Realisierung einer universellen Steuerungsoberfläche für mobile Serviceroboter

Baccalauriat
am Arbeitsbereich TAMS

Studiengang Informatik
der
Universität Hamburg

Andre Stroh
Denis Klimentjew

Betreuung: Prof. Dr. Jianwei Zhang

Abgabetermin: 12. Mai 2006

Inhaltsverzeichnis

1 Einleitung

Dieser Arbeit entstand während des Projekts „Serviceroboter“ an der Universität Hamburg und beschäftigt sich mit der Realisierung einer graphischen Oberfläche zur gleichzeitigen Steuerung mehrerer Roboter. In diesem Kapitel werden die Beweggründe dieses Projekts, heutiger Stand der Entwicklung der Robotersteuerungen, Ziele und Aufbau dieser Arbeit wiedergegeben.

1.1 Motivation

Die Roboter-Technologie schreitet immer weiter fort und findet ein immer größeres Einsatzspektrum. Roboter sind besonderes in der Industrie, Landwirtschaft und Medizin weit verbreitet. Aber auch in den Bereichen, die für den Menschen schädlich oder sogar gefährlich sind, werden verstärkt Roboter eingesetzt. Menschen wundern sich schon lange nicht mehr, dass die Roboter langsam Produktionshallen verlassen und Schritt für Schritt, ein festes Bestandteil unseres Lebens werden.

Ihre Eigenschaften sind Zuverlässigkeit, Flexibilität und Robustheit. Schon länger erledigen sie ihre Aufgaben teil-/ wenn nicht sogar vollautonom. Es existieren sogar halbautomatische Geräte, wie die Defibrillatoren [Philips], die dem ungeschulten Helfer im Notfall Entscheidungen abnehmen und durch Sprachanweisungen während den gesamten Erste-Hilfe-Maßnahmen begleiten.

Doch die Visionen der Roboterentwickler sind schon viel weiter, die Roboter helfen im Haushalt, bei der Pflege kranker und älterer Menschen, passen auf die Kinder auf, und so weiter. Die Entwicklung ging schon in den 80er Jahren dazu über, Roboter für menschliche Bedürfnisse außerhalb der Industrie, zu entwerfen. Seit mehreren Jahren arbeiten die Forscher in Deutschland, wie auch auf der ganzen Welt, an der Entwicklung des Serviceroboters.

Schon 1994 wurde an dem Fraunhofer-Institut für Produktionstechnik und Automatisierung (IPA) die Definition des Serviceroboters festgelegt:

„Ein Serviceroboter ist eine frei programmierbare Bewegungseinrichtung, die teil- oder vollautomatisch Dienstleistungen verrichtet. Dienstleistungen sind dabei Tätigkeiten, die nicht der direkten industriellen Erzeugung von Sachgütern, sondern der Verrichtung von Leistungen für Menschen und Einrichtungen dienen.“

Mittlerweile sind nicht nur Prototypen der Serviceroboter in der Entwicklung, sondern sogar erste fertige Produkte auf dem Markt erhältlich. Diese Entwicklung wird unterschiedlich von Menschen wahrgenommen, während in Japan die Technikbegeisterung

enorm hoch ist, herrscht in Europa eher Skepsis in Bezug auf neue Technologien [Heute]. Dennoch ist die Tatsache, dass die Bevölkerung in Europa immer älter wird und dass das Bevölkerungswachstum zu 95% in Entwicklungsländern [WeltBev] stattfindet, deutet an, dass die Menschheit eventuell ohne solche Entwicklungen, wie die Haushaltsroboter und Pflegeroboter, in der Zukunft nicht mehr auskommt.

1.2 Stand der Technik

Steuerung von mobilen autonomen Roboter ist ein wichtiges Thema, das zurzeit nur wenig allgemeine Beachtung bekommt, da jeder Entwickler, jedes Entwicklungsteam, eigene Steuerung entwirft und einem ausgearbeiteten Szenario anpasst.

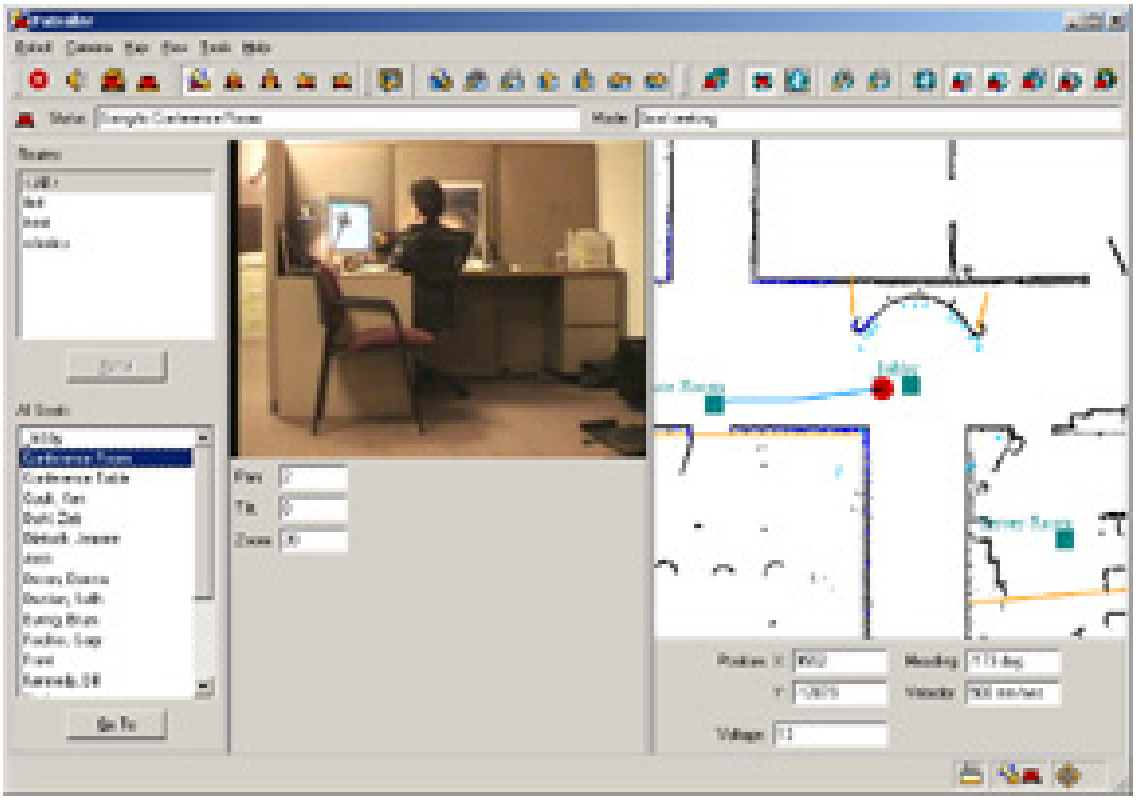


Abbildung 1: Robot Control GUI (Steuerungsbedienoberfläche) der ActivMedia Robotics, mit der Darstellung der Karte, der Steuerelemente und der, von Roboterkameras aufgenommene, Bilder. Der Roboter befindet sich in einer räumlichen Umgebung und soll sich sicher, ohne Kollisionen, zum Zielpunkt bewegen.

Da die meiste mobile Roboter monotone Aufgaben bewältigen, wird keine graphische Steuerungsoberfläche benötigt. Es wird, wie zum Beispiel bei CNC gesteuerten Maschinen, ein Programm geladen und ausgeführt. Bei Haushaltsrobotern kommen Algorithmen zum Einsatz, die das Vorgehen eines Roboters festlegen. Die wichtigsten Aspekte, der beiden oben genannten Steuerungen, sind Einsatzbereicherkennung und Kollisionsvermeidung. Der permanente Eingriff in die Steuerung findet nicht statt. Es wird höchstens die Möglichkeit der Fernsteuerung, wie zum Beispiel beim AIBO® (AIBO® ist ein eingetragener Name der Sony Corporation), dem Benutzer zur Verfügung ge-

stellt. Zum Zwecke der Orientierung werden auf dem Bildschirm Sensordaten ausgegeben, im Fall von AIBO®, die von Kameras aufgenommenen Bilder der Umgebung. Des Weiteren werden die Steuerungsfunktionen, um den Roboter in Bewegung zu setzen oder ein Ziel zu definieren, bereitgestellt.

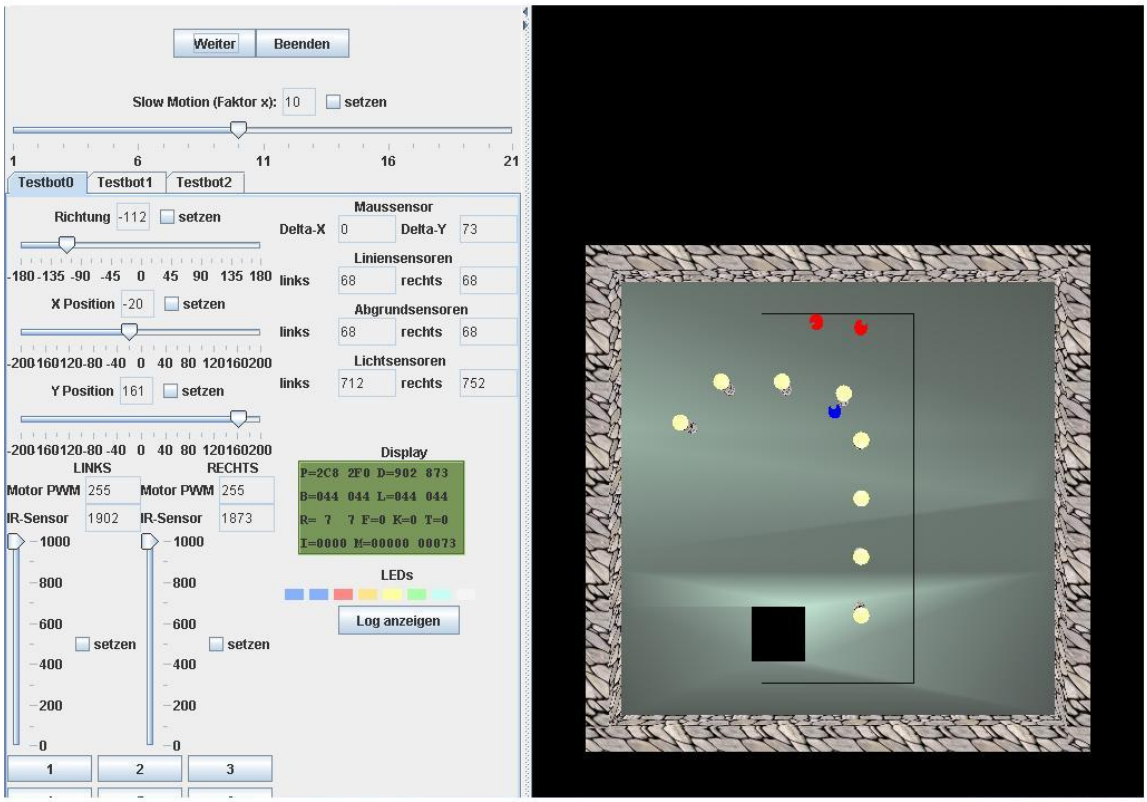


Abbildung 2: Graphischen Steuerungsoberfläche des c`t-Bot Roboters. Auf der rechten Seite befindet sich die Karte der räumlichen Umgebung mit der Darstellung der Roboter. Auf der linken Seite befinden sich Steuerungselemente, jeder Roboter erhält einen eigenen Tab, so dass die Sensoren abgelesen, die Position gesetzt und Geschwindigkeit manipuliert werden kann.

Die Einsatzmöglichkeiten für einen Serviceroboter sind vielseitig. Um einen Serviceroboter an die unterschiedlichen Aufgaben anzupassen, muss die Steuerung des Roboters dynamisch gestaltbar sein. Da die Neukonfiguration des Roboters auch für wenig geschultes Roboterbedienpersonal einfach und intuitiv sein muss, zeigt sich das Problem, dem Roboter eine neue Aufgabe zu erteilen, ohne einen Experten zu benötigen.

So kamen viele Serviceroboterentwicklerteams auf die Idee der graphischen Steuerungsoberfläche, um die Steuerung plausibel und intuitiv zu gestalten. Ein Beispiel dafür ist die Robot Control GUI von ActivMedia Robotics, die nicht nur Steuerungselemente enthält, sondern auch die Umgebungskarte, die ausgelesenen Sensorenwerte und die von den Roboterkameras aufgenommenen Bilder, anzeigt. Abbildung 1 stellt die Robot Control GUI von der ActivMedia Robotics dar.

