemäß Aufgabenstellung nicht modelliert.

Das Paket *Robot* enthält das Paket *HardwareInterfaces*, welches die Möglichkeiten schafft, die Hardwareschnittstellen des *Robots* direkt anzusprechen. Das Paket *RobotControl* enthält Klassen zur Steuerung und Speicherung des Zustandes des *Robots*, sowie die *Handler*, die den Hardwareschnittstellen übergeben werden.

Der Server besitzt in dieser Iterationsstufe noch keine Hardwareschnittstellen und enthält daher lediglich Klassen zur Verwaltung der *Robots* und den jeweiligen *Destinations*.

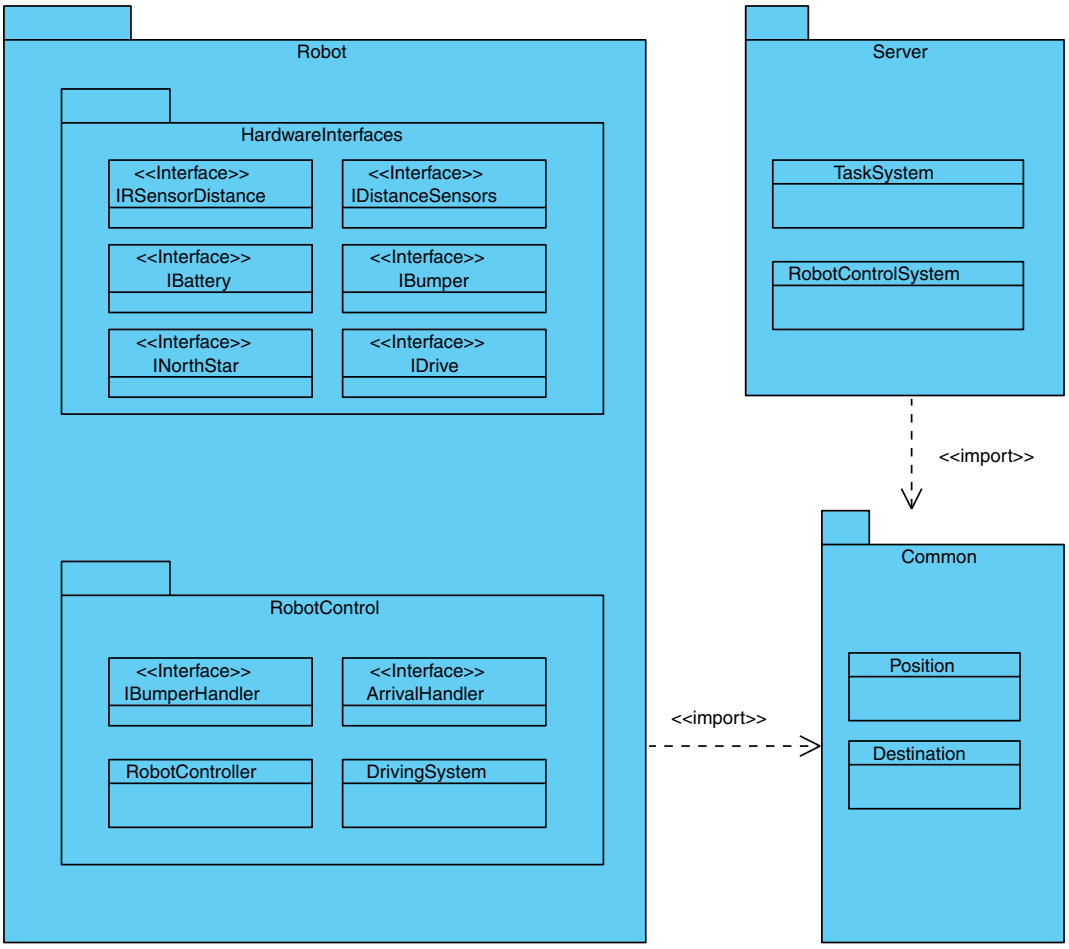

Abbildung 7: Paketdiagramm zur strukturellen Gliederung der Software

7 Paketdetails

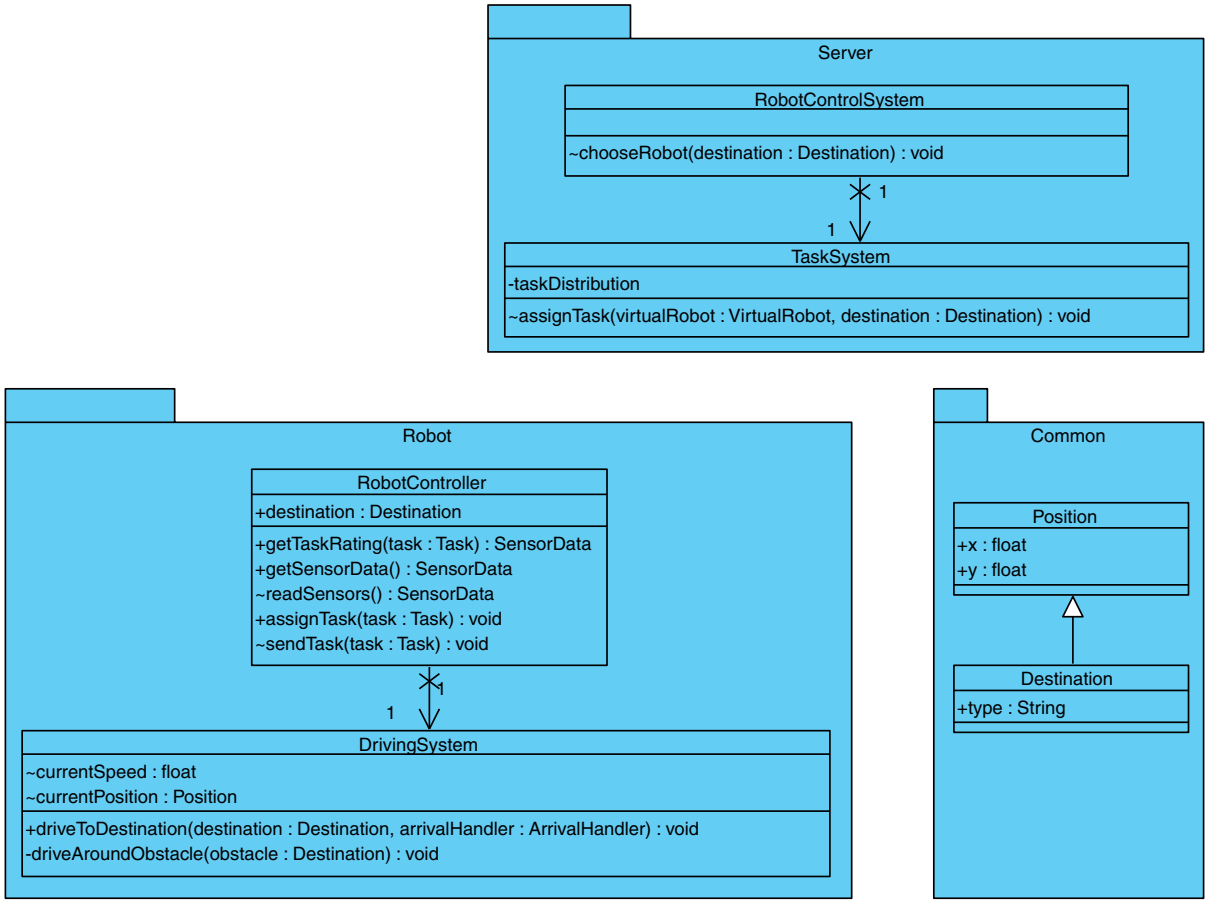

Abbildung 8: Klassendiagramm zur detaillierten Beschreibung der strukturellen Gliederung

7.1 Paket Robot

Im Folgenden beschreiben wir die wichtigen Klassen des Pakets *Robot* und ihre zugehörigen wichtigen Methoden, sowie ihre Interaktion untereinander.

7.1.1 Beschreibung der Klasse RobotController

Die Klasse *RobotController* ist die Hauptklasse des *Robots*, da sie den aktuellen Zustand des *Robots* enthält. So hat diese Klasse die Möglichkeit, maximal eine *Destination* zu speichern. Diese *Destination* kann ein vom *Server* zugeteiltes Ziel sein, der dem *Robot* zugehörige *Charger*, oder gerade kein Ziel, also `null` sein. Nur wenn der *Robot* gerade keine *Destination* gespeichert hat, kann er neue Aufträge vom *Server* annehmen.

Beschreibung der Methode `assignTask`

Der Server kann in dieser Aufbaustufe direkt auf die Methoden vom *Robot* zugreifen. Mit *assignTask* kann er dem *Robot* so direkt eine neue *Destination* zuweisen.

Beschreibung der Methode `getTaskRating`

Der Server lässt einen jeweiligen *Robot* prüfen, wie gut seine Aktuelle Lage in Bezug auf eine *Destination* ist. So sammelt der *Robot* seine Sensordaten und wertet diese aus.

Beschreibung der Methode `collectSensorData`

Der Server kann einen *Robot* dazu auffordern, ihm seine Soensordaten zu schicken. Der *Robot* fragt dann seine Hardwareschnittstelle mittels der Methode `readSensors` nach seiner Position und seinem Akkustand an, und gibt diese Informationen, zusammen mit Informationen über den Aktuellen Zustand des *Robots* bzw. seine aktuelle *Destination*, zurück and den *Server*.

7.1.2 Beschreibung der Klasse DrivingSystem

Diese Klasse beschreibt den aktuellen Zustand des Fahrsystems des *Robots*. Es sind Informationen über die aktuelle Geschwindigkeit enthalten und die Methode, die gerade ausgeführt wird, gibt Auskunft über die aktuelle Beschäftigung des *Robots*.

Beschreibung der Methode `driveToDestination`

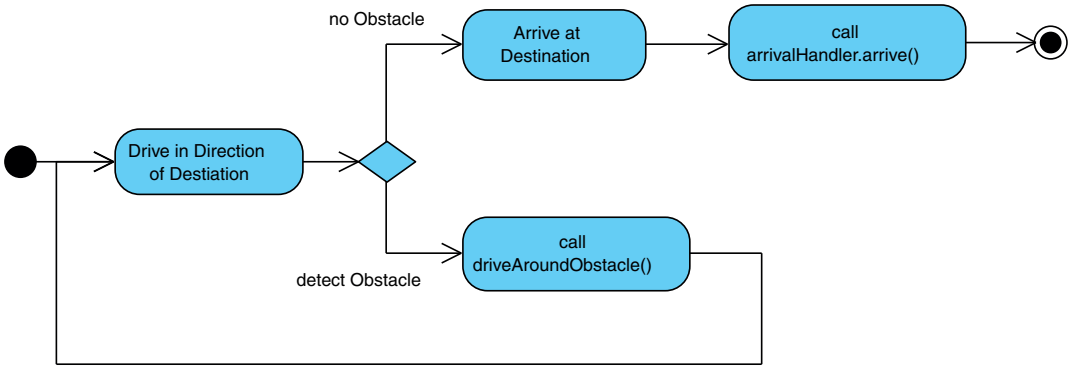

Abbildung 9: Aktivitätsdiagramm zu Metode `driveToDestination`

Wenn diese Methode aufgerufen wird, macht der *Robot* sich auf den Weg zur übergebenen *Destination*. Wenn der *Robot* an dieser *Destination* angekommen ist, wird die `arrive`-Methode des übergebenen *ArrivalHandlers* ausgeführt. Wenn sich ein *Obstacle* auf dem Weg befindet, wird die Methode `driveAroundObstacle` aufgerufen, bis das *Obstacle* umfahren wurde.

Abbildung 9 zeigt ein entsprechendes Aktivitätsdiagramm.

Beschreibung der Methode `driveAroundObstacle`

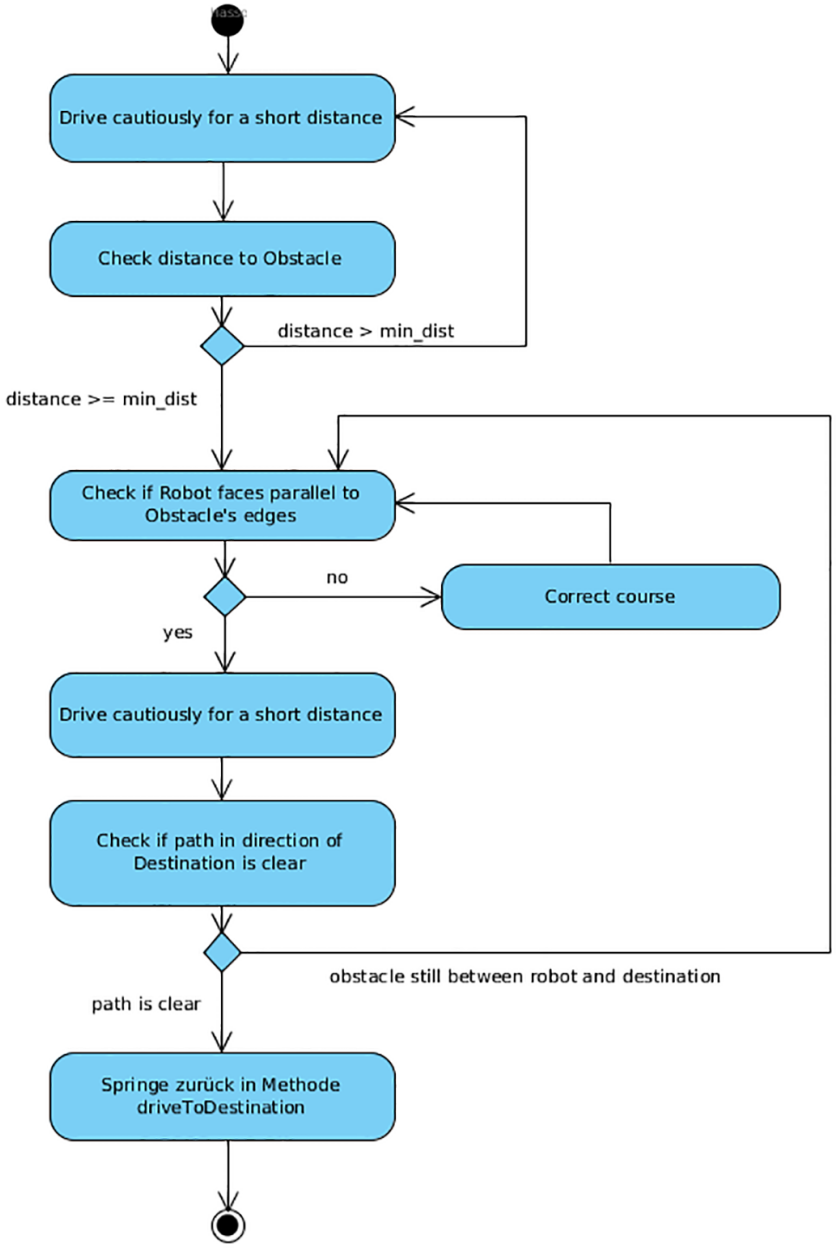

Abbildung 10: Sequenzdiagramm zur Beschreibung der Methode `driveAroundObstacle`

Diese Methode wird von `driveToDestination` mit des Position eines *Obstacles* aufgerufen, wenn ein *Obstacle* zu umfahren ist. Dabei entscheidet sich der Roboter zunächst ob er links oder rechts an dem *Obstacle* vorbeifährt, und hält sich dann mithilfe seiner Sensoren immer auf einem bestimmten Abstand zum Hindernis, bis zwischen *Obstacle* und der Luftlinie zur *Destination* genug Platz für den *Robot* ist.

Abbildung 10 zeigt ein entsprechendes Sequenzdiagramm. Dabei ist `min_dist` eine vorher festgelegte Konstante, welche die Mindestdistanz, die der *Robot* halten muss, wenn er an einem *Obstacle* vorbeifährt, speichert.

7.2 Paket Server

Im folgenden beschreiben wir die wichtigen Klassen des Pakets *Server* und ihre zugehörigen Methoden, sowie ihre Interaktion zwischeneinander.

7.2.1 Beschreibung der Klasse TaskSystem

Das *TaskSystem* des *Servers* verarbeitet alle *Tasks*, die es mit der Zuordnung vom *RobotControlSystem* übergeben bekommt. Dafür besitzt es eine Struktur, die die *taskDistribution* intern verwaltet und somit die *Tasks* und die jeweils zugeordneten *Robots* kennt.

Beschreibung der Methode `assignTask`

Die Methode `assignTask` führt die Zuordnung und Abspeicherung der Roboter und *Tasks* bzw. *Destinations* durch.

7.2.2 Beschreibung der Klasse RobotControlSystem

Das *RobotControlSystem* sorgt bei eingehenden *Tasks* dafür, dass ein passender *Robot* ausgewählt wird. Diese Information übergibt es dann an das *TaskSystem*, das für die endgültige Zuordnung zuständig ist.

Beschreibung der Methode `chooseRobot`

Die Methode `chooseRobot` wählt für den aktuell eingegangenen *Task* einen *Robot* aus. Dazu fragt es die Sensorwerte der verschiedenen *Robotes* ab und wählt den am besten geeigneten aus.

8 Abläufe

Während in Kapitel 3 die Interaktion der Hauptkomponenten *Server* und der *RobotUnit* abgehandelt wurden, werden im Folgenden die Abläufe der Use Cases genauer spezifiziert und auch interne Komponentenabläufe beschrieben, im speziellen die Abläufe innerhalb der *RobotUnit*.

Interaktion bei Ausführung von Use Case 1 – DriveToDestination

Wie schon der Name des Use Cases *DriveToDestionation* beschreibt, befindet sich die *RobotUnit* bei Ausführung diese Use Cases in einem Fahrvorgang, der vorher konkret mit der Zielposition und er Geschwindigkeit vom Server eingestellt wird. Es reagieren intern ihre Software mit den Sensor-/Hardwarekomponenten, wenn ein Hindernis auftaucht und dieses mit Hilfe der durch die Sensoren gesammelten Informationen umfahren werden muss. Der unmittelbare Fahrvorgang hingegen wird über die *IDrive*-Schnittstelle gesteuert, die zurückmeldet, wenn der *Robot* sein Ziel erreicht hat.

Abbildung 11 zeigt ein entsprechendes Sequenzdiagramm.

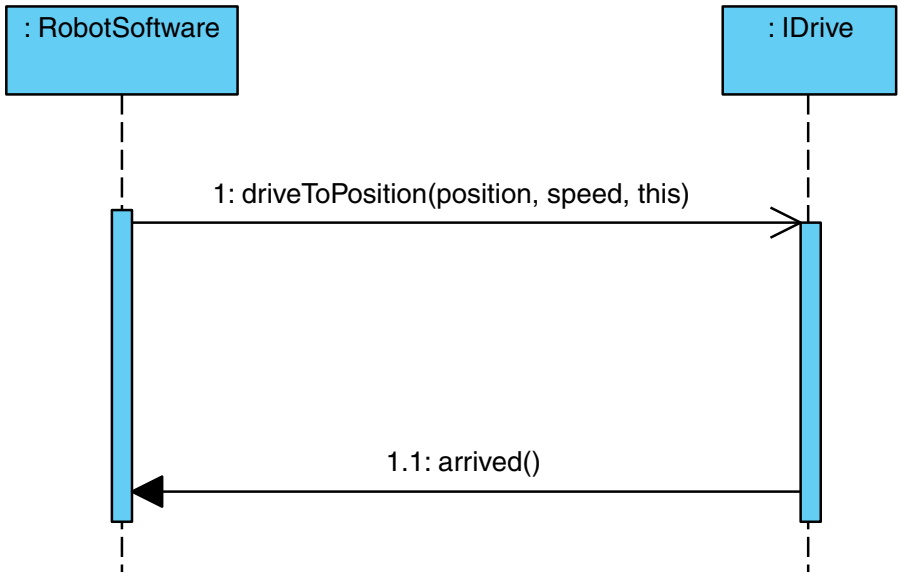

Abbildung 11: Sequenzdiagramm von *DriveToDestination*

Interaktion bei Ausführung von Usecase 2 – ReadSensors

In Abbildung 12 wird der reine Anfrageprozess zwischen *Server* und der *RobotUnit* aus Kapitel 3 erweitert. Nach der Anfrage wird der *Robot* alle nötigen Informationen naheinander abfragen: Sowohl die *Position* als auch die *Orientation* werden von der *INorthStar*-Schnittstelle zurückgeliefert. Für den Batteriestatus muss die *IBattery*-Schnittstelle angefragt werden. Erst wenn alle Informationen als Gesamtpaket bereitstehen, können sie an den *Server* zurückgemeldet werden.

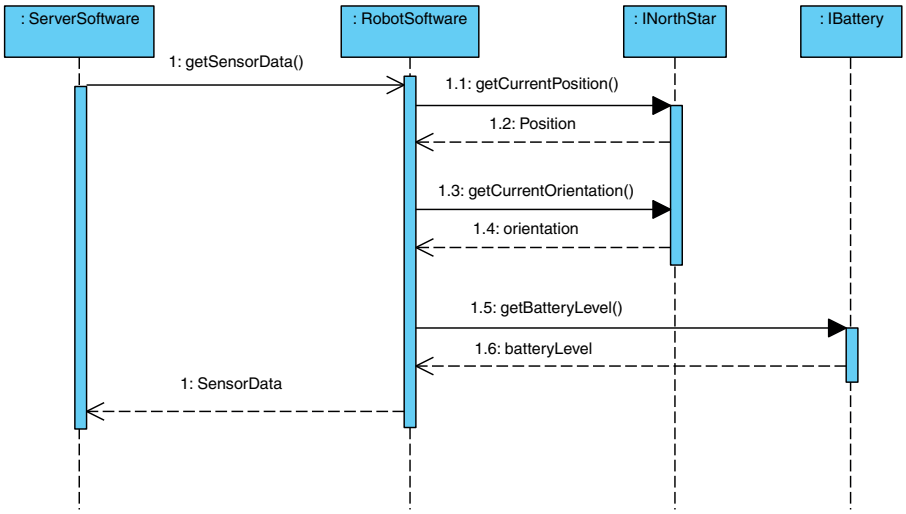

Abbildung 12: Sequenzdiagramm von *ReadSensors*

Interaktion bei Ausführung von Use Case 3 – Charging

Auch beim Use Case *Charging* findet keine Kommunikation mit der Komponente *Server* statt, dafür allerdings zwischen der *RobotUnit* und der *ChargingStation*. Hat der *Robot* einen bestimmten kritischen Ladestand erreicht (den er regelmäßig überprüft), steuert er die *ChargingStation* an. Der Ladevorgang triggert automatisch, sobald der *Robot* die *Position* der Ladestation erreicht hat, und er interagiert so lange mit ihr, bis seine Batterie wieder aufgeladen ist. Innerhalb des *Robots* werden zum Anfahren der *ChargingStation* die Komponenten *IBattery* mit der Position der robotereigenen Ladestation und *IDrive* benötigt. Konkret: Wenn *IDrive* `arrived()` zurückgibt, kann der Ladevorgang begonnen werden.

Abbildung 13 zeigt ein Sequenzdiagramm für den Use Case *Charging*.

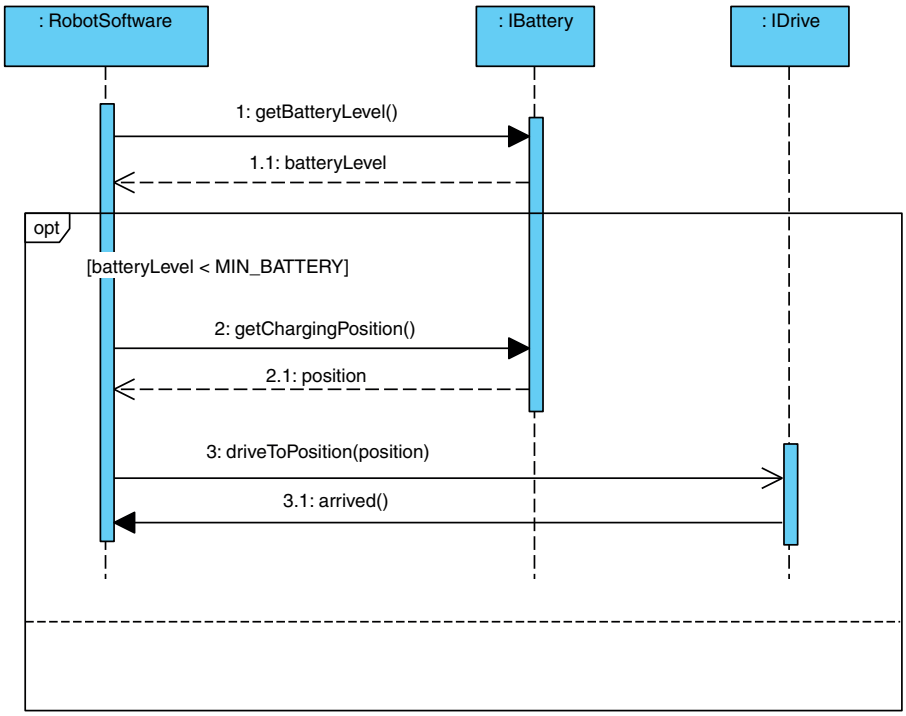

Abbildung 13: Sequenzdiagramm von `Charging`

9 Produkteinsatz

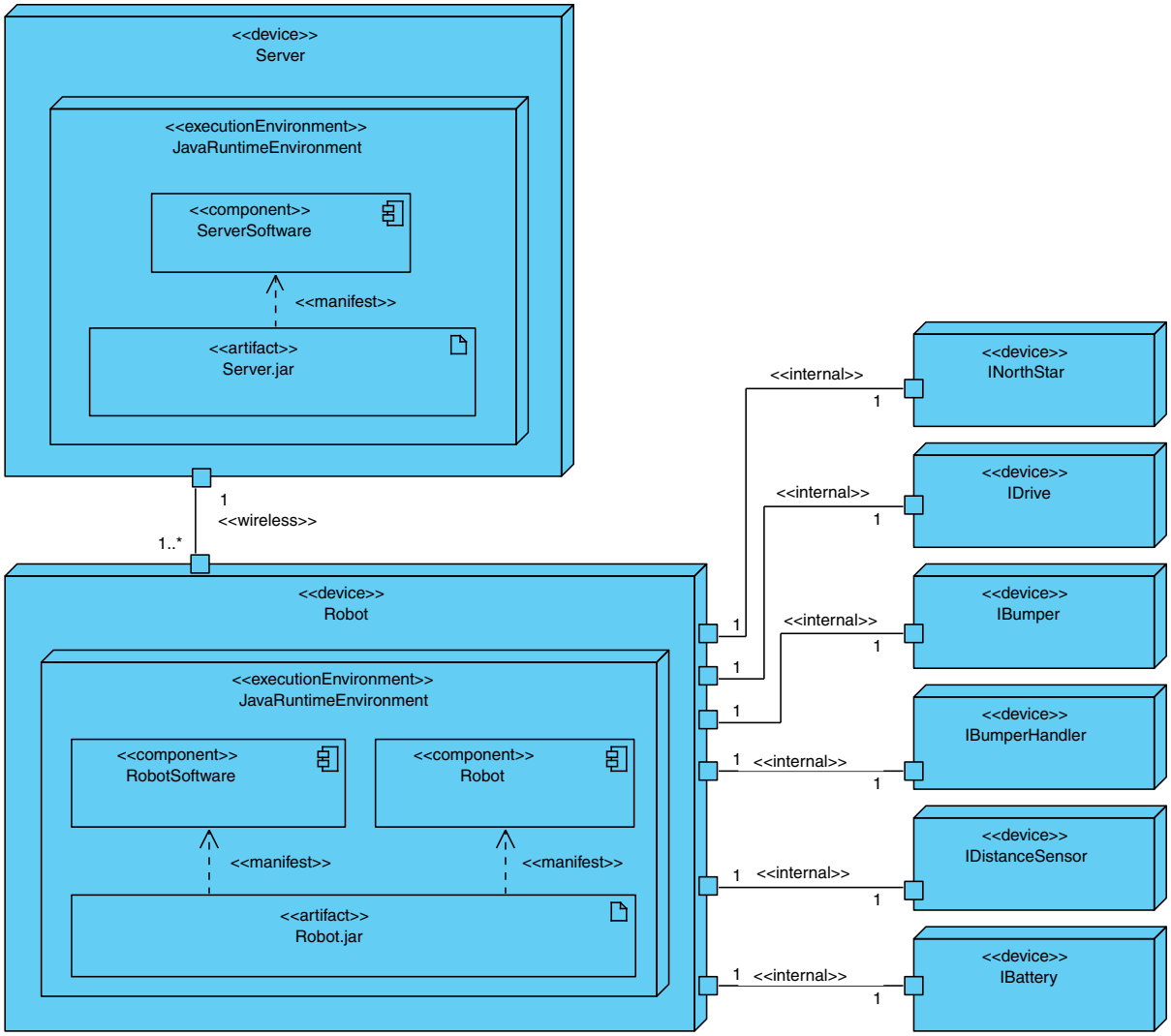

Abbildung 14: Verteilungsdiagramm des Gesamtsystems

Abbildung 14 zeigt einen Gesamtüberblick über den Produkteinsatz.

In diesem Abschnitt wird der geplante Einsatz des Systems beschrieben, wobei insbesondere auf die Systemumgebung, in der das Produkt eingesetzt werden soll, und die Zuordnung der Software zu dieser eingegangen wird.

Das Gesamtsystem besteht u.a. aus einem *Server*, der mit mindestens einem *Robot* verbunden ist, wobei zwischen *Server* und *Robot* direkt kommuniziert werden kann. *Robots* können nicht mit anderen *Robots* kommunizieren.

Auf dem *Server* und den *Robots* läuft ein *Java Runtime Environment*, das dem Ausführen der entsprechenden Software dient.

Robot.jar dient der Kapselung der verfügbaren Funktionen. Es werden auch die *Interfaces* zusammengefasst, die dem Ansprechen von *IDrive*, *INorthStar*, *IDistanceSensor*, *IBumper*, *IBumperHandler* und *IBattery* dienen.

Server.jar funktioniert analog zu *Robot.jar* und stellt alle grundlegenden Funktionen zur

Verfügung.

Zusätzlich benötigte Funktionalitäten werden aus dem Common-Paket importiert und sind sowohl bei *Robots* als auch dem *Server* verfügbar.