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	09.05.2016
Bearbeitender	Sauder, Jonathan	09.05.2016
Bearbeitender	Lüpke, Fabian	09.05.2016
Bearbeitender	Hering, Jonas	09.05.2016
Bearbeitender	Braun, Jakob	09.05.2016
Bearbeitender	Cremerius, Jonas	09.05.2016
Bearbeitender	Wischner, Jakob	09.05.2016
Bearbeitender	Schwenkert, Daniel	09.05.2016
Bearbeitender	Jäkel, Dominik	09.05.2016
Bearbeitender	König, Bastian	09.05.2016

Inhaltsverzeichnis

1 Zielbestimmung

Die Aufgabe des Projektteams ist es ein System zu entwickeln, das selbstfahrender Roboter in einer Stadt der Zukunft koordiniert. In dieser Stadt existieren keine Straßen mehr – umso komplexer wird der Prozess das Navigationsverhalten der Roboter mit den richtigen Parametern einzustellen und zu modellieren. Die Voraussetzungen und Spezifikation dafür zu erfassen ist Ziel dieser Analyse – die sich insbesondere mit dem Fahr- und dem Aufladevorgang der Roboter und ihren Manövrierfähigkeiten gegenüber Hindernissen auseinandersetzt.

Im weiteren Verlauf des Projektes sollen die Roboter ebenfalls Krankentransporter und schließlich Bestandteil eines Taxiservicedienstes werden. Alle hier getroffenen Annahmen, Modellierungen und Implementierungen könnten damit im Rahmen der späteren Anwendungen revidiert werden.

Als Ausgangspunkt stellt das Unternehmen eine unbegrenzte Anzahl von Robotern zur Verfügung, die mit Aktoren, Sensoren und einem Ortungsgerat ausgestattet sind, und im Wechselspiel mit einem Server agieren.

Das Primärziel des Projektes ist es dabei ein möglichst effizientes Transportsystem zu entwickeln, um für alle Verkehrsbegebenheiten im Großraum einer Zukunftsstadt vorbereitet zu sein. Davon würde die Zielgruppe Mensch nachhaltig profitieren, indem Verkehrsunfällen, Staus und die zeitliche Inanspruchnahme des Fahrens auf ein Minimum reduziert werden. Allerdings wird das System zuerst vor einige Probleme im Fahrvorgang gestellt, mit denen sich die folgenden Kapitel beschäftigen – werden sie gelöst, könnte langfristig ein einzelnes Unternehmen die gesamte Transportinfrastruktur einer Großstadt Übernehmen. Das System wird von sich aus selbstständig und vollständig autonom agieren. Abgesehen davon Sicherheitsregeln einzuhalten, sollten damit keine Vorkenntnisse für die Fahrt mit den Robotern notwendig sein, um den Personennahvekehr für die breite Masse zu öffnen.

2 Produkteinsatz

2.1 Beschreibung des Problembereichs

Der Problembereich lässt sich für diese Analysestufe in drei Punkte einteilen, um eine Gesamtkoordination effizient zu gestalten:

1. Zuallererst muss das automatische Fahren im zweidimensionalen Koordinatensystem zu einem angegebenen Zielpunkt(*Destination*) fehlerfrei möglich sein. Ebenfalls steht eine manuelle Steuerung zur Verfügung, um unerwarteten Ereignissen zu begegnen, die allerdings wiederum sinnvoll in das Gesamtsystem integriert werden muss.

2. Als zweite Herausforderung geht es darum den Schaden von Kollisionen zu minimieren; wie diese erkannt, wie auf sie reagiert wird und diese Kollisionen durch vorrauschendes Umfahren von Objekten (*Obstacle*) verhindert werden können. Dies schließt sowohl bewegliche Objekte als auch unbewegliche Objekte mit ein.

3. Im dritten Punkt gilt es einen möglichst reibungs- und unterbrechungslosen Batteriebetrieb zu garantieren, um zeitlich vorrauschendes Fahren zu ermöglichen. Im Falle eines niedrigen Batteriestands also unverzüglich an seiner individuellen Ladestation (*Charger*) anzudocken und aufzuladen, beziehungsweise längere Fahrten nicht anzutreten, die den Batteriestand gefährden würden.

(a) Fahren zu einem angegebenen Zielpunkt

(b) Umfahren von Objekten

(c) Automatisches Laden

Abbildung 1: Visualisierung der Problembereiche

Abbildung 1 zeigt die drei o.g. Problembereiche in einer fotografischen Repräsentation.

Daran anknüpfend wird mit folgenden Annahmen gearbeitet:

1. Zur Vereinfachung des Fahrverhalten muss jedes Zielobjekt physisch erreichbar sein. Ziele sind dementsprechend keine Hindernisse, das gleiche gilt für Ladestationen.

2. Roboter können autark vom *Server* agieren, damit ihre Ziele bestimmen und eine Ladestation anfahren.

3. Bei den beweglichen Objekten wird davon ausgegangen, dass es sich im Verkehrsbetrieb ausschließlich um andere Roboter handelt. Für die unbeweglichen Objekte ist es notwendig, dass ihre Position von vornerein bekannt ist und sie unverändert bleibt.

2.2 Glossar

Begriff	Erklärung
Robot	Roboter, der sich im 2-Dimensionalen Raum bewegen kann.
Server	Verteilt Aufträge an die Roboter
Charger	Ladestation für den Roboter
Destination	Vom Server verwaltete Ziele, die der Roboter ansteuern kann.
Obstacle	Hindernisse, die der Roboter erkennt und umfährt

2.3 Modell des Problembereichs

Abbildung 2 zeigt ein Klassendiagramm, welches das Modell des Problembereichs grafisch darstellt.

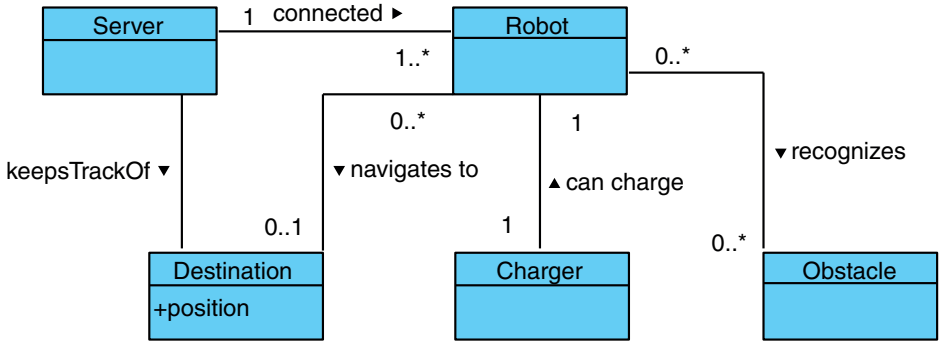

Abbildung 2: Klassendiagramm des Problembereichs

2.4 Beschreibung der Geschäftsprozesse

Beschreibung zu 1: Choose Robot

Auslösendes Ereignis:	Der Server will den für ein bestimmtes Ziel bestmöglichen Roboter auswählen und diesen zum Ziel schicken.
Ergebnis:	Alle Roboter haben ihre Sensordaten ausgelesen und diese dem Server mitgeteilt. Daraufhin hat der Server einen Roboter ausgewählt, der für die Zielanfahrung optimal geeignet ist, und diesem den Auftrag zum Anfahren des Ziels übermittelt.
Mitwirkende:	Server, Roboter

Abbildung 3 zeigt ein Aktivitätendiagramm, welches den Ablauf des Geschäftsprozesses *1: Choose Robot* darstellt.

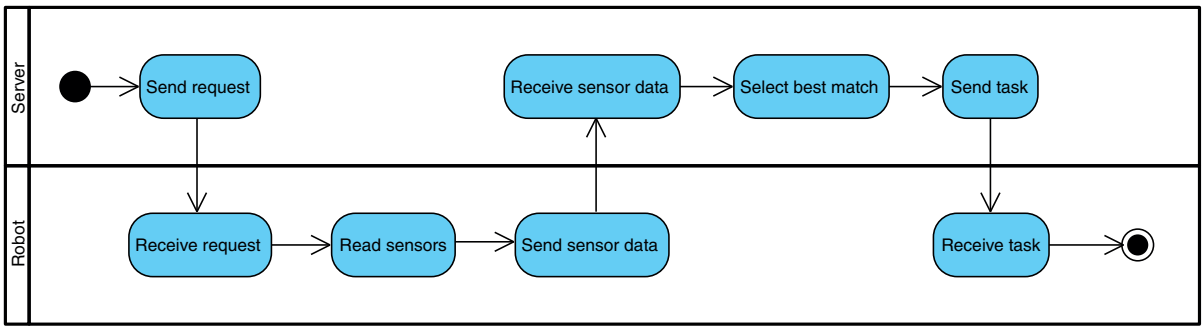

Abbildung 3: Illustration von *1: Choose Robot* durch Aktivitätendiagramm

Um den für ein gegebenes Ziel bestmöglichen *Robot* auszuwählen, sendet der *Server* zunächst an alle *Robots* eine Anfrage (*request*), woraufhin die *Robots* ihre Sensordaten (*sensor data*) auslesen und an den *Server* senden. Auf Grundlage dieser Daten wählt der *Server* den *Robot* aus, der für die Zielanfahrung optimal positioniert ist und sendet ihm die konkrete Aufforderung (*task*) zum Anfahren des Ziels.

Wie die Sensordaten und der Algorithmus zur Wahl des optimalen *Robots* konkret definiert sind, ist eine Entwurfsentscheidung, die zum Zeitpunkt dieser ersten Ausbaustufe noch nicht getroffen wurde.

Beschreibung zu 2: Drive to Destination

Auslösendes Ereignis:	Ein *Robot* hat eine *Destination* erhalten.
Ergebnis:	Der *Robot* ist unter Umfahrung von womöglichen *Obstacles* zur *Destination* gefahren.
Mitwirkende:	Ein einzelner, autonomer *Robot*.

Abbildung 4 zeigt ein Aktivitätendiagramm, welches den Ablauf des Geschäftsprozesses *2: Drive to Destination* darstellt.

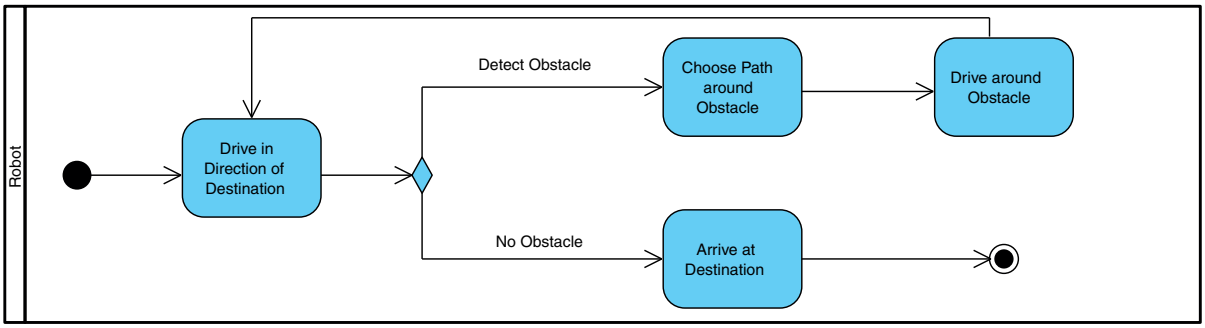

Abbildung 4: Illustration von *2: Drive to Destination* durch Aktivitätendiagramm

Dieser Prozess wird autonom vom *Robot* ausgeführt. Er wird genau dann ausgeführt, wenn der Roboter eine *Destination* vom Server erhalten hat, die er anfahren soll. Da der *Robot* an dieser Stelle dem Server schon bestätigt hat, dass sein Akkustand hoch genug ist, ist der einzige Sonderfall, den wir betrachten müssen, dass ein *Obstacle* auf der direkten Linie zwischen *Robot* und *Destination* liegt. Dann wird der *Robot* nach Wahl des besten Umweges das *Obstacle* umfahren und wieder die Fahrt zur *Destination* Aufnehmen.

3 Produktfunktionen

3.1 Use Cases

Die Abbildungen 5 und 6 zeigen die Funktionalitäten der beiden Teilsysteme *Robot* und *Server*.

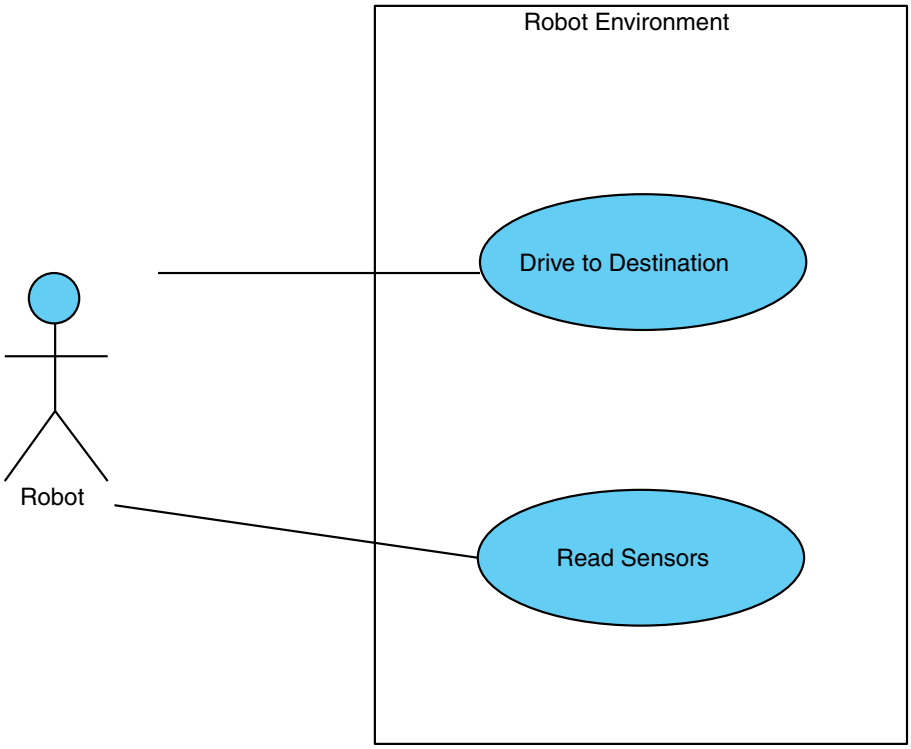

Abbildung 5: Use Case Diagramm 1: Use Cases des Roboters

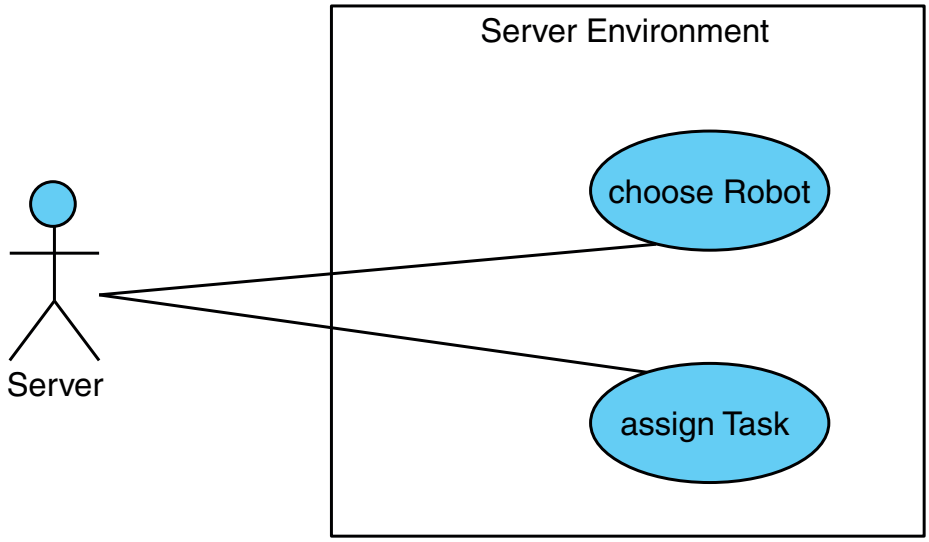

Abbildung 6: Use Case Diagramm 2: Use Cases des Servers

3.2 Beschreibung zu Use Case 1: Drive to Destination

Charakterisierende Informationen

Übergeordneter elementarer Geschäftsprozess:	Drive to Destination
Ziel des Use Cases:	*Robot* fährt eine *Destination* an
Umgebende Systemgrenze:	*Robot*
Vorbedingung:	Ein spezieller *Robot* wurde vom Server ausgewählt und sein Akkustand ist hoch genug um den Auftrag auszuführen
Nachbedingung bei erfolgreicher Ausführung:	*Robot* ist an der *Destination* angekommen und meldet dies dem *Server*
Beteiligte Nutzer:	*Robot*
Auslösendes Ereignis:	*Robot* hat eine *Destination* erhalten.

Dieser Use Case wird dann benutzt, wenn ein *Robot* eine aktuelle *Destination* hat. In dieser Ausbaustufe betrachten wir lediglich das Anfahren einer *Destination* von einem *Robot*. Der *Robot* dreht sich zunächst in die Richtung der *Destination* und fährt so lange in Luftlinie, bis er entweder ankommt oder ein Hindernis erkennt. Ein konkreter Prozess zur Erkennung eines Hindernisses ist eine Entwurfsentscheidung und wird daher noch nicht berücksichtigt. Wenn der *Robot* ein *Obstacle* erkennt, dann sucht er einen guten Weg um das Hindernis herum. Wie er diesen Weg sucht ist auch zunächst eine Entwurfsentscheidung und noch nicht relevant. Auf diesem Weg umfährt er dann das *Obstacle* und nimmt erneut die Fahrt auf Luftlinie auf.

Szenario für den Standardablauf (Erfolg)

Schritt	Nutzer	Beschreibung der Aktivität
1	Robot	*Robot* dreht sich in Richtung aktueller *Destination*
2	Robot	*Robot* fährt geradeaus
3	Robot	*Robot* erreicht *Destination*

Szenarien für alternative Abläufe (Misserfolg oder Umwege zum Erfolg)

Schritt	Bedingung für Alternative	Beschreibung der Aktivität
3	*Robot* erkennt ein *Obstacle* zwischen auf seinem Weg zum Ziel	*Robot* sucht sich einen Umweg um das *Obstacle*, umfährt das *Obstacle* und nimmt schließlich Standardablauf wieder auf

3.3 Beschreibung zu Use Case 2: Read Sensors

Charakterisierende Informationen

Übergeordneter elementarer Geschäftsprozess:	Choose Robot
Ziel des Use Cases:	*Robot* kann über seinen Akkustand und seine Position Auskunft geben
Umgebende Systemgrenze:	*Robot*
Vorbedingung:	*Robot* hat eine Anfrage vom *Server* erhalten
Nachbedingung bei erfolgreicher Ausführung:	*Robot* schickt die ermittelten Informationen an den *Server*
Beteiligte Nutzer:	*Robot*
Auslösendes Ereignis:	*Robot* hat eine Anfrage vom *Server* erhalten, seine Sensoren zu lesen und sie dem *Server* zu schicken

Im Rahmen vom Geschäftsprozess *Choose Robot* sammelt der *Server* Informationen über jeden *Robot*. Diese Informationen (z.B Akkustand, aktuelle Position, ob der *Robot* gerade ein Ziel verfolgt) kann der *Robot* von seinen Hardware-Schnittstellen anfordern. Dieses Use Case wird dann ausgeführt, wenn der *Robot* eine Anfrage vom Server erhält, seine Sensoren zu lesen, und endet damit, dass der *Robot* die zusammengefassten Informationen an den *Server* schickt.

Szenario für den Standardablauf (Erfolg)

Schritt	Nutzer	Beschreibung der Aktivität
1	Robot	*Robot* erhält Anfrage vom *Server*
2	Robot	*Robot* sammelt Informationen von seiner Hardwareschnittstelle und fasst sie zusammen
3	Robot	*Robot* schickt zusammengefasste Informationen an den Server

3.4 Beschreibung zu Use Case 3: Charging

Charakterisierende Informationen

Übergeordneter elementarer Geschäftsprozess:	Drive to Destination
Ziel des Use Cases:	Ziel ist es, dem *Robot* zu ermöglichen seine Ladestation anzufahren
Umgebende Systemgrenze:	*Robot*
Vorbedingung:	Der *Robot* erreicht sein vorgegebenes Ziel
Nachbedingung bei erfolgreicher Ausführung:	Der *Robot* erreicht seine Ladestation
Beteiligte Nutzer:	*Robot*
Auslösendes Ereignis:	*Robot* erreicht *Destination* (Use-Case)

Jedem Robot wird, laut Aufgabenstellung, eine feste Ladestation zugewiesen. Der Robot fährt vollständig autonom diese Ladestation an. Daher ist keine Kommunikation mit dem Server notwendig.

Szenario für den Standardablauf (Erfolg)

Schritt	Nutzer	Beschreibung der Aktivität
1	Robot	*Robot* erreicht Ziel
2	Robot	*Robot* fährt zur Ladestation
3	Robot	*Robot* erreicht Ladestation und lädt sich auf
4	Robot	*Robot* erhält neues Ziel und fährt dorthin

3.5 Beschreibung zu Use Case 4: Choose Robot

Charakterisierende Informationen

Übergeordneter elementarer Geschäftsprozess:	Choose Robot
Ziel des Use Cases:	passenden *Robot* für die durch den *Server* gesendete *Destination* auswählen
Umgebende Systemgrenze:	*Robot* und *Server*
Vorbedingung:	*Server* hat eine neue *Destination* bekommen
Nachbedingung bei erfolgreicher Ausführung:	dem ausgewählten *Robot* wird die Task übertragen
Beteiligte Nutzer:	*Robot* und *Server*
Auslösendes Ereignis:	*Server* empfängt neue *Destination*

Szenario für den Standardablauf (Erfolg)

Schritt	Nutzer	Beschreibung der Aktivität
1	Server	*Server* sendet Anfragen an alle *Robots*
2	Robot	*Robots* empfangen Anfrage und führen dann den Use-Case „read Sensor“ aus
3	Robot	*Robots* senden ihre ermittelten Sensorwerte an den *Server*
4	Server	*Server* empfängt Daten und wählt den am Besten geeigneten *Robot* aus

3.6 Beschreibung zu Use Case 5: Assign Task

Charakterisierende Informationen

Übergeordneter elementarer Geschäftsprozess:	Choose Robot
Ziel des Use Cases:	*Robot* den Task (die *Destination*) zuweisen
Umgebende Systemgrenze:	*Server* und *Robot*
Vorbedingung:	„choose Robot“ hat am Besten geeigneten *Robot* gefunden und ausgewählt
Nachbedingung bei erfolgreicher Ausführung:	ausgewähler *Robot* steuert die *Destination* an
Beteiligte Nutzer:	*Server* und *Robot*
Auslösendes Ereignis:	im Use-Case „choose Robot“ wurde passender *Robot* ausgewählt

Szenario für den Standardablauf (Erfolg)

Schritt	Nutzer	Beschreibung der Aktivität
1	Server	*Server* überträgt ausgewähltem *Robot* den Task

4 Produktumgebung

4.1 Systemumgebung

Im nachfolgenden Abschnitt werden die bekannten Komponenten des Systems und die dazugehörigen Schnittstellen beschrieben. Grundsätzlich besteht das System aus mindestens einem *Robot*, hierfür geeigneten *Chargern* und einem zentralen *Server*.

4.1.1 Hardwareumgebung

Server

Es existiert ein zentraler *Server*. Dieser *Server* verfügt über ausreichende Ressourcen und ist in der Lage direkt mit den Transportvehikeln zu kommunizieren.

IServerCore

Der *Server* verfügt über einen zentralen Hauptprozessor, über den auf alle weiteren Komponenten zugegriffen werden kann. Dies wird der Einfachheit halber angenommen.

IServerMessageHandler

Mit der Kommunikationseinheit *IServerMessageHandler* kann der *Server* Nachrichten direkt an einen *Robot* senden. Der Einfachheit wegen wird angenommen, die dahinterstehende Infrastruktur zur Übertragung von Nachrichten ist stets stabil.

Robot

Das Transportvehikel *Robot* erfüllt Grundfunktionalitäten wie das Anfahren von Zielen, das Umfahren von Hindernissen und das Laden des Akkumulators. Nachfolgend werden die zentralen Hardwarekomponenten des *Robots* beschrieben.

IRobotCore

Der *Robot* verfügt über einen zentralen Hauptprozessor, über den auf alle weiteren Komponenten zugegriffen werden kann. Dies wird der Einfachheit halber angenommen.

IRobotMessageHandler

Mit der Kommunikationseinheit *IRobotMessageHandler* kann der *Robot* Nachrichten, die vom zentralen *Server* für das Vehikel abgesetzt wurden, direkt empfangen.

IRobotEngine

Als Antrieb nutzt das Transportvehikel einen omnidirektionalen Antrieb mit 3 Motoren, über welchen es sich vorwärts, nach rechts, nach links oder durch Drehen um die eigene Achse bewegen lässt. Der Antrieb ermöglicht eine Fahrt in verschiedenen Geschwindigkeiten, wobei eine nicht näher spezifizierte Grenze nicht überschritten werden kann.

IBattery

Jeder *Robot* verfügt über einen Akkumulator, der zur Energieversorgung dient. Eine ausreichende Ladung des Akkumulators ist deshalb zum Betrieb des *Robots* unbedingt erforderlich. Der Akkumulator hat eine maximale Ladekapazität und kann über einen *Charger* geladen werden. Die genaue Beschaffenheit des Akkumulators ist nicht bekannt.

INorthStar

Die Komponente *INorthStar* kann ähnlich einem GPS-Modul die Position des *Robots* feststellen. Hierbei ist die Komponente in der Lage, die Koordinaten der Position in einem zweidimensionalen Koodinatensystem und die aktuelle Ausrichtung des Vehikels zu ermitteln.

IRSensorDistance

Das Transportvehikel verfügt über 9 Infarotdistanzsensoren, die an der kreisförmigen Außenwand des Vehikels im Abstand von jeweils 40 angeordnet sind. Über sie ist die Feststellung der Entfernung des Vehikels von allen bewegten und unbewegten Objekten in der Umgebung möglich.

IBumper

Bei einer Kollision des *Robots* mit einem anderen Objekt wird dieses Ereignis über die Sensorik einer Kollisionserkennung festgestellt, worauf weitere Schritte eingeleitet werden können.

Charger

Es existieren Ladestationen, über die sich die *IBattery* der *Robots* vollständig laden lässt. Eine Ladestation ist dabei genau einem *Robot* zugeordnet. Zum Laden muss ein *Robot* seine Ladestation anfahren, worauf die Ladung sofort beginnt. Jede Ladestation verfügt über eine genaue Position.

4.1.2 Softwareumgebung

Server

Auf dem *Server* läuft, da nicht anders angegeben, ein Standard-Betriebssystem. Darauf läuft eine Laufzeitumgebung, die die benötigten Methoden zur Kommunikation mit den Robots bereitstellt. Die wichtigste (und einzige bisher spezifizierte) Methode, ist die CallRobot() Methode um einen *Robot* anzurufen und einen Datenaustausch herzustellen.

IServerMessageHandler

Der *IServerMessageHandler* ist die Komponente, die ausgehende Informationen des *Servers* für einen *Robot* versendet. Das Verhalten und die enthaltenen Methoden sind dabei allerdings nicht näher spezifiziert.

Robot

Nachfolgend werden die zentralen Softwareschnittstellen des *Robots* beschrieben.

IRobotCore

Auf der zentralen Recheneinheit des *Robots*, dem *IRobotCore*, läuft eine JavaRuntimeEnvironment, in der sich die gesamte Steuerung und die Verwendung der Komponenten und Schnittstellen abspielt.

IRSensorDistance

Die *IRSensorDistance* Komponente stellt drei Methoden bereit, mit denen die Distanz und der Winkel des entdeckten Objekts und der an der Entdeckung beteiligte Infrarotsensor erfasst und in Arrays gespeichert werden.

IDistanceSensor

Die besagten Arrays kann der *IDistanceSensor* auslesen. Er hat dazu ebenfalls drei Methoden. getIRDistances() gibt ein Array mit allen erfassten Objekten zurück, getIRDistancesInRange() gibt ebenfalls ein Array zurück, das allerdings nur die Objekte in einem gewissen Abstand enthält und getNearestIRDistances() gibt nur das nächste Objekt zurück.

INorthStar

Die Komponente *INorthStar* ist für die Positionierung zuständig und greift dabei auf ein Device zur Standortbestimmung zurück. Sie hat zwei Methoden. Eine liest die aktuelle Position aus, die andere die aktuelle Ausrichtung. Die Position besteht dabei aus zwei float-Werten, einer x- und einer y-Koordinate.

IRobotMessageHandler

Der *IRobotMessageHandler* ist die Komponente, die eintreffende Anrufe des *Servers* verarbeitet. Das Verhalten und die enthaltenen Methoden sind dabei allerdings nicht näher spezifiziert. So kann man davon ausgehen, dass jegliche Methoden der anderen Komponenten über den *IRobotCore* aufgerufen und die Ergebnisse an die *IMessageHandler* Komponente übergeben werden können, damit diese die Informationen auf Anfrage des *Servers* beschaffen und übermitteln kann.

IBumperHandler

Die *IBumperHandler* ist die Komponente zum Umgang mit Zusammenstößen. Die Kollisionserkennung *IBumper* registriert einen Aufprall und die *IBumperHandler* Komponente stellt zwei Methoden zum Umgang mit dem Aufprall bereit.

IDrive

Die Bewegungssteuerung des Robots heißt *IDrive*. Sie stellt vier Methoden bereit. Die Methoden driveToPosition() und driveToPositionCautiously() erwarten beide eine Position, eine Geschwindigkeit und einen ArrivalHandler, der eine Methode zur Meldung aufruft, wenn der *Robot* am Ziel angekommen ist. Die Methoden sind beide dafür da, ein gegebenes Ziel anzufahren, wobei bei der Zweiten die Höchstgeschwindigkeit geringer ist. Die höchstgeschwindigkeit für Cautiously, Regular und Fast Methoden stellt die *IDrive* Komponente bereit, wobei Cautiously $\leq$ Regular $\leq$ Fast gilt.
Die anderen beiden Methoden, drive() und driveCautiously() unterscheiden sich ebenfalls nur in der Höchstgeschwindigkeit und ermöglichen ein manuelles Fahren. Dabei erwarten sie die Drehung des *Robots* als float-Wert sowie die Vorwärts- und die Seitwärtsgeschwindigkeit, ebenfalls als float-Wert.

IBattery

Bei *IBattery* handelt es sich um die Akkusteuerung des *Robots*. Sie stellt zwei Methoden bereit. getBatteryLevel() gibt den aktuellen Akkuladestand als float-Wert zurück, getChargingPosition() gibt die Position des zugeordneten *Chargers* zurück.

4.1.3 Ressourcenübersicht

In Abbildung 7 werden die in 4.1.1 und 4.1.2 beschriebenen Komponenten und Schnittstellen im Verteilungsdiagramm dargestellt.

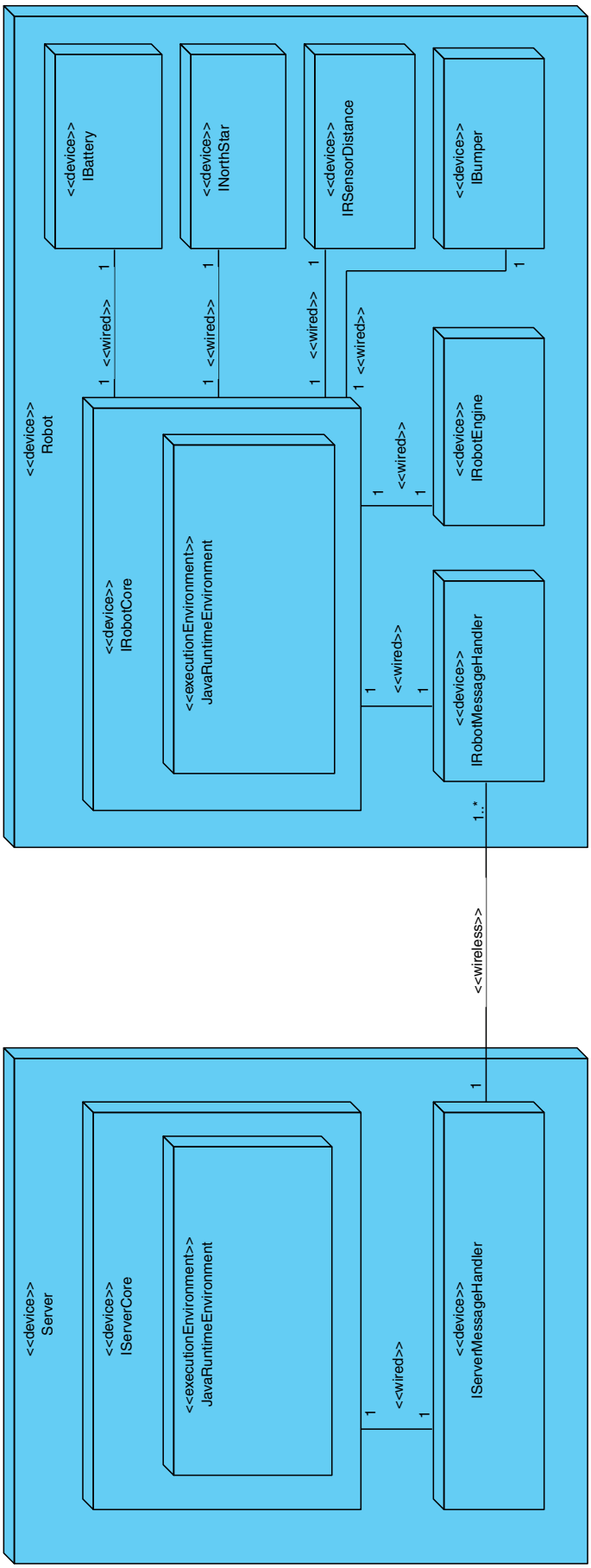

Abbildung 7: Verteilungsdiagramm