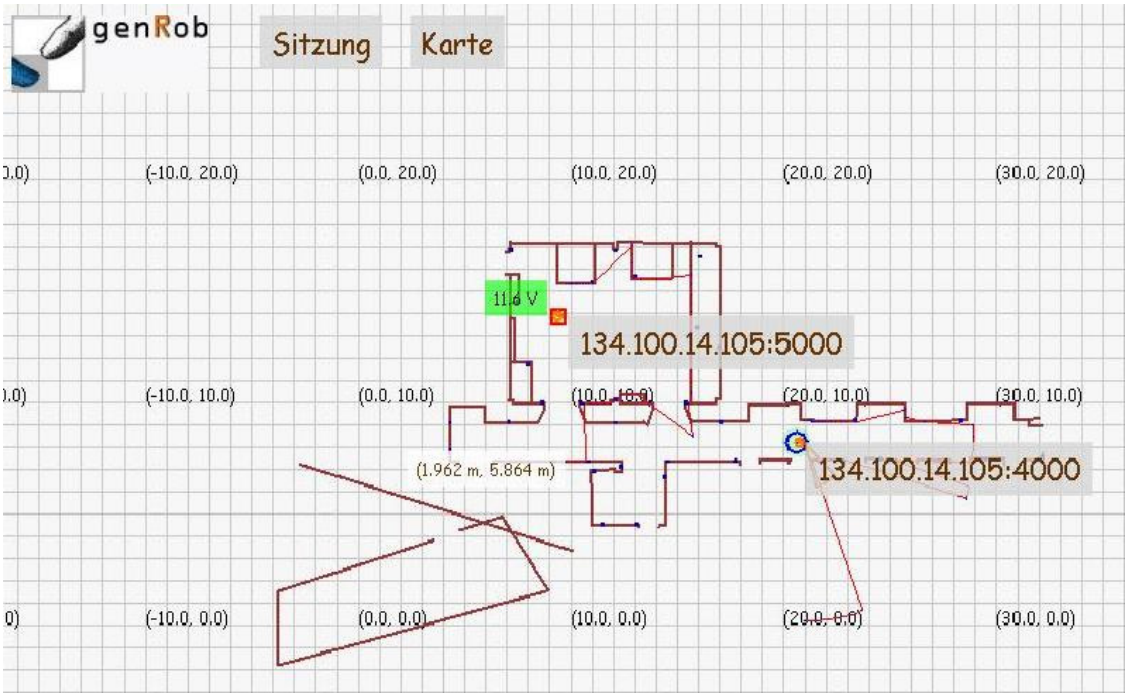


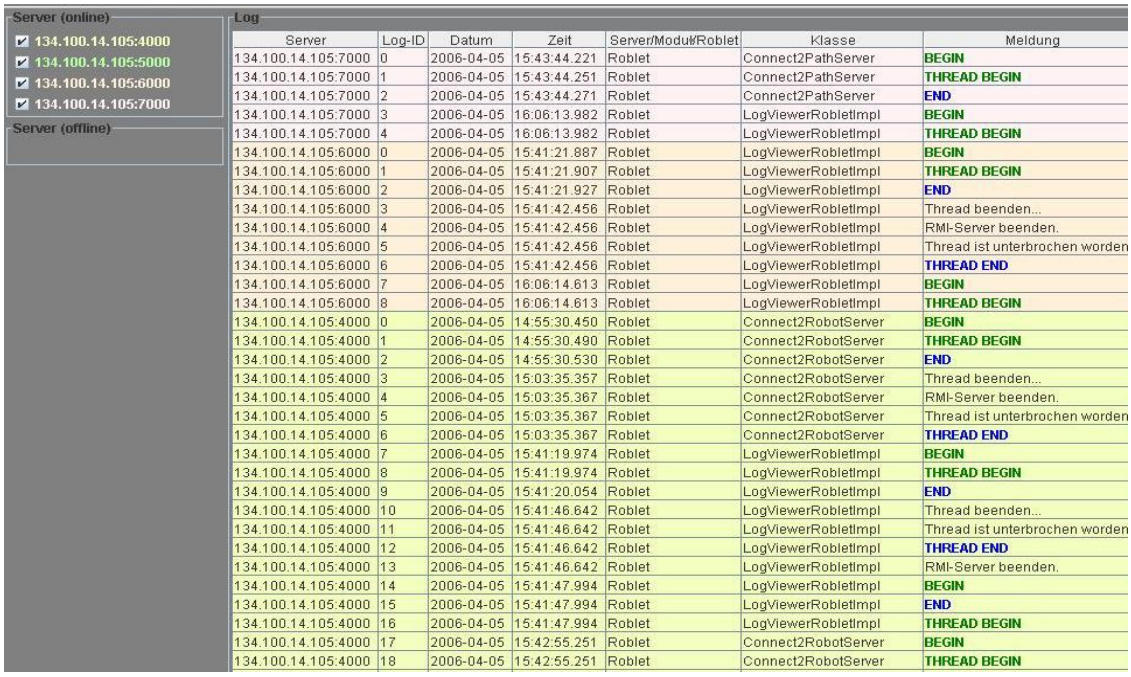
Abbildung 3: Eine in genView dargestellte Umgebungskarte mit zwei Robotern. Die Roboter werden als geometrische Figuren, die die Umrisse der Hardware repräsentieren, angezeigt. Die an den Roboter eingeblendeten IP-Adressen mit den Ports dienen zur Identifikation der Roboter. Durch Anklicken dieser IP-Adressen und werden Menüs geöffnet, die einige Steuerungsfunktionen der Roboter beinhalten. Des Weiteren werden der Batteriestatus und die Laserscans der beiden Roboter angezeigt. Durch zwei Menüpunkte im oberen Bereich des Fensters können Manipulationen an der Karte vorgenommen und eine neue Sitzung gestartet werden.

Als weiteres Beispiel dient die graphische Steuerung und Simulation des c`t-Bot [c't] Roboters. Mit dieser graphischen Oberfläche können mehrere Roboter parallel gesteuert werden. Die Aktivitäten der Roboter werden in der Karte dargestellt. Die linke Hälfte der graphischen Oberfläche wird zur Anzeige und zur Manipulation der Steuerelemente genutzt. Jeder in die GUI eingebundener Roboter erhält seinen eigenen Tab. In demselben Tab sind die Display-Anzeige und die LED's des Roboters mit eingebunden. Dadurch werden sämtliche Statusinformationen angezeigt. Die rechte Hälfte des Fensters stellt einen dreidimensionalen Raum dar, in dem die Bewegungen der Roboter angezeigt werden. In Abbildung 2 wird einen die graphische Oberfläche des c`t-Bot Projektes gezeigt.

Für bestimmte Aufgabengebiete, in denen Serviceroboter eingesetzt werden können, sind graphische Steuerungsoberflächen besonderes geeignet, so liegt die Idee nahe, eine universelle Steuerungsoberfläche zu entwerfen. Es wurde von mehreren Entwicklerteams erkannt und teilweise umgesetzt, wie zum Beispiel das Rocon-Projekt, eine Systemumgebung zur Steuerung und 3D-Visualisierung [Rocon], an der Universität Ulm.

Eine graphische Steuerungsoberfläche bietet viele Vorteile. Zum Beispiel hat das Roboterbedienpersonal alle nötigen Funktionen im Blickfeld. Die Aufgaben können an

den Serviceroboter plausibler und schneller gestellt werden, es ist leichter nachzuprüfen, dass der Roboter seine Umgebung richtig wahrnimmt, wo er sich zurzeit befindet, oder ob alle wichtigen Werte in Ordnung sind. Durch den Einsatz unterschiedlicher visueller Gestaltungsmittel wird ein mächtiges Werkzeug erschaffen, das den Entwicklern die Möglichkeit gibt, die gesamte Steuerung an die Wünsche und Bedürfnisse der Bediener anzupassen.



Server	Log-ID	Datum	Zeit	Server/Modul/Roblet	Klasse	Meldung
134.100.14.105:7000	0	2006-04-05	15:43:44.221	Roblet	Connect2PathServer	BEGIN
134.100.14.105:7000	1	2006-04-05	15:43:44.251	Roblet	Connect2PathServer	THREAD BEGIN
134.100.14.105:7000	2	2006-04-05	15:43:44.271	Roblet	Connect2PathServer	END
134.100.14.105:7000	3	2006-04-05	16:06:13.982	Roblet	LogViewerRobletImpl	BEGIN
134.100.14.105:7000	4	2006-04-05	16:06:13.982	Roblet	LogViewerRobletImpl	THREAD BEGIN
134.100.14.105:6000	0	2006-04-05	15:41:21.887	Roblet	LogViewerRobletImpl	BEGIN
134.100.14.105:6000	1	2006-04-05	15:41:21.907	Roblet	LogViewerRobletImpl	THREAD BEGIN
134.100.14.105:6000	2	2006-04-05	15:41:21.927	Roblet	LogViewerRobletImpl	END
134.100.14.105:6000	3	2006-04-05	15:41:42.456	Roblet	LogViewerRobletImpl	Thread beenden...
134.100.14.105:6000	4	2006-04-05	15:41:42.456	Roblet	LogViewerRobletImpl	RMI-Server beenden.
134.100.14.105:6000	5	2006-04-05	15:41:42.456	Roblet	LogViewerRobletImpl	Thread ist unterbrochen worden
134.100.14.105:6000	6	2006-04-05	15:41:42.456	Roblet	LogViewerRobletImpl	THREAD END
134.100.14.105:6000	7	2006-04-05	16:06:14.613	Roblet	LogViewerRobletImpl	BEGIN
134.100.14.105:6000	8	2006-04-05	16:06:14.613	Roblet	LogViewerRobletImpl	THREAD BEGIN
134.100.14.105:4000	0	2006-04-05	14:55:30.450	Roblet	Connect2RobotServer	BEGIN
134.100.14.105:4000	1	2006-04-05	14:55:30.490	Roblet	Connect2RobotServer	THREAD BEGIN
134.100.14.105:4000	2	2006-04-05	14:55:30.530	Roblet	Connect2RobotServer	END
134.100.14.105:4000	3	2006-04-05	15:03:35.357	Roblet	Connect2RobotServer	Thread beenden...
134.100.14.105:4000	4	2006-04-05	15:03:35.367	Roblet	Connect2RobotServer	RMI-Server beenden.
134.100.14.105:4000	5	2006-04-05	15:03:35.367	Roblet	Connect2RobotServer	Thread ist unterbrochen worden
134.100.14.105:4000	6	2006-04-05	15:03:35.367	Roblet	Connect2RobotServer	THREAD END
134.100.14.105:4000	7	2006-04-05	15:41:19.974	Roblet	LogViewerRobletImpl	BEGIN
134.100.14.105:4000	8	2006-04-05	15:41:19.974	Roblet	LogViewerRobletImpl	THREAD BEGIN
134.100.14.105:4000	9	2006-04-05	15:41:20.054	Roblet	LogViewerRobletImpl	END
134.100.14.105:4000	10	2006-04-05	15:41:46.642	Roblet	LogViewerRobletImpl	Thread beenden...
134.100.14.105:4000	11	2006-04-05	15:41:46.642	Roblet	LogViewerRobletImpl	Thread ist unterbrochen worden
134.100.14.105:4000	12	2006-04-05	15:41:46.642	Roblet	LogViewerRobletImpl	THREAD END
134.100.14.105:4000	13	2006-04-05	15:41:46.642	Roblet	LogViewerRobletImpl	RMI-Server beenden.
134.100.14.105:4000	14	2006-04-05	15:41:47.994	Roblet	LogViewerRobletImpl	BEGIN
134.100.14.105:4000	15	2006-04-05	15:41:47.994	Roblet	LogViewerRobletImpl	END
134.100.14.105:4000	16	2006-04-05	15:41:47.994	Roblet	LogViewerRobletImpl	THREAD BEGIN
134.100.14.105:4000	17	2006-04-05	15:42:55.251	Roblet	Connect2RobotServer	BEGIN
134.100.14.105:4000	18	2006-04-05	15:42:55.251	Roblet	Connect2RobotServer	THREAD BEGIN

Abbildung 4: logViewer mit aufgelisteten Loginformationen der Roblet®-Server. Im linken Bereich des Fensters können die im Netz laufenden Roblet®-Server ausgewählt werden, zu denen die Loginformationen angezeigt werden sollen. Im rechten Bereich des Fensters werden die Loginformationen der Roblet®-Server angezeigt. Die unterschiedliche Färbung des Hintergrundes im Logbereich dient zu einfacherer Zuordnung der Information an die einzelnen Server.

Durch früher durchgeführte Projekte und Forschungen am Arbeitsbereich standen uns einige Werkzeuge zur Verfügung. Darunter waren auch graphisch realisierte Komponenten, wie genView, das die Karte mit Umrissen der Räume und der Roboter anzeigt und einige Steuerungselemente enthält. Abbildung 3 stellt einen Schnappschuss von genView mit der Karte der Umgebung und zwei Roboter dar.

Des Weiteren stand das Tool logViewer zur Verfügung, das die Log-Einträge der Roblet®-Server sammelt und in Form einer Tabelle einem Benutzer bereitstellt. Abbildung 4 zeigt eine von logView erstellte Tabelle mit den Log-Einträgen.

1.3 Zielsetzung

Diese Arbeit entstand während des Projektes „Serviceroboter“ an dem Department Informatik der Universität Hamburg. Im Vordergrund des Projektes stand die Realisierung einer multimodularen graphischen Steuerung eines Roboters.

Die Realisierung der graphischen Steuerungsoberfläche ist ein Teil des genRob®-Projektes, dessen Ziele wie folgt formuliert sind.

Ziel des genRob®-Projektes ist die Entwicklung von Software zur Vereinfachung der Arbeit mit mobilen Robotern und allgemein mit verteilten Systemen [genRob®].

An dem Arbeitsbereich TAMS der Universität Hamburg wird das Szenario des Roboters weiterentwickelt, der in einer biotechnologischen Laborumgebung eingesetzt wird. Ein Szenario, das seinen Ursprung an der Universität Bielefeld hat.

Die gesamte Architektur basiert auf verteilten Systemen, die über das Netzwerk miteinander gekoppelt werden. Die Benutzung von Java und einer Java Virtual Maschine erlauben einen plattformunabhängigen Einsatz der Roboter.

Durch den Einsatz mehrerer voneinander unabhängig laufender Werkzeuge zur Überwachung und Steuerung des Roboters entstehen einige Nachteile, die die Benutzung des Roboters erschweren. Ein Benutzer muss mehrere Programme starten, viele Fenster werden nebeneinander Angeordnet. Dadurch kann der Benutzer sehr schnell den Überblick verlieren und das Gefühl bekommen überfordert zu sein.

Die Projektteilnehmer wollten die Benutzung des Roboters erleichtern, durch die Entwicklung einer gemeinsamen Benutzungsoberfläche, in die alle wichtigen Steuerungselemente eingebunden werden. Das Hauptkriterium für die Gestaltung der graphischen Steuerungsoberfläche ist die einfache Erweiterbarkeit. Zusätzlich sollte die Oberfläche dynamisch mehreren, am besten allen, möglichen Szenarien angepasst werden können.

Diese Arbeit beschreibt die Steuerung des Roboters und die Anzeige der wichtigsten Messinstrumente. Diese werden in die, von parallel arbeitender Gruppe realisierte, GUI eingebunden. Alle Gruppen haben sich für die Benutzung einer zentralen Kontrollinstanz entschieden, so dass eigene Anwendungen in Form von Plugins eingebunden werden. Die integrierten Plugins bieten durch Services, bestimmte Dienste nach außen an. So können Veränderungen oder Modifikationen jederzeit dynamischen vorgenommen werden, ohne die grundsätzliche Architektur oder andere Dienste zu beeinflussen.

Die Vorteile der graphischen Steuerung sind einleuchtend, wenn die Einsatzgebiete von mobilen Robotern betrachtet werden. Die Roboter werden in Landwirtschaft, Medizin, Weltraum und sogar Unterwasser eingesetzt. Sie haben verschiedene Aufgaben, von

Überwachung von Patienten oder Objekten bis zur Fertigung in der Industrie oder Bedienung komplizierter Anlagen allerart. Das Roboterbedienpersonal, soll schnell

eingearbeitet werden können, mit nur wenigen Schulungen und ohne viel Wissen über die zugrunde liegende Software-, Hardwarearchitektur. Die Steuerung der Roboter soll intuitiv, einfach und bedienfreundlich sein.

Als weiteres Ziel dieses Projektes wird die Steuerung mehrerer Roboter, unter Benutzung der gleichen graphischen Steuerungsoberfläche, gesetzt.

Zum Schluss wird die dynamische Anpassung der graphischen Steuerungsoberfläche an verschiedene Aufgaben multipler Roboter anhand eines Experiments untersucht. Dafür werden die Funktionen der Routenplanung eines Wachroboters realisiert, der in der Lage sein sollte komplexe Objekte zu bewachen.

1.4 Aufbau der Arbeit

Nach dem in der Einleitung Motivation, Stand der Technik und Zielsetzung beschrieben worden sind, wird in diesem Abschnitt ein Überblick über die weitere Teile dieser Arbeit gegeben.

Am Anfang wird auf die Hardware- und Softwareumgebung eingegangen, die den Ausgangspunkt des hier behandelten Projektes bilden. Zusätzlich wird die Entwicklungsumgebung und das Versionsverwaltungsprogramm beschrieben, sowie einige grundlegende Vereinbahrungen zur Kommunikation mit den Server.

Danach werden, unter Einbezug von Ideen und Entwicklungen anderer graphischer Steuerungsoberflächen, die Ziele des Projektes formuliert.

Anschließend wird der gesamte Projektentwurf vorgestellt und anhand eines Diagramms erläutert.

Daran Angelehnt wird die Verteilung einzelner Gruppenaufgaben beschrieben. Da diese Arbeit mit zwei anderen Teilen dieses Projekts, graphische Oberfläche (engl. GUI) und die Darstellung der Karte, eng zusammenhängend ist, wird auf die Aufgaben und die Ergebnisse der beiden Gruppen näher eingegangen.

Mit der Formulierung der Gruppenaufgaben gehen wir zu den bedeutenden Teilen dieser Arbeit über. Zuerst werden eigene Ziele, unter Berücksichtigung der Hard-, Softwareumgebung und schon vorhandenen Komponenten, formuliert.

Aus diesen Zielen wird ein Entwurf erarbeitet. Anschließend wird der Entwurf formuliert und spezifiziert.

Nach dem Abschluss der Spezifikation wird die Implementierung beschrieben. Auf die Darstellung des vollständigen Quellcodes wird verzichtet, stattdessen werden die im-

plementierten Klassen und deren Methoden beschrieben und die Realisierung markanter Details, als Auszüge des Quellcodes, präsentiert.

Die von den Projektteilnehmern entwickelte graphische Steuerungsoberfläche war als ein dynamisches Werkzeug geplant und realisiert worden. Das wird durch ein Experiment am Ende des Hauptteils überprüft. Dafür wird das Szenario eines Wachroboters beschrieben und spezifiziert, die graphische Steuerungsoberfläche wird dem Szenario angepasst, in dem, die für dieses Szenario benötigte Funktionen implementiert und getestet werden. Anschließend werden die Ergebnisse des Experiments ausgewertet.

In dem Fazit dieser Arbeit werden die Erfahrungen dieses Projektes zusammengefasst und analysiert. Dabei wird besonderes auf die Vor- und Nachteile graphischer Steuerungsoberfläche eingegangen.

Zum Abschluss wird ein Ausblick, über mögliche Weiterentwicklungen und Erweiterungen dieses Projektes, gegeben.

2 Hardware- und Softwareumgebung

In diesem Abschnitt wird die Hardware- und die Softwareumgebung des Roboters, so wie sie für diese Arbeit zur Verfügung stand, dargestellt und erläutert.

2.1 Hardwareumgebung

Der zur Verfügung stehende Roboter basiert auf einen MP-L655 der Firma NEOBOTIX (GPS GmbH Stuttgart), einer flexible navigierbaren Plattform, die mit einem Differentialantrieb ausgestattet ist. Die oben genannte Plattform erlaubt den Einsatz in beliebiger Umgebung mit flachem Boden. Der auf dem Roboter eingesetzte Rechner basiert auf einer x86-Architektur, einem Pentium IV-Prozessor. Zur visuellen Erfassung der Umgebung besitzt der Serviceroboter zwei Digitalkameras.

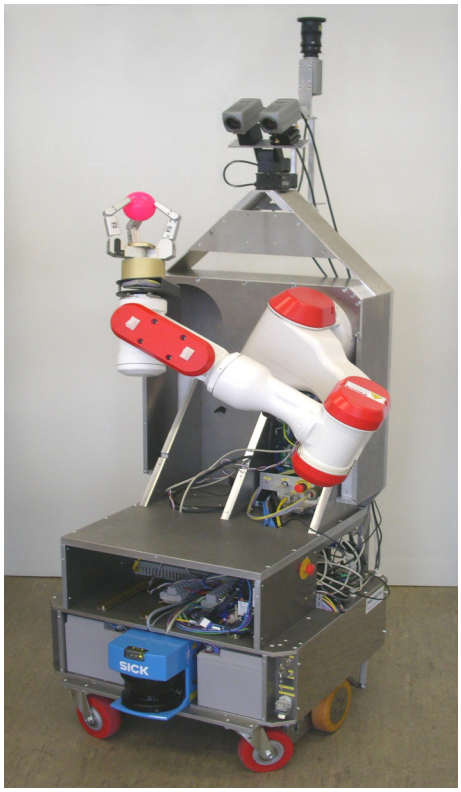


Abbildung 4: Serviceroboter, am Arbeitsbereich TAMS der Uni Hamburg, auf einer flexiblen navigierbaren Plattform, mit Kameras und einen Greiferarm mit einer Dreifingerhand.

Die beiden Kameras vom Typ Sony DFW-VL500 sind auf einer Pan-Tilt-Einheit installiert, die eine flexible Navigation der Kameras ermöglicht. Eine weitere Kamera, DFW-SX900, bildet zusammen mit einem hyperbolidem Spiegel ein Omnivisionsystem, das eine 360° Aufnahme der Roboterumgebung ermöglicht. Diese Aufnahme kann in ein Panoramabild umgerechnet werden und die Raumerfassung erleichtern.

Abbildung 5: Darstellung der Umgebung durch ein Omnivisionssystem, damit werden 360° Bilder der Umgebung erstellt und können zu einem Panoramabild umgerechnet werden.

Abbildung 6 stellt ein umgerechnetes Panoramabild aus dem Omnivisionssystem dar.

Abbildung 6: Ein umgerechnetes Panoramabild aus den Omvivisionssystem des Roboters. Durch die visuelle 360° Panoramaaufnahme ist der Roboter, an der Uni Hamburg Fachbereich Informatik, in der Lage die gesamte Umgebung visuell zu erfassen.

Des Weiteren besitz der Roboter einen Greiferarm mit einer Dreifingerhand [TAMS], unterschiedliche Sensoren, zur Messung der Antriebstemperatur und Ausgabe des Batteriestatus beziehungsweise der Geschwindigkeit. Die Sensorenwerte geben dem Benutzer eine Auskunft über den aktuellen Zustand des Roboters oder können zum starten von Routinen eingesetzt werden. Der Roboter besitzt einen stufenlos geregelten Antrieb, der das Skalieren der Fahrgeschwindigkeit des Roboters erlaubt. Zur Abstandsmessung und Lokalisierung des Roboters, anhand von Reflektor-Marken, stehen zwei Laserscanner, die nach vorne und hinten gerichtet sind, zur Verfügung. Außerdem wird durch die Laserscanner eine rechtzeitige Erkennung und Vermeidung von Kollisionen ermöglicht.

Als Hardwareplattform, auf der Client-Seite und für alle in dieser Arbeit eingesetzten Server, können einer oder mehrere Rechner beliebiger Architektur dienen. Der Einsatz eines heterogenen Rechners ist möglich und ruft keine Probleme hervor, die behandelt werden müssen.

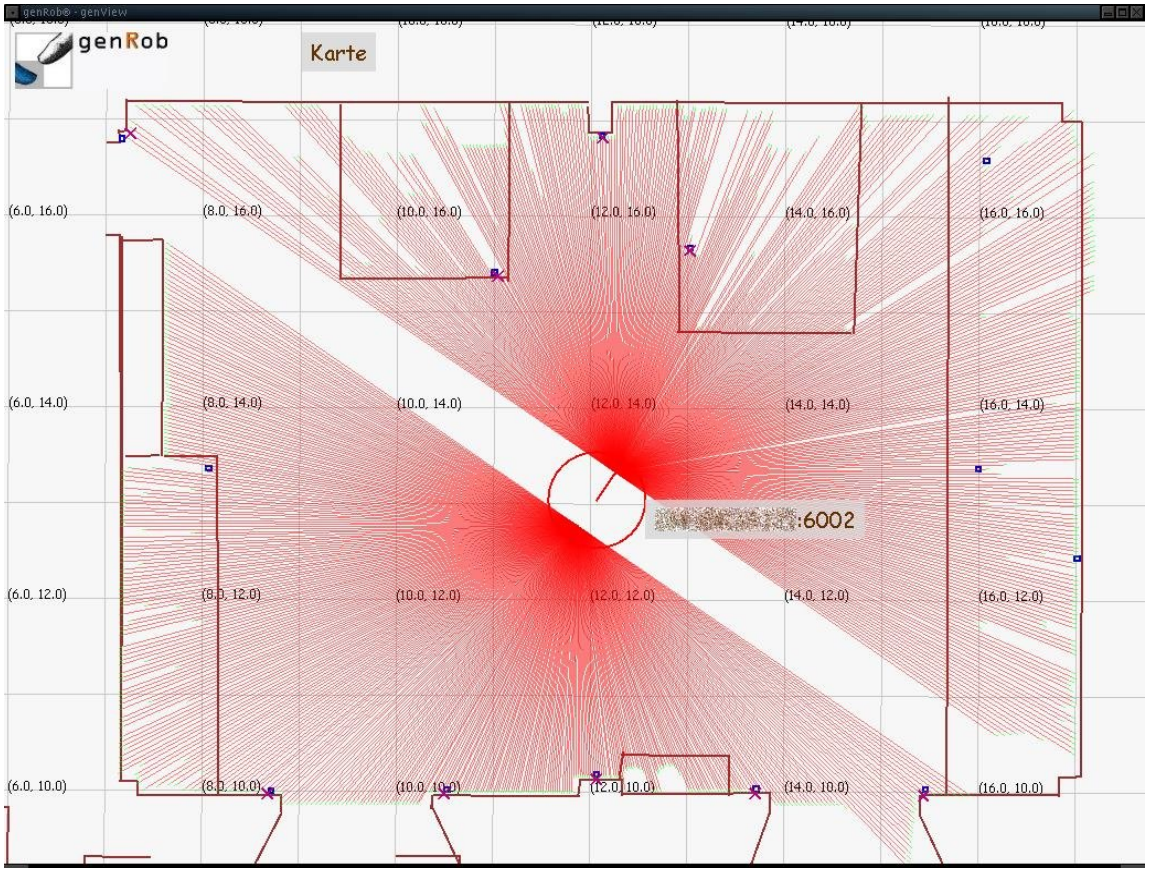


Abbildung 7: Eine graphische Darstellung des Roboters und der Laserscans in einer Linienkarte im Visualisierungswerkzeug genView. Anhand der visuellen Präsentation der Laserscans, ist erkennbar, dass der Roboter mit zwei Laserscannern ausgestattet ist. Diese Ausstattung des Roboters ermöglicht die vollständige Raumerfassung, so dass bei der Bewegung die Hindernisse rechtzeitig erkannt und Kollisionen vermieden werden.

2.2 Softwareumgebung

Für die Steuerung des Serviceroboters, wird die, im genRob®-Projekt entwickelte, Software eingesetzt. Das sind in erster Linie mehrere Roblet®-Server, ein Verzeichnisdienst-Server und verschiedene Applikationen. Abbildung 7 stellt die Systemarchitektur des Serviceroboters dar. Wie aus der Abbildung hervorgeht, basiert der Serviceroboter auf einem verteilten System, das in diesem Fall auf ein Linux-Betriebsystem aufsetzt. Das Betriebssystem läuft seinerseits auf der Hardware eines Rechners. Ein modularer Aufbau der Architektur ermöglicht eine schnelle und unkomplizierte Erweiterung, Wartung oder Ersetzung einzelner Komponente. Der Entwurf und die Realisierung eines verteilten Systems sind komplex, da die Verbindungen zwischen den Server, auf die später noch näher eingegangen wird, hergestellt werden müssen.

Wie schon oben beschrieben besteht eine verteilte Architektur aus einzelnen Komponenten, die im genRob®-Projekt als Server oder Applikationen realisiert werden.

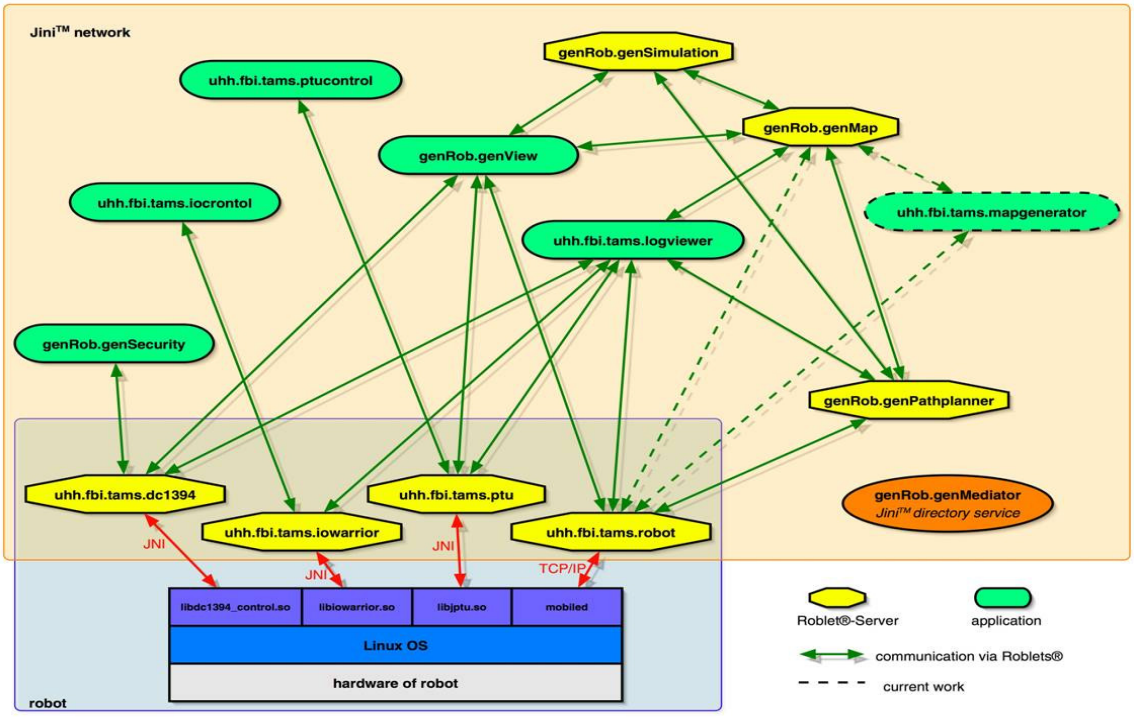


Abbildung 7: Systemarchitektur des Serviceroboters, dabei wird die verteilte Architektur des Systems graphisch verdeutlicht. Dafür wird der modulare Aufbau des Roboters, der Roblet®-Server, der Applikationen und die Kommunikation zwischen den einzelnen Komponenten dargestellt.

Applikationen sind verteilte Client-Anwendungen, die die Dienste der Server in Anspruch nehmen und zum Teil schon in der Einleitung behandelt worden sind. Das Ziel des Projektes „Serviceroboter“ ist die Entwicklung einer multimodalen graphischen Steuerungsoberfläche, in die einzelne Client-Anwendungen eingebunden werden und ihre Dienste nach Außen anbieten. Da das Einbinden der vorhandenen Client-Anwendungen nicht zur Aufgabe des Projektes gehörte, fanden diese Werkzeuge keine Anwendung, sondern dienten als Beispiel einer möglichen Realisierung. So wurden von den Projektteilnehmern neue Client-Anwendungen entworfen und realisiert, auf die im weiteren Verlauf dieser Arbeit noch eingegangen wird.

Server sind Programme, die meistens auf Rechnern im Netz laufen und auf Anfragen von Client-Anwendungen warten. Die in diesem Projekt eingesetzten Server sind Roblet®-Server und besitzen einen modularen Aufbau. Der strukturelle Aufbau der Roblet®-Server wird in der Abbildung 8 graphisch dargestellt.

Ein modularer Aufbau der Server ermöglicht eine dynamische Gestaltung der Funktionalität und dadurch einen universellen Einsatz. Die Funktion des Servers befindet sich im Modul „genRob®-genControl“, dass sich aus mehreren Java-Paketen (engl. Java Packages) zusammensetzt und die Bibliotheken zur Kommunikation bereitstellt.

Dieses Modul setzt auf die Roblet®-Bibliothek auf, in der sich die Basisdefinitionen der Roblet®-Architektur befinden. Des Weiteren können applikationsspezifische Module mit eingebunden werden, die zum Beispiel zur Steuerung eines Roboters dienen.

Durch Einbindung oder Ersatz des entsprechenden Moduls kann die Funktion des Servers erweitert, beziehungsweise modifiziert werden. Die Module beinhalten Einheiten (engl. Units), in denen die Funktionen des Moduls implementiert werden und durch ihre Schnittstellen Zugriff auf die Funktionen erlauben. Bevor die Funktionen der Einheiten benutzt werden können, müssen die entsprechenden Einheiten geladen werden. Das Laden der Module geschieht nach Bedarf zur Laufzeit. Auf den Inhalt der einzelnen Module wird später detailliert eingegangen.

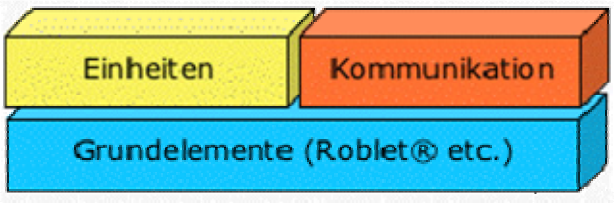


Abbildung 8: Struktureller Aufbau der Roblet®-Server. Die Roblet®-Server basiert auf der Roblet®-Bibliothek und können zusätzliche applikationsspezifische Module enthalten [RobBibl].

Im Folgenden wird ein kurzer Überblick über die Roblet®-Server gegeben, die zu der Realisierung des Projektes genutzt werden.

Zum Test der Steuerungsoberfläche werden zwei Roblet®-Server eingesetzt, die jeweils eines der folgenden Simulationsmodule laden, dass sind „genRob.genSimulation.-ModuleImpl“ und “genRob.genSimulation.hamburgModuleImpl“. Die erste Simulation eines Service-Roboter, stammt von Stanek, den Gründer das genRob®-Projektes. Die zweite Simulation entstand am Arbeitsbereich TAMS der Universität Hamburg und repräsentiert den echten Service-Roboter an diesem Arbeitsbereich.

Der Einsatz der Simulationen erlaubt schnelleres und unkompliziertes Testen der graphischen Steuerungsoberfläche, zusätzlich wird das Risiko vermindert, dass bei eventuell vorhandenen Fehlern in der Implementierung, der echte Roboter beschädigt wird.

Wie oben beschrieben, können die beiden Roblet®-Server der Service-Roboter bei Bedarf, um weitere Funktionen erweitert werden. Dazu werden die Roblet®-Server mit benötigte Module ausgestattet, die beim Start des Servers geladen werden.

Zusätzlich können gleichzeitig mehrere Roblet®-Server gestartet werden, um das Vorhandensein mehrerer Service-Roboter zu modellieren. Um ein reibungslosen Betrieb mehrerer Service-Roboter zu verwirklichen, müssen Server auf verschiedenen Ports oder auf unterschiedlichen Rechnern laufen.

Zur Darstellung einer räumlichen Umgebung in der graphischen Steuerungsoberfläche wird eine Instanz benötigt, die die entsprechende Umgebungskarte liefert. Diese Aufgabe übernimmt ein weiterer Roblet®-Server „genRob.genMap“.

Durch den Einsatz des genRob.genMap-Servers ist nicht nur ein Abfragen, der auf dem Server gespeicherten Umgebungskarte, sondern auch das Manipulieren derselben, möglich. Neben der Darstellung des Grundrisses der Räumlichkeiten, können zusätzlich andere Hindernisse oder Orientierungsmarken in der Karte gespeichert und in einer grafischen Umgebung angezeigt werden.

Auch der Roblet®-Server kann gleichzeitig mehrmals gestartet werden. Nach dem Start werden die Informationen zwischen den Servern ausgetauscht, so dass alle aktiven Server in Besitz einer Kopie einer und derselben Umgebungskarte sind. Damit können Server leicht auf den neuesten Stand gebracht werden, es reicht nämlich nur ein Server zu aktualisieren. Des Weiteren wird durch die Präsenz mehrerer Server im Netz die Sicherheit erhöht, da mehrere Quellen mit redundanter Information vorhanden sind.

Die Vorteile der verteilten Architektur liegen auf der Hand, nicht nur, dass eine neue Karte schnell zur Verfügung gestellt werden kann, sondern der gesamte Server kann problemlos ausgetauscht werden.

Die zu Verfügung stehenden Serviceroboter sind so konzipiert, dass sie bei der Übergabe eines Bewegungsbefehls mit einem Koordinatenpaar, sich auf den übergebenen Zielpunkt ausrichten und losfahren. Diese direkte Ansteuerung eines Koordinatenpunktes führt zu einer möglichen Kollision. Um eventuelle Kollisionen zu Vermeiden, muss ein kollisionsfreier Pfad berechnet werden.

Diese Aufgabe übernimmt der genRob.genPath-Server. Dafür benötigt dieser Server die Start- und Zielkoordinaten, die Karte der Umgebung mit allen Hindernissen und den Radius des Roboters. Diese Parameter werden für die Berechnung des Pfades eingesetzt. Durch den Einsatz unterschiedlicher Algorithmen kann die Berechnung des Pfades beeinflusst werden. Zum Beispiel kann der kürzeste oder der sicherste Weg berechnet werden. Der letztere Algorithmus basiert auf Voronoigraphen, der den größtmöglichen Abstand zu allen Hindernissen garantiert.

Da der Roboter nur ein Koordinatenpaar zur selben Zeit verarbeiten kann, wird zusätzlich eine Komponente benötigt, die die vom Pfadserver erhaltene Koordinatenliste speichert, paarweise an den Roboter weitergibt und solange mit dem nächsten Paar wartet, bis die Bewegung komplett ausgeführt wird. Um diese Aufgabe bewerkstelligen zu können, muss diese Komponente noch implementiert werden.

Da in Abhängigkeit vom Algorithmus die Berechnung des Pfades enorme Rechenleistung beanspruchen kann, aber der Server die meiste Zeit nicht aktiv ist, wurde der genRob.genPath-Server so konzipiert, dass er in der Lage ist, mehrere Roboter, die sich im

im Netz befinden, mit der Koordinatenlisten zur Versorgen. So würde es reichen einen Server auf einem leistungsstarken Rechner nur einmal für alle Roboter laufen zu lassen, was zur Folge hat, dass die Ressourcen geschont werden. Natürlich können, zum Zwecke der Ausfallabsicherung, auch mehrere Pfadserver im gleichen Netz gleichzeitig gestartet werden.

Um die Informationen über das Vorhandensein der Server im Netz und deren Zustände zu verwalten, wird ein zusätzlicher Dienst benötigt. Diese Aufgabe übernimmt der genRob.genMediator-Server. Alle im Netz gestarteten Server melden sich und ihre Dienste beim genRob.genMediator-Server an und übermitteln ihm periodisch eigenen Zustand, so dass der Server permanent über Vorgänge im Netz informiert ist.

Die Kommunikation zwischen den Servern, wird über genRob.genMediator-Server realisiert. Damit wird vermieden, dass Dienstleistungen, die nicht mehr zur Verfügung stehen, anderen Server weiter angeboten werden.

Für das Projekt ist der genRob.genMediator-Server enorm wichtig, weil automatisch die Client-Anwendungen über neue hinzugekommene oder nicht mehr vorhandene Dienste informiert. So werden nur die Dienste angeboten, die auch tatsächlich vorhanden sind und es müssen keine diesbezüglichen Ausnahmefehler behandelt werden.

Die Kommunikation zwischen den einzelnen Komponenten findet durch Roblets® statt. Die grundlegende Idee hinter dem Roblet®-Konzept basiert auf der verteilten Architektur. Ein Benutzer schickt ein Teil einer Anwendung, ein so genanntes Roblet®, auf einen sich im Netz befindlichen Server. Das Roblet® wird auf dem Server ausgeführt. Nach Beendung des Programms wird der Server in den ursprünglichen Zustand zurückgesetzt, was dem nächsten Benutzer oder Server erlaubt ein Roblet® zu dem gewünschten Server zu senden und damit eine Kommunikation aufzubauen. Da immer ein Objekt oder eine Fehlermeldung von Roblet®-Server zurückgegeben wird, stellt es kein Problem, dass falls Rückgabewerte erwünscht sind, diese vor der Beendung des Roblets® zurückzuschicken. Abbildung 8 präsentiert eine Graphik zur Kommunikation zwischen zwei Instanzen auf Roblet®-Basis.

Um eine Instanz Roblet® zu bilden, muss die entsprechende Klasse zwei Interfaces implementieren. Das erste Interface „org.roblet.Roblet“ definiert alle möglichen Methoden, die von der Klasse, die das Roblet® bildet, implementiert werden müssen. Das zweite Interface „java.io.Serializable“ garantiert, dass das verschickte Roblet®, serialisierbar ist. Falls der Benutzer und der Roblet®-Server auf einem Rechner laufen, kann auf das java.io.Serializable verzichtet werden.

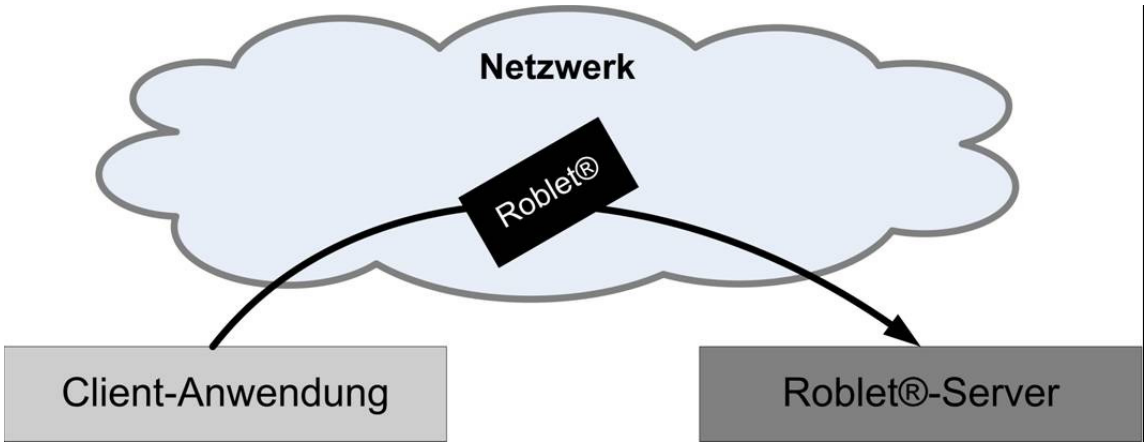


Abbildung 9: Darstellung der Roblet®-Architektur, ein einfaches Szenario. Client-Anwendung sendet ein Roblet®, das aus Code und Daten besteht, an den Roblet®-Server. Der Server führt das Roblet® aus und sendet ein Objekt zurück. Der Server wird im ursprünglichen Zustand hinterlassen.

Nach dem das Roblet® auf dem Server angekommen ist, wird es ausgeführt. Das geschieht in dem der Roblet®-Server die Methode execute(), die in jedem Roblet® implementiert werden muss, aufruft. Wie schon oben beschrieben, wird, wenn kein Fehler auftritt, immer ein Objekt von der execute()-Methode an den Benutzer versandt, was die Rückgabe erforderlicher Parameter erleichtert [Roblet®].

Das Roblet®, wie auch die Roblet®-Server sind in Java implementiert und setzen auf der Java Virtuell Maschine auf, was die Plattformunabhängig garantiert. In Abbildung 9 wird der schematischer Aufbau Roblet®-Server in verschiedenen Schichten dargestellt.

Die Vorteile der Roblet®-Architektur sind eindeutig, Roblets® können für beliebige Anwendungen eingesetzt werden. Die Erweiterung des Systems ist, durch Einbindung weiterer Server, möglich. Zusätzliche Vorteile bringt die Beschaffenheit eines Roblets®, es besteht aus ausführbarem Code und Daten. Es muss also nicht erst eine Verbindung aufgebaut werden um anschließend die Daten zu versenden, sondern beide Schritte werden gleichzeitig ausgeführt. Dadurch werden Server entlastet und die Anzahl der ausgetauschten Nachrichten reduziert.

Für einige Aufgaben dieses Projektes wird eine permanente Kommunikation mit den Server benötigt, dabei werden die Roblets® für den Aufbau solcher Verbindungen eingesetzt. Für die dauerhaften Verbindungen kommt RMI, Remote Method Invocation, zum Einsatz. RMI realisiert den Aufruf einer Methode, eines Java-Objekts, das sich auf einem entfernten Rechner oder einer anderen Java Virtuell Maschine befindet. Auf der Client-Seite sieht der Aufruf der entfernten Methode genauso wie der lokale aus, nur die, während der entfernten Rechnerkommunikation, möglichen Ausnahmefehler müssen behandelt werden.

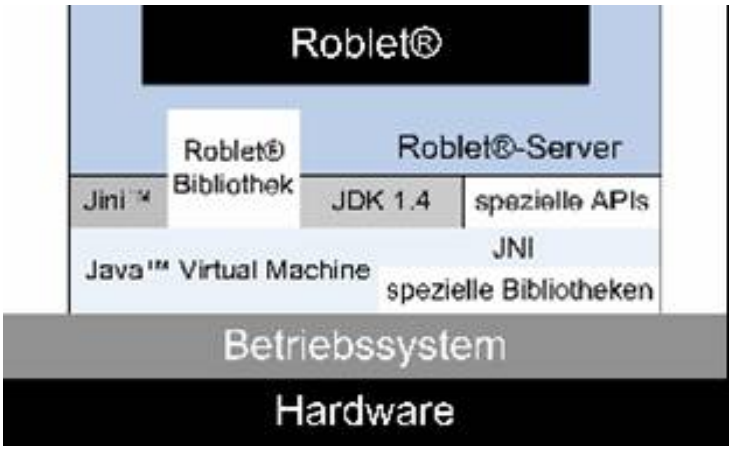


Abbildung 10: Schematischer Aufbau der Roblet®-Server in verschiedenen Schichten. Die Roblet®-Server sind genau so, wie Roblets® in Java implementiert und setzen auf Java Virtual Maschine auf.

Auf der Client-Seite wird von der RMI-Umgebung eine Klasse erzeugt, der so genannte Stub. Diese Klasse simuliert ein entferntes Objekt, das Aufgerufen werden soll. Damit wird ein lokaler Zugriff auf ein, in Wirklichkeit entferntes, Objekt ermöglicht. Die Abbildung 10 stellt den Aufbau der RMI-Verbindung graphisch dar.

Der Stub nimmt den Aufruf entgegen und leitet ihn, entsprechend verändert an den Server, wo das Objekt sich tatsächlich befindet. Zurückgelieferte Informationen werden dann dem Benutzer präsentiert.

Für den Benutzer nicht sichtbar werden mehrere Schritte ausgeführt, die zum Erfolg dieses Aufrufes führen. Zuerst registriert ein Server ein Remote Objekt bei der RMI-Registry unter einem eindeutigen Namen, dieser muss auf der Client-Seite bekannt sein. Der Client fordert eine Referenz auf dieses Objekt bei der RMI-Registry unter diesen Namen an und ruft, anhand dieser Referenz, eine Methode auf. Die Existenz dieser Methode wird von dem Remote Interface garantiert. Der Server liefert an den Benutzer die Rückgabewerte oder, im Falle eines Ausnahmefehlers, eine entsprechende Meldung zurück [Wiki:RMI].

Die RMI-Verbindung in diesem Projekt wird mit Hilfe von Roblets® eingerichtet. Diese hat den Vorteil, dass die Verbindung solange Aktiv bleibt bis sie explizit getrennt wird, somit wird die Ausführung von zeitintensiven Prozessen mit möglicher Verzögerung realisiert.

Als Programmierumgebungen wird Eclipse, in der Version 3.1, eingesetzt. Ab Version 3.0 stellt Eclipse nur den Kern dar, in den die Plugins eingebunden werden. Diese Plugins bilden die eigentliche Funktionalität des Programms. Eclipse und die Plugins sind in Java implementiert. So ist auch die Programmierumgebung plattformunabhängig.

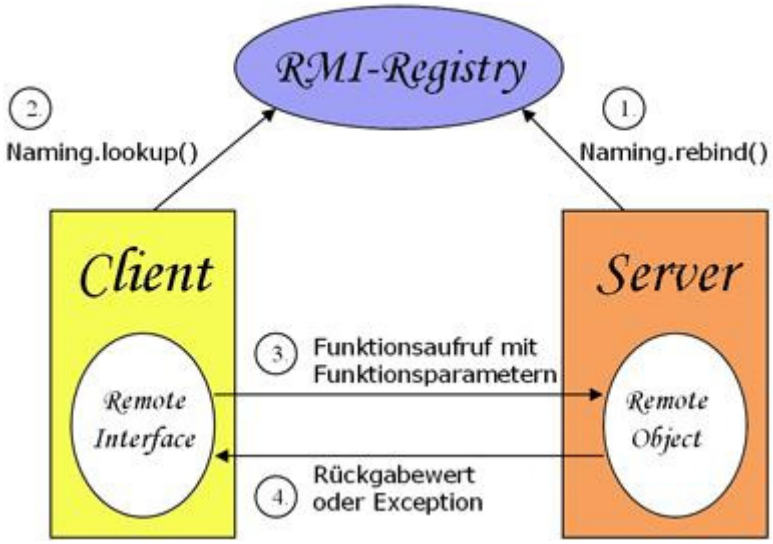


Abbildung 11: Aufbau einer RMI-Verbindung, die in diesem Projekt durch Roblets® aufgebaut wird. Durch RMI wird ein lokaler Zugriff auf ein, in Wirklichkeit entferntes, Objekt ermöglicht.

In erster Linie wird Eclipse zur Programmierung in Java genutzt, doch durch das Einbinden weiterer Plugins wird die Implementierung von C, C++, Perl, PHP, Ruby und Python Programmen ermöglicht. Abbildung 11 zeigt ein Schnappschuss der graphischen Oberfläche von Eclipse 3.1 [Wiki:Eclipse].

Für die Erstellung der graphischen Oberfläche wird in diesem Projekt SWT eingesetzt. SWT ist eine Bibliothek, die Eclipse für Gestaltung graphischer Oberflächen zur Verfügung stellt. Diese Bibliothek wird permanent von IBM gepflegt und weiterentwickelt.

SWT nutzt native graphische Elemente des Betriebsystems. Damit muss die Bibliothek jedem Betriebsystem angepasst werden. Dadurch ist Eclipse, falls SWT eingesetzt wird, nicht mehr plattformunabhängig. Obwohl SWT erst 2001 entwickelt wurde, ist die Bibliothek für mehrere Systeme und Architekturen verfügbar [Wiki:SWT].

Für die Versionsverwaltung wird in diesem Projekt die Open-Source-Software Subversion eingesetzt. Obwohl die Bedienung von Subversion stark an CVS erinnert, ist Subversion eine eigenständige Entwicklung [Wiki:SVN].

Die Ähnlichkeit, zwischen den beiden Entwicklungswerkzeugen, ist nicht verwunderlich, da CVS weit verbreitet ist. So lässt sich vermuten, dass die Entwickler mit besserer Versionsverwaltung und ähnlicher Bedienung nicht nur neue Kunden gewinnen wollten, sondern auch die CVS-Anhänger abwerben.

Die Versionsverwaltung, ermöglicht mehreren Entwicklern am gleichen Projekt zu arbeiten. Dabei werden alle Versionen in einem zentralen Verzeichnis, engl. Repository, das sich meistens auf einem Server im Netz befindet, aufbewahrt. Jeder Entwickler bekommt eine Kopie des Projekts, die er nach Belieben verändern kann. Nach dem, die

Veränderungen vorgenommen worden sind, können sie in das Repository aufgenommen werden. Konflikte, die dadurch entstehen, dass mehrere Entwickler parallel an der gleichen Stelle im Quellcode Veränderungen vorgenommen haben, werden von SVN automatisch erkannt. Die Lösung des entstandenen Konflikts muss aber, genau wie bei CVS, manuell vorgenommen werden. Zugriff auf frühere Versionen ist jederzeit möglich.

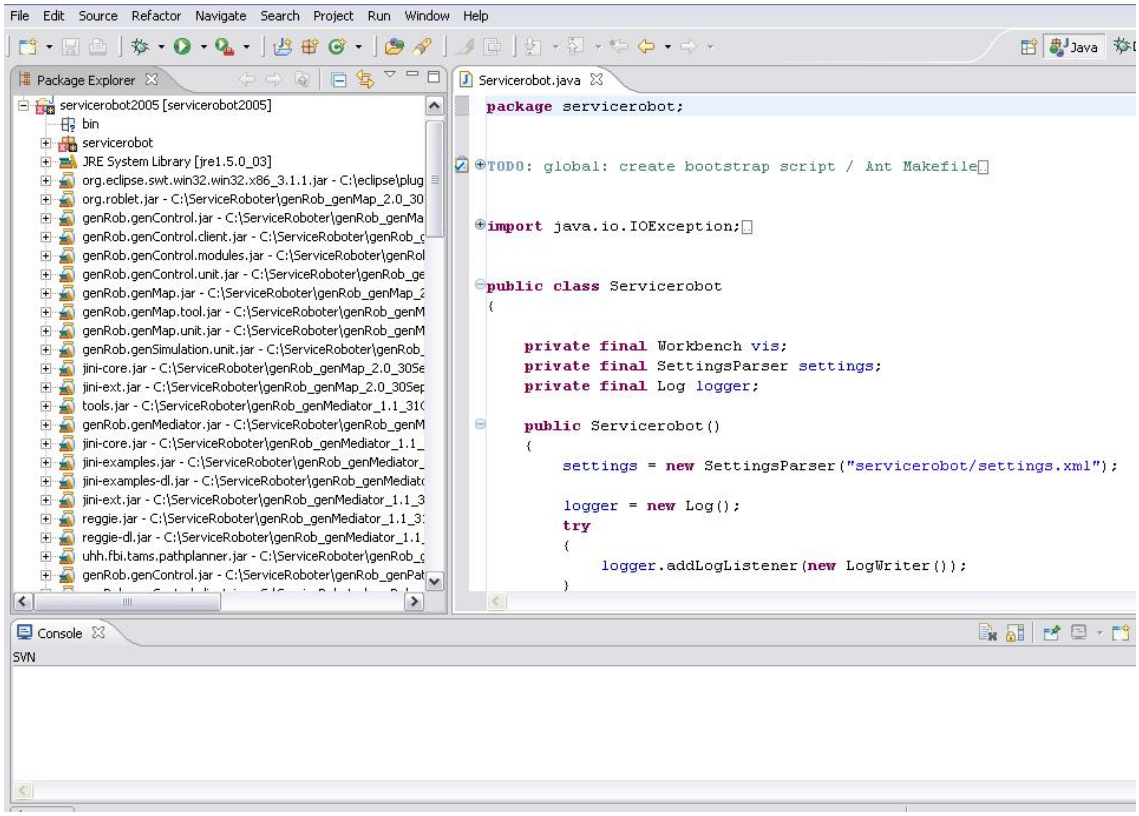


Abbildung 12: Die grafische Oberfläche von Eclipse 3.1 mit einem geöffneten Java-Projekt. Auf der linken Seite der Oberfläche sind die eingebundenen Bibliotheken und die Projekthierarchie, und auf der rechten Seite, ein Editor-Fenster zur Eingabe des Quelltextes, zu sehen. In der Konsole, im unteren Bereich der Oberfläche, werden Fehler und sonstige Information angezeigt.

Um Subvision für dieses Projekt nutzen zu können, wurde die Versionsverwaltung, in Form eines Plugins, in Eclipse eingebunden. Damit wird Synchronisation, Aktualisierung der Daten auf dem Server und auf dem lokalen Arbeitsverzeichnis und so weiter aus Eclipse möglich.

Durch die Verbindung voneinander unabhängigen Komponenten wurde eine stabile, vielfältig einsetzbare Entwicklungsumgebung erschaffen.

2.3 Gesamter Projektentwurf

In diesem Abschnitt wird der gesamte Projektentwurf vorgestellt und erläutert. Da diese Arbeit sich mit der Realisierung einer multiplen graphischen Steuerung der Servicerobter beschäftigt, wird zuerst auf den Aufbau der graphischen Steuerungsoberfläche

eingegangen. Anschließend werden die Plugins und Services beschrieben, die zur Steuerung der Roboter in die graphische Oberfläche eingebunden werden.

Ein Teil der Informationen über den Aufbau des Projektes in diesem Kapitel stammen aus dem Projektbericht von Björn Engelmann und Benjamin Leipold, die sich mit der Implementierung des Rahmenwerks beschäftigten.

Der gesamte Aufbau dieses Projekts basiert auf der Plugin-Architektur, so dass die Oberfläche nur ein Rahmenwerk, engl. Framework, zur Verfügung stellt. Bei Bedarf werden entsprechende Funktionen, durch Plugins, in die Oberfläche eingebunden. Die Plugins erlauben eine flexible und erweiterbare Gestaltung der graphischen Oberfläche, da die eingebundenen Funktionen mit geringem Aufwand verändert oder erweitert werden können. Genauso können weitere Funktionen hinzugefügt oder nicht mehr benötigte Plugins entfernt werden. Für dieses Projekt wurde entschieden, dass die Plugins manuell durch den Benutzer geladen werden, so werden nur die benötigten Plugins eingebunden, was der Umgang mit den Ressourcen verbessert.

Wie schon in vorhergehenden Kapiteln beschrieben, basiert dieses Projekt auf der verteilten Architektur. Daher soll auch der Projektentwurf dieses berücksichtigen. Die, in diesem Projekt entwickelte, graphische Oberfläche ist unabhängig und besteht aus drei Fenstern. Das Hauptfenster für die Anzeige der Karte und der Roboter, unteres Fenster für die Konsole und seitliches Fenster für Plugins und Services. Die Kommunikation nach außen wird durch die Plugins realisiert, die nach Bedarf eingebunden werden. Die Plugins werden geladen und können mitteilen welche weiteren Tabs sie benötigen.

Für die Realisierung des oben dargestellten Models wird eine weitere Instanz benötigt, ein Kontroller, der dafür sorgt, dass die Kommunikation zwischen den Plugins, Services und Server ermöglicht und aufrechterhalten bleibt. Außerdem wird die graphische Steuerungsoberfläche mit Informationen über die im Netz vorhandenen Server und von ihnen bereitgestellte Units versorgt.

Der Kontroller nutz den Aufbau der Softwarearchitektur, die in dem entsprechenden Kapitel beschrieben wurde. Alle Server melden sich bei dem genMediator an und informieren ihn über eigenen Zustand. Deswegen fordert der Kontroller die Informationen, über aktive Server im Netz, nicht von einzelnen Roblet®-Servern sondern vom „genMediator“ an. Dabei wird abgefragt welche Units der entsprechende Server anbietet, diese werden in dem RuntimeModel vermerkt. Da Service-Plugins, die von ihnen benötigte Units in der Konfigurationsdatei festlegen, werden die Informationen, mit auf den Server vorhandenen Units, von dem Kontroller verglichen und Services, für die die angeforderten Units vorhanden sind, in der graphischen Oberfläche dargestellt.

Die Aufgabe des Kontrolleres besteht darin Plugins zu laden und deren Zustand zu speichern. Für dieses Projekt werden zwei Arten von Plugins eingesetzt, GUI-Plugins und Services.

GUI-Plugins werden vom Kontroller eingebunden und erhalten eine Möglichkeit graphische Objekte in der Steuerungsoberfläche darzustellen. Auch die Visualisierung und die Entgegennahme von Benutzerdaten werden durch GUI-Plugins realisiert.

Service-Plugins werden zur Aufbau der Kommunikation mit dem Roblet®-Server und dem Datenaustausch, zwischen dem System und dem Server, eingesetzt. Die Kommunikation zwischen dem Service und dem Server wird durch Roblets® realisiert.

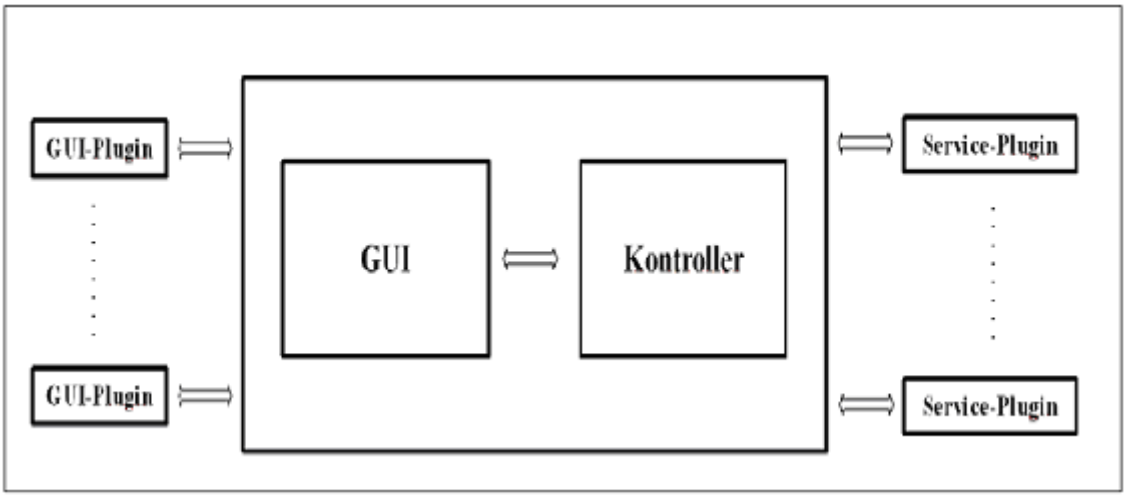


Abbildung 13: Grober Aufbau der Steuerungsoberfläche. Die Plattform besteht aus zwei Komponenten, der graphischen Oberfläche und dem Kontroller. Die beiden Arten von Plugins werden durch den Kontroller eingebunden. Dadurch wird es möglich das System flexibel und erweiterungsfähig zu gestalten.

Da die beiden Arten von Plugins unterschiedliche Aufgaben bewerkstelligen, werden deren Zustände von dem Kontroller auch unterschiedlich gespeichert. Die Zustände der GUI-Plugins werden beim Start der graphischen Oberfläche festgestellt und in RuntimeModel festgehalten. Die Service-Plugins werden nur dann in dem RuntimeModel vermerkt, wenn die Roblet®-Server, mit den von Service-Plugins benötigten Units, im Netz vorhanden sind. Die grobe Struktur des gesamten Systems wird in der Abbildung 14 dargestellt.

Die Kommunikation zwischen der graphischen Oberfläche und dem Kontroller wird ereignisgesteuert realisiert. Dazu werden in beiden Teilen entsprechende Interfaces implementiert. Durch die ereignisgesteuerte Kommunikation werden beide Komponenten auf dem aktuellen Stand gehalten.

Wie in der Abbildung 14 verdeutlich wird, stellt der Kontroller eine zentrale Instanz der graphischen Steuerungsoberfläche dar. Durch die Kommunikation mit dem „genMediator“ werden die Informationen über im Netz vorhandene Server und Units, die sie bereitstellen, festgestellt, in dem Runtime-Modell vermerkt und mit den Units, die Service-Plugins benötigen, verglichen. Die Services, für die Units verfügbar sind, werden in der graphischen Oberfläche dargestellt und können von dem Benutzer gestartet werden.

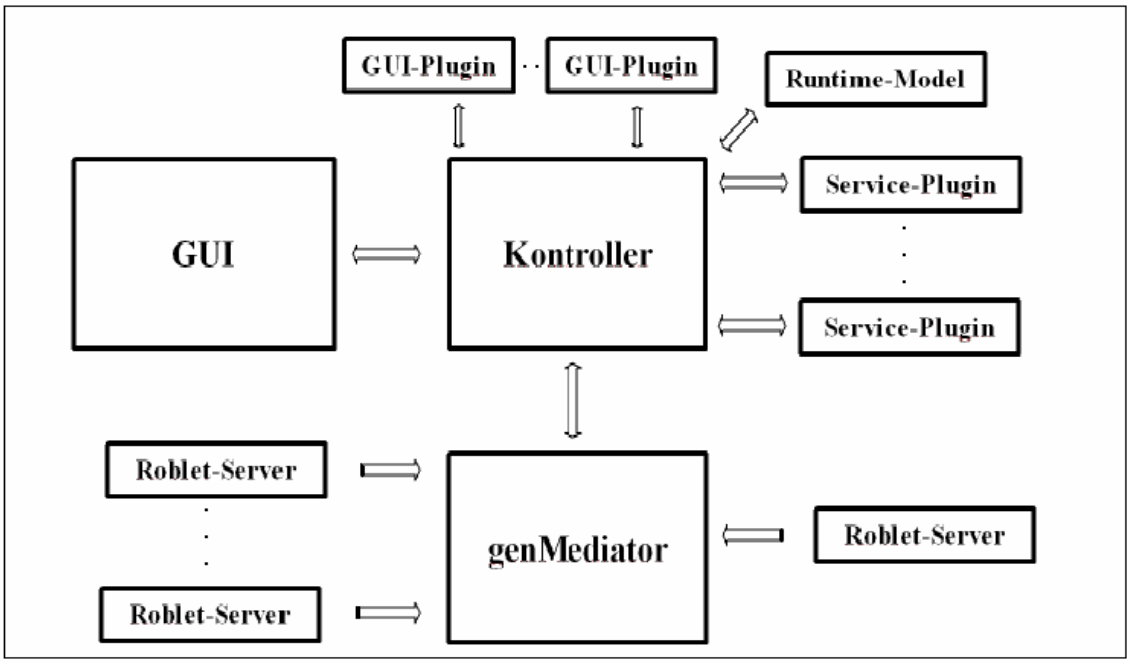


Abbildung 14: Gesamte Projektarchitektur. Das Herz der Architektur ist der Kontroller, der im permanenten Kontakt mit dem genMediator steht und ihn über Anzahl der Server im Netz und ihren Zustand abfragt. Die Kommunikation zwischen dem Kontroller und der GUI findet ereignisgesteuert statt. Über Services wird der Zugriff auf die Server und ihre Funktionen ermöglicht. Die GUI-Plugins werden beim Start vom Kontroller eingebunden und stellen die Schnittstelle zwischen dem Benutzer und der Kontrollstrukturen.

Beim Start eines Services wird eine bidirektionale Verbindung zu dem entsprechenden Server, durch die Roblets®, eingerichtet und die benötigten Informationen ausgetauscht. GUI-Plugins werden beim Start der graphischen Steuerungsoberfläche von dem Kontroller eingebunden und erhalten die Möglichkeit, falls sie gestartet werden, die Oberfläche zu manipulieren. Falls für die graphische Darstellung Daten von den, im Netz vorhandenen, Servern benötigt werden sind GUI-Plugins in der Lage Dienste der gestarteten Services in Anspruch zu nehmen.

Da jedes beliebige Plugin in die Oberfläche eingebunden werden kann, entstehen beachtliche Sicherheitsrisiken, die behandelt werden müssen. In diesem Projekt wird es dadurch gelöst, dass die Plugins mit bestimmten, speziell auf sie zugeschnittenen Privilegien ausgestattet werden. Damit können den Plugins nur die Rechte zur Verfügung gestellt werden, die sie auch tatsächlich benötigen. In der Fachwelt wird diese Vorgehensweise als Sandbox bezeichnet, weil nicht vertrauenswürdige Anwendungen mit gewissen Einschränkungen ausgeführt werden. Damit kann zum Beispiel verhindert werden, dass die Plugins das System manipulieren oder andere Plugins behindern.

Um das gesamte System zu stabilisieren, wird für jedes Service-Plugin ein eigener Thread gestartet. So können einzelne Plugins besser gehandhabt werden.

Im Hauptteil dieser Arbeit wird vertieft auf die Entwicklung und Implementierung der Services und GUI-Plugins, die auf dem oben beschriebenen System basieren und in der graphischen Oberfläche die multiple Steuerung realisieren, eingegangen.

2.4 Spezifizierung der Pläne

Zum Anfang des Projektes standen zwei Roblet®-Server zur Verfügung, die Steuerungseinheiten für Serviceroboter enthalten (siehe Kapitel 2.2). Für diese Serviceroboter sollen Client-Anwendungen entworfen werden, die die Kommunikation zu den Roblet®-Servern übernehmen. Der Architektur des Gesamtprojektes folgend, sollen diese Aufgabe Services übernehmen, die als Middleware zwischen Plugins und Server eingesetzt werden (siehe Kapitel 2.3).

Da die Kommunikation mit drei Roblet®-Server, die zwei Roboter- und einem Pfad-Server repräsentieren, gewährleistet werden soll, müssen einer oder mehrere Services implementiert werden, die Verbindungen mit den entsprechenden Servern bereitstellen. Die Verbindung zwischen den Services und den Servern soll nach Möglichkeit beim Start des Services aufgebaut werden und während der gesamten Laufzeit aufrechterhalten bleiben, so dass ein permanenter Zugriff auf die Kontrollstrukturen der Roboter und des Pfad-Servers möglich ist. Des Weiteren handelt es sich um mobile Roboter, die ihre Position verändern können. Da die Roboter in der graphischen Steuerungsoberfläche dargestellt werden, ist eine permanente Lokalisierung unvermeidlich. Die Verbindungen zwischen den Services und den Servern wird somit durch RMI (Remote Method Invocation) realisiert, das durch Roblets® eingerichtet wird. (siehe Kapitel 2.2).

Die Services sollen nach dem Aufbau der Verbindungen mit den Servern den gesamten Umfang an Steuerungselementen, der auf den Server geladenen Module, auf der Client-Seite zur Verfügung stellen. Die von den Services bereitgestellten Steuerungselemente der Roboter werden dann bei der Umsetzung der interaktiven Steuerung im Map-Plugin eingebunden. Der Pfad-Server berechnet für die Roboter kollisionsfreie Pfade in einer Umgebungskarte und muss daher zum Anfahren eines vom Benutzer festgelegten Zielpunktes zum Einsatz kommen.

Die Darstellung der Roboter soll im Map-Plugin als einfache geometrische Figur erscheinen, die durch Mausaktivitäten selektiert werden kann und die einzelnen Befehle der Robotersteuerung anbietet. Das Selektieren eines Roboters soll durch eine optische Veränderung der Darstellung des Roboters angedeutet werden, so dass sichtbar ist, an welchem Roboter der Befehl angewandt wird. Zusätzlich bieten die Roboter verschiedene Sensordaten wie Laserscans, die eine visuelle Darstellung der gescannten Umgebung erhalten sollen. Der Batteriestatus und die Temperaturanzeige der Getriebe am Fahrwerk sollen, in der direkten Umgebung der Roboter zur besseren Zuordnung der Daten, angezeigt werden.

Da mehrere Roboter vom gleichen Typ auf der gleichen Karte darstellbar sein sollen, müssen die Roboter eine eindeutige Identität haben, die sich durch eine im Netz eindeutige IP-Adresse zusammen mit einer Portnummer für die realisieren lässt. Roboter verschiedenen Typs sollen auch verschiedene Darstellungen in der Karte haben, da diese sich im Funktionsumfang und an ausführbaren Operationen unterscheiden, die für gewisse Aufträge von Bedeutung sein können.

Zum Planen von Fahraufträgen soll ein Plugin bereitgestellt werden, dass editieren von Fahrplänen für die Roboter ermöglicht. Dieses Plugin soll natürlich auch Funktionen zu Speichern, Laden, und Zusammenführen von Fahrplänen anbieten. Das Plugin soll beim Abschluss dieses Projektes zum Durchführen eines Experiments dienen, in dem die Serviceroboter zum Bewachen eines Gebäudeobjekts eingesetzt werden und bestimmte Positionen in der Karte anfahren müssen.

3 Realisierung der graphischen Steuerung

Die Realisierung der grafischen Steuerungsoberfläche wurde in mehrere Teilaufgaben aufgeteilt. Jede Gruppe bestand aus zwei beziehungsweise drei Teilnehmer, die sich mit verschiedenen Komponenten der grafischen Steuerungsoberfläche beschäftigen. In dieser Arbeit wird die Realisierung einer grafischen Darstellung der interaktiven Steuerung der im Kapitel 2.2 erwähnten Serviceroboter behandelt. Dabei sollen den Servicerobotern, nach der Realisierung dieser Teilaufgabe bestimmte Befehle durch die grafische Steuerungsoberfläche übermittelt und festgelegte Aufgaben ausgeführt werden können.

Bevor entschieden werden kann, welche Services beziehungsweise Plugins für die Realisierung einer grafischen Darstellung und interaktiven Steuerung benötigt werden, müssen zuerst die einzelnen Komponenten betrachtet werden, mit denen die Services beziehungsweise Plugin Verbindungen aufbauen oder strukturelle Abhängigkeiten besitzen. Dabei werden folgende Fragen beantwortet:

1 Welche Funktionen bieten die einzelnen Komponenten?
2 Welchen Typ haben die Verbindungen und wie werden sie zu den einzelnen Komponenten aufgebaut?
3 Welche Abhängigkeiten bestehen zwischen den Services beziehungsweise den Plugin und den einzelnen Komponenten?

Um sich ein besseren Überblick über das Vorgehen in diesem Kapitel zu verschaffen, wird eine Zusammenfassung, der zu behandelnden Komponenten, gegeben. Als Erstes werden die applikationsspezifische Module der Roblet®-Server betrachtet, da die benötigten Services einen direkten Zugriff auf die Funktionen der Module haben werden. Zweitens wird die Frage behandelt, welche Services zur Realisierung der graphischen Steuerung der Roboter benötigt werden. Da die Services beziehungsweise Plugins mit der GUI gekoppelt sind, wird dessen Einbindung in die GUI näher betrachtet. Danach werden die Konfigurations-Dateien der Services beziehungsweise Plugins behandelt, da in diesen Dateien einheitspezifische Einträge für die einzelnen Komponenten festgelegt werden, die für die grafische Erscheinung der Plugins und die Verbindung der Services zu den Servern verantwortlich sind. Als Nächstes wird der Verbindungsaufbau zwischen den Services und Servern betrachtet. Im Anschluss wird auf die dauerhafte Verbindung zwischen Services und Servern eingegangen. Die grafische Darstellung und die interaktive Steuerung der Serviceroboter beziehungsweise das Entwerfen der Fahrpläne für die Roboter wird im engen Zusammenhang mit einem Plugin zur grafischen Darstellung der Umgebungskarten realisiert. Dieses Plugin zeichnet nicht nur eine Umgebungskarte in die GUI, sondern bietet des Weiteren Schnittstellen zum Einzeichnen und Erteilen von verschiedenen Steuerungsbefehlen für die, in die Karte eingebundene, gra-

phische Objekte. Das Plugin wird als Map-Plugin bezeichnet und in einem weiteren Abschnitt dieses Kapitels behandelt.

3.1 Die Roblet®-Server

Der Aufbau der Roblet®-Server wurde im Kapitel 2.2 betrachtet, wo auch auf den modularen Aufbau eingegangen wurde. An dieser Stelle werden die applikationsspezifische Module, der zu verfügungstehende Roblet®-Server, genauer betrachtet. Es werden alle Einheiten (engl. Units) und die von den Einheiten bereitgestellten Schnittstellen beschrieben, weil wie schon oben erwähnt, die Services der Roboter einen direkten Zugriff auf die Einheiten der Module besitzen.

3.1.1 Die Robotersimulation des genRob®-Projektes

Der Robotersimulation des genRob®-Projektes ist ein Roblet®-Server, der ein applikationsspezifisches Modul mit Einheiten zur Simulation eines Serviceroboters lädt und Funktionen zur interaktiven Steuerung anbietet.

Das Modul beinhaltet vier Java-Pakete (engl. Packages) und zwei Ausnahmefehlerbehandlungen (engl. Exceptions). Die vier Pakete sind zum Steuern und Abfragen von Sensorwerten des Roboters vorgesehen. Die Ausnahmebehandlungen dienen zur Signalisierung von irregulären Situationen innerhalb der Simulation. Im Folgenden werden alle im Modul enthaltene Pakete und deren Einheiten (engl. Units) betrachtet.

Das Kopf-Paket (engl. Head Package) enthält zwei Einheiten. Die erste Einheit „Head“ liefert die Koordinaten der Kameras und der Pan-Tilt-Unit eines Kopfes. Die zweite Einheit „PanTiltUnit“ beinhaltet zwei Methoden zum Abfragen des Horizontalen- und Vertikalen-Winkels des Kopfes.

Das Karten-Paket (engl. Map Package) enthält zwei Einheiten und zwei Hilfsklassen. Die Einheit „EdgeMap“ und „ReflectorMap“ beinhalten Methoden zum Verwalten von Kanten und Reflektor-Marken in einer Kanten- und Reflektor-Karte. Die Methoden der beiden Einheiten sind für die dynamische Gestaltung der Karte notwendig. Die beiden Hilfslassen „Edge“ und „Reflector“ werden für die Erzeugung neuer Kanten und Reflektor-Marken benötigt.

Das Mobile-Paket (engl. Mobile Package) enthält fünf Einheiten, die das Fahren und Abfragen von Statusinformation des Roboters erlauben. Die Erste „Brakes“ beinhaltetet Methoden zum Prüfen und Ändern des Bremszustandes. Die Zweite „Heads“ beinhaltet Methoden zum Holen und Abfragen der Koordinaten der Köpfe eines Roboters. Die Dritte „Localisation“ beinhaltet Methoden zum Abfragen und Kalibrieren des Roboters. Die Vierte und Fünfte, „Scanners“ und „Target“, beinhalten Methoden zum Verwalten der Scanner und das Verändern der Position des Roboters. Des Weiteren enthält diese

Einheit eine Hilfsklasse „Frame“, die eine Position des Roboters angibt. Die Instanzen dieser Klasse beinhalten ein Koordinatenpaar und eine Orientierung in Robiant. Die Hilfsklasse „Robiant“ erlaubt eine, für die Roboter spezifizierte, Angabe der Orientierung. Diese Hilfsklasse bietet Methoden zur Umrechnung von verschiedenen Winkelangaben in Robiant und umgekehrt an. Die von der Einheit „Target“ ausgelösten Ausnahmen „BrakesException“ meldet Fehler, wenn versucht wird den Roboter mit angezogener Bremse zu fahren.

Das Sensor-Paket (engl. Sensor Package) enthält eine Einheit und zwei Hilfsklassen. Die Einheit „Scanner“ stellt eine Methode zum Abfragen eines aktuellen Datensatzes eines Scanners zur Verfügung. Die Hilfsklassen „Scan“ und „Spot“ beschreiben einen Datensatz und eine Position eines Laserscanners.

Für nähere Informationen zu den zur Verfügung stehenden Einheiten und deren Methoden wird auf die Dokumentation des genRob®-Projektes verwiesen [genRobSyst].

Im nächsten Abschnitt wird ein weiterer Serviceroboter des Arbeitsbereiches TAMS betrachtet.

3.1.2 Der Serviceroboter der Universität Hamburg am Department Informatik

Der Roboter-Server des Arbeitsbereiches TAMS ist ein weiterer, dem Projekt zur Verfügung gestellter Roblet®-Server, der ein applikationsspezifisches Modul mit Einheiten eines Serviceroboters lädt und Funktionen zur interaktiven Steuerung anbietet.

Der Roboter der Universität Hamburg am Department Informatik hat einen größeren Funktionsumfang als die Simulation des Roboters aus dem genRob®-Projekt. Das Modul des Roboters enthält vier Java-Pakete, mehrere Hilfsklassen und Ausnahmebehandlungen. Die einzelnen Pakete enthalten Einheiten zum Steuern des mobilen Roboters und Abfragen von Sensoren. Im Folgenden werden die einzelnen Einheiten und im Anschluss die Hilfsklassen und die Ausnahmebehandlungen näher beschrieben.

Das Geometrie-Paket (engl. Geometry Package) beinhaltet eine Einheit „Geometry2D“, die eine Methode zum Abfragen der Abmessungen der Hardware des Roboters enthält. Der Rückgabewert ist eine Instanz der Hilfsklasse „Polygon“, auf die später eingegangen wird.

Das Lokalisierungs-Paket (engl. Localisation Package) beinhaltet drei Einheiten, die das Kalibrieren und Abfragen von Sensoren des Roboters erlauben. Die Erste „Calibration“ beinhaltet eine Methode, die den Roboter an eine festgelegte Position setzt. Als Parameter bekommt diese Methode eine Instanz der Hilfsklasse „Pose“. Die Zweite „Landmarks“ beinhaltet zwei Methoden, die das Abfragen der vom Roboter aktuell-lokalisierten und der beim Start des Roboters angegebenen Positionen der Landmarken.

Als Rückgabewert liefern die beiden Methoden Instanzen der Hilfsklasse „Landmark“. Des Weiteren enthält diese Einheit die Schnittstelle „Localisation“, die eine Methode zum Abfragen der aktuellen Position des Roboters beinhaltet. Als Rückgabewert liefert die Methode eine Instanz der Hilfsklasse „Pose“.

Das Bewegungs-Paket (engl. Motion Package) enthält mehrere Einheiten zum Steuern der Mobilen-Einheit des Roboters. Die Einheit „Brakes“ beinhaltet Methoden zum Aktivieren und Deaktivieren der Bremsen und eine weitere Methode, die den aktuellen Bremsenzustand liefert. Die Einheit „Colision“ stellt Methoden zum Abfragen des Kollisionsradius des Roboters und ob der Roboter blockiert ist, bereit. Die Einheit „Destination“ bietet eine Methode zum Abfragen des aktuellen Ziels des Roboters. Als Rückgabewert liefert die Methode eine Instanz der Hilfsklasse „Pose“. Die Einheit „Motion“ enthält sämtliche Methoden zum Bewegen der Mobilen-Einheit. Diese Methoden erlauben den Roboter rückwärts, vorwärts, gerade aus und an eine angegebene Position, zu fahren. Des Weiteren beinhaltet diese Einheit Methoden zum Rotieren, Anhalten und Abfragen des Bewegungszustandes. Die Methoden der Einheit „Velocity“ erlauben das Steuern der Geschwindigkeit der Mobilen-Einheit des Roboters.

Das Sensor-Paket (engl. Sensor Package) enthält Einheiten zum Abfragen von Sensorwerten des Roboters. Die Einheit „Battery“ beinhaltet Methoden zum Abfragen der aktuellen Spannung, minimal zulässiger Spannung und den Spannungswert, wann die Batterie wieder aufgeladen werden sollte. Die Methoden der Einheit „DriveTemperature“ liefern Temperaturen der Motorgetriebe. Die Einheit „Laserscaner“ beinhaltet Methoden zum Auslesen der Scannerwerte, bestimmen der Position in der Umgebungskarte und Abfragen der Anzahl der Laserscanner. Die Einheiten „OdometryLog“ stellen Methoden zur Verfügung, die das Starten und Beenden des Schreibens der Odometry-Messwerte erlauben.

Nachdem alle Einheiten behandelt sind, sollen jetzt die einzelnen Hilfsklassen näher betrachtet werden. Die Instanzen der Hilfsklassen werden benötigt um Methoden der Einheiten zu Parametrisieren oder diese Methoden liefern Rückgabewerte, die vom Typ dieser Klassen sind. Die Klasse „Coord“ bildet Instanzen, die 2D-Koordinatenpunkte in der Landkarte repräsentieren. Die Instanzen der Klasse „Frame2D“ beinhalten 2D-Koordinatenpunkte und eine Orientierung. Die Klasse „Landmark“ erzeugt Instanzen, die Landmarken mit Position, Winkel der Orientierung, einer eindeutiger Identitäts-Nummer und des Sichtbereiches repräsentieren. Die Instanz der Klasse „Laserscan“ beinhaltet die Punkte eines Laserscans, die vom Typ der Klasse „Point2D“ sind und einen x- und y-Wert beinhalten. Die Hilfsklasse „Polygon“ erzeugt Instanzen, die Polygone darstellen. Damit ein Polygon erzeugt wird, muss beim Aufruf des Konstruktors ein Array von 2D-Punkten mitgegeben werden. Die Instanzen der Klasse „Pose“ werden zum Setzen und Auslesen der Roboter-Position in der Landkarte benötigt. Eine Position des Roboters wird durch ein Koordinatenpaar und der Orientierung gekennzeichnet und

allgemein in der Literatur als „Pose“ bezeichnet. Die Klasse „Robiant“ beinhaltet statische Methoden um Winkelangaben in Robiant umzurechnen und umgekehrt.

Die im Modul vorhandenen Ausnahmenbehandlungen verhindern unzulässige Zugriffe auf den Roboter und helfen Fehler bei der Implementierung zu beheben. Die Beschreibungen der diversen Ausnahmen können der Dokumentation des Moduls entnommen werden [MobRob].

Als nächstes wird ein weiterer Roblet®-Server betrachtet, der einen sicheren Pfad zwischen zwei beliebigen Koordinatenpunkten der Umgebungskarte berechnet.

3.1.3 Der Pfad-Server

Zur Navigation der Roboter in einer Umgebungskarte kommt ein Pfad-Server zum Einsatz. Dieser lädt ein applikationsspezifisches Modul mit einer Einheit zur Pfadberechnung.

Der Pfad-Server (engl. Path Server) lädt beim Start das Modul „genRob.genPath“. Außerdem muss dem Server mindestens ein Algorithmus zur Berechnung eines Pfades mitgegeben werden. Für diese Arbeit wird ein, am Arbeitsbereich TAMS entwickelter, Algorithmus „uhh.fbi.tams.pathplanner“ eingesetzt. Dieser Algorithmus berechnet den kürzesten Pfad zwischen zwei Punkten. Er berücksichtigt dabei die aktuell im Karten-Server gespeicherte Karte und den Umfang des Roboters. Das geladene Modul enthält eine Einheit, in der die Berechnung des Pfades durchgeführt wird.

Das Modul enthält eine Einheit, einige Ausnahmebehandlungen und vier Hilfsklassen. Im Folgenden werden die einzelnen Komponenten dieses Moduls näher betrachtet.

Die Einheit „Planner“ beinhaltet eine Methode zum Berechnen eines Pfades. Beim Aufruf diese Methode müssen der Start-, Zielpunkt und Umfang des Roboters als Parameter für die Berechnung mitgegeben werden. Die Hilfsklasse „Path“ bildet die Instanz des Berechneten Pfades und beinhaltet Methoden zum Hinzufügen von neuen Datenelemente und Zurückgeben eines Pfad-Datenläufers (engl. Path Iterator). Die Hilfsklasse „PathIterator“ bildet die Instanzen der Datenläufer und stellt Methoden zum Überprüfen, ob ein weiteres Element abgeholt werden kann, und zum Ausgeben eines Datenelementes bereit. Die Hilfsklasse “Position“ dient zum Erzeugen von neuen Datenelementen. Ausnahmebehandlungen dieser Einheit melden Fehler, wenn kein Pfad berechnet werden konnte oder ein Implementierugsfehler im Berechnungsalgorithmus vorliegt.

3.1.4 Weitere Roblet®-Server

Für das Projekt „Serviceroboter“ kommen noch zwei weitere Roblet®-Server zum Einsatz, dies ist der Karten-Server „genRobgenMap“ und der Verzeichnisdienst-Server

„genMediator“. Die beiden Server basieren genauso, wie die oben genannten, auf der Roblet®-Architektur. Auf diese Server wurde schon im Kapitel 2.2 eingegangen. Da die Roboter-Services aber keinen direkten Bezug zu den beiden Servern haben, werden auf die applikationsspezifische Module der beiden Server hier nicht eingegangen. Für nähere Informationen zum Karten- beziehungsweise Verzeichnisdienst-Server wird hier auf die Dokumentation des genRob®-Projektes verwiesen [genRobSyst]. Dennoch sollten die Dienstleistungen der beiden Server erwähnt werden, da diese für das Verständnis des Gesamtkonzeptes benötigt werden. Ein Karten-Server verwaltet Karten verschiedener Umgebungen und stellt diese Clients-Anwendungen zur Verfügung. Alle Roblet®-Server, die sich in einem Netz befinden, melden ihre Dienstleistungen bei dem Verzeichnisdienst-Server an. Client-Anwendungen können, die im Netz verfügbare Dienste, beim Verzeichnis-Server erfragen oder sich automatisch über neu hinzugekommene oder nicht mehr verfügbare Services informieren lassen.

Nachdem die Roblet®-Server, die mit der Realisierung der Services der Roboter direkt zusammenhängen, ausgiebig beschrieben wurden, wird die Entscheidung über die benötigte Services getroffen.

3.2 Services zur Realisierung der graphischen Steuerung der Roboter

In diesem Abschnitt werden die Entscheidungen getroffen, welche Services für die Realisierung der grafischen Darstellung und der interaktiven Steuerung der Roboter zum Einsatz kommen.

Die Services dienen in erster Linie als Middleware zwischen Plugins und Server und sind für die Kommunikation zwischen den beiden zuständig. Die Services bauen eine Verbindung zu den Servern auf und halten sie während ihrer gesamten Laufzeit bis der Server beendet wird. Da jeder Service nur mit genau einem Server verbunden sein kann, wird für jeden Server ein Service benötigt. Für die Realisierung einer grafischen Steuerung für die beiden oben genannten Roboter mit Einbezug des Pfad-Servers wird somit, für drei unterschiedliche Server, auch die entsprechende Anzahl von Services benötigt.

Für jeden, in die Kontrollstruktur der GUI eingebundenen, Server kann ein passender Service gestartet werden, so dass eine Kommunikation zu mehreren Servern gleichen Typs, aber an verschiedenen Ports laufend, aufgebaut werden kann. Die Entscheidung, ob einer der Services für einen Server angeboten wird, ist anhand der Konfigurationsdatei des Services festgelegt. In der Konfigurationsdatei des Services werden die, auf dem Server zu benutzenden Units, festgehalten. Somit wird dieser nur dann angeboten, wenn die entsprechenden Units auf dem Server vorhanden sind.

3.3 Anbindung der Services und Plugins an die GUI

In diesem Abschnitt wird die Anbindung der Plugins und Services an die graphische Steuerungsoberfläche betrachtet. Da die Anbindung und Konfigurationsdateien der Plugins und Services nur geringfügige Unterschiede aufweisen, wird zur einfachen Handhabung in diesem und folgenden Abschnitt, genau wie im Kapitel 2.3, zwischen Plugins und Services nicht unterschieden und beide als Plugins bezeichnet. Ist jedoch eine Unterscheidung notwendig, so werden die Plugins als GUI-Plugins und Services bezeichnet.

Vom Kontroller der GUI werden Schnittstellen bereitgestellt, die das Anbinden der Plugins erlauben. Dafür müssen diese von den Plugins implementiert werden. Die Schnittstellen stellen Methoden zum Starten und Beenden der Plugins und weitere Methoden zur Kommunikation mit anderen Plugins bereit. Beim Start eines neuen Plugins, werden die, an diesem Plugin interessierende Plugins, benachrichtigt, so dass mit dem neu gestarteten Plugin Verbindungen geknüpft werden können. Die Plugins erhalten durch den parametrisierten Konstruktor der Plugin-Klasse einen Konnektor überreicht. Dieser stellt Methoden zum Senden von Daten an andere Plugins bereit. Die Daten können entweder an alle oder an ein bestimmtes Plugin versandt werden. Des Weiteren beinhaltet der Konnektor Methoden zum Schreiben von Log-Informationen. Dies können Ausnahmefehler oder beliebige Nachrichten sein.

Der Unterschied zwischen den Schnittstellen der Services und GUI-Plugins ist, dass die GUI-Plugins einen weiteren Parameter im Konstruktor erhalten, der das Einfügen von graphischen Komponenten in die GUI erlaubt. Im Gegensatz beinhalten Services eine weitere Methode zum Aufbau einer Verbindung zu einem Roblet®-Server. Die zusätzliche Methode der Services erhält als Parameter eine Instanz vom Typ „Server“, aus dem Paket „genRob.genControl.client“, der von Kontroller bereitgestellt wird. Diese Instanz beinhaltet eine Methode run(), die das Versenden von Roblets® ermöglicht.

Um das Verbindungsmuster der Plugins mit anderen Instanzen für jedes Plugin spezifisch zu gestalten, beinhaltet jedes Plugin eine Konfigurationsdatei, die im nächsten Abschnitt betrachtet wird.

3.4 Die Konfigurationsdatei der Plugins und Services

In der Konfigurationsdatei jedes Plugins werden spezifische Angaben festgehalten, die für die Anbindung des Plugins an die graphische Steuerungsoberfläche notwendig sind und das Verbindungsmuster mit anderen Komponenten festlegen. Auf den Inhalt der Konfigurationsdateien wird im Folgenden näher eingegangen.

Die Konfigurationsdatei ist eine XML-Datei, die Parameter für die Anbindung des Plugins an die graphische Steuerungsoberfläche und Informationen über Komponenten, zu denen ein Plugin Verbindungen aufbauen möchte, bereitstellt. Des Weiteren werden der Name, unter welchem das Plugin in der GUI erscheinen soll und die Plugin-Klasse festgelegt, durch die das Plugin von der GUI angesprochen werden kann.

Die Oberfläche der GUI ist in drei Bereiche aufgeteilt, in denen die GUI-Plugins Fenster platziert werden. Dafür werden Konfigurationsdateien der GUI-Plugins mit einem Parameter ausgestattet, der die Platzierung der Fenster bestimmt.

Für die Kommunikation der Services mit einem Server wird in die Konfigurationsdatei eine Einheit (engl. Unit) eingetragen, die auf dem Server vorhanden sein muss, bevor mit dem Server eine Verbindung aufgebaut wird. Ist diese Einheit auf dem Server nicht vorhanden, so wird der Service für diesen Server nicht angeboten.

Für Verbindungen eines Plugins mit anderen Plugins werden in der Konfigurationsdatei die Namen der Plugin-Klassen und die Richtung der Verbindung eingetragen. Die Richtungen können uni- oder bidirektional sein.

Für die Kommunikation der Services mit den Roblet®-Servern kommen Roblets® zum Einsatz. Im nächsten Abschnitt wird der Aufbau der RMI-Verbindungen zwischen Services und Roblet®-Server betrachtet.

3.5 Aufbau der RMI-Verbindungen zwischen den Services und den Roblet®-Servern

Für den Aufbau einer RMI-Verbindung zwischen Services und Roblet®-Server werden auf der Client-Seite eine Schnittstelle, ein so genanntes „Remote Interface“, und ein entferntes Objekt, das das „Remote Interface“ auf der Server-Seite implementiert, so genanntes „Remote Object“, benötigt. Für weitere Informationen zu RMI siehe Kapitel 2.2.

Die Services enthalten eine Schnittstelle, die das „Remote Interface“ repräsentiert. Die Spezifikation des „Remote Interfaces“ erhält diese Schnittstelle, in dem sie von der Schnittstelle „BasicRemoteRoblet“ und die seinerseits von „Remote“, aus dem Paket „java.rmi“, erbt. Des Weiteren wird ein „Remote Object“ benötigt, das das „BasicRemoteRoblet“ implementiert. Um das „Remote Objekt“ auf einem Roblet®-Server platzieren zu können, muss es in den Rumpf eines Roblets® verpackt werden. Dies wird durch das Erben der Eigenschaften der Klasse „RMIRoblet“ realisiert. Die Klasse „RMIRoblet“ ist in einem Rumpf eines Roblets® verpackt, so dass die Instanzen dieser Klasse und die von dieser erben, die Roblets® repräsentieren und an einen Roblet®-Server versandt werden können. Das versandte Roblet® läuft dann persistent auf dem Server und erlaubt eine Kommunikation zwischen Client und Server über RMI. Die

Die Schnittstelle „BasicRemoteRoblet“ und die Klasse „RMIRoblet“ sind am Arbeitsbereich TAMS implementiert worden und befinden sich im Paket „uhh.fbi.tams.utils.genRob“. Für mehr Informationen zu „BasicRemoteRoblet“ und „RMIRoblet“ siehe Dokumentation [Utils].

Beim Start der Services wird ein Roblet® erzeugt und durch die Methode run(), die sich in der Service-Klasse befindet, zum Roblet®-Server versandt. Für nähere Informationen zu der Service-Klasse siehe Kapitel 3.3. Dieses Roblet® wird auf dem Roblet®-Server ausgeführt und bleibt persistent bis die Methode end() aufgerufen wird. Als Rückgabewert liefert der Server eine Instanz vom Typ „Object“. Da der Anwender aber weiß vom welchem Typ der Rückgabewert sein muss, kann die Instanz „Object“ zu dem entsprechendem Rückgabewert gecastet werden. In diesem Fall wird es zum Typ des „Remote Interfaces“ umgewandelt.

Nachdem das entfernte Objekt auf dem Server platziert wurde und eine Referenz, durch das Remote Interface, auf das Objekt existiert, kann auf das Objekt, wie auf ein lokales Objekt zugegriffen werden. Die Kommunikation über das Netz bleibt dem Anwender vollständig verborgen. Der einzige Unterschied zu einem sich lokalbefindlichen Objekt ist, dass beim Aufruf der Methoden des entfernten Objektes Ausnahmefehler abgefangen werden müssen, die bei der Kommunikation durch das Netz auftreten können.

3.6 Dauerhafte Verbindung zwischen Services und Servern

Die dauerhafte Verbindung zwischen den Services und den Roblet®-Server wird durch RMI realisiert. Mehr zu RMI siehe Kapitel 2.2. Der Service enthält das Remote Interface, durch das die Methoden des entfernten „Remote Objects“ aufgerufen werden können. Diese Methoden werden auf dem Server ausgeführt und liefern, falls deklariert, einen Rückgabewert.

3.7 Das Map-Plugin

Das Plugin „Map“ stellt die Umgebungskarte und verschiedene geometrische Objekte in einem Bereich der GUI dar. In diesem Projekt sind es geometrische Umrisse und Sensordaten der Roboter beziehungsweise Punkte, der Fahrplanung. Dieses Plugin dient auch als Benutzerschnittstelle zur interaktiven Steuerung der in die Karte eingezeichneten Roboter. Um die Roboter in eine Karte einzeichnen und steuern zu können, müssen die Services der Roboter die Schnittstellen „Usable“ und „Movable“ implementieren. Die Schnittstelle „Movable“ beinhaltet eine Methode moveTo( ), die als Parameter einen Float aus dem Paket „java.awt.geom“ erhält. Dieser Parameter wird an die Methode aus der Klasse „MapCanvas“ überreicht, wenn das geometrische Objekt in der Karte, durch Anklicken mit der Maus, aktiviert wurde und damit signalisiert, dass es auf

einen Eingabewert wartet. Die Eingabe des Parameters wird, nach dem Aktivieren des Objektes, durch ein anschließendes Klicken mit der Maus, an einer beliebigen Stelle in der Karte, umgesetzt. Die Koordinaten der Position, an der das Klicken geschah, werden ausgelesen und an die Methode der Schnittstelle „Movable“ weitergereicht. Die Schnittstelle „Usable“ erbt Methoden aus der Schnittstelle „Drawable“. Die Schnittstelle „Drawable“ beinhaltet Methoden zum Einzeichnen von verschiedenen geometrischen Objekten in die Umgebungskarte. Die Schnittstelle „Usable“ bietet Methoden zur Generierung von PopUp-Menüs für die, in der Karte eingezeichnete, Objekte.

3.8 Der Pfad-Service

Die hier behandelten Serviceroboter stellen zwar Methoden zum Anfahren eines Koordinatenpunktes in ihrer Umgebung bereit, jedoch keine Methoden zum Abfahren eines Pfades, dass die Mobilität der Roboter in Gewisserweise einschränkt. Um diese Einschränkung zu kompensieren, werden die Bewegungs-Methoden in Roboterservices, mit Einbezug des Pfad-Servers, implementiert und auf den Roboter dann, in Form einer Liste von Koordinatenpunkten, abgearbeitet.

Beim Start des Pfadservices wird eine RMI-Verbindung zum Pfad-Server aufgebaut, die während der gesamten Laufzeit des Services gehalten wird. Der Pfadservice benutzt zur Pfadberechnung die Methode plan(), aus der Schnittstelle „Planner“. Diese Schnittstelle bildet die Unit des „Plannners“ und ist auch in der Konfigurationsdatei als erforderliche Unit des Services eingetragen. Beim Verbinden des Pfadservices mit den Robotern, wird eine Referenz des Pfadservices an die Roboter versandt. Die Roboter erhalten somit einen Zugriff auf die Methode zur Pfadberechnung des Pfadservices. Der gleiche Pfadservice kann von mehreren Robotern parallel genutzt werden, so dass das Starten von mehreren Instanzen des Pfadservices nicht erforderlich ist. Da der Pfadservice interaktiv durch die Roboterservices benutzt wird, sind keine weiteren Zugriffsmöglichkeiten vorgesehen.

Es ist denkbar mehrere Pfad-Server mit verschiedenen Algorithmen parallel laufen zu lassen um die einzelnen Roboter mit verschiedenen Pfadberechnungsalgorithmen zu versorgen. Der Pfadberechnungsalgorithmus für einen Roboter wird zu dem Zeitpunkt festgelegt, in dem die Verbindung zwischen Roboter- und Pfadservices aufgebaut wird.

Der Pfadservice besteht aus einem „Remote Interface“, einer Klasse, aus der das Roblet® gebildet wird und die das entfernte „Remote Object“ der RMI-Verdindung repräsentiert, sowie einer Service-Klasse, die die vom Kontroller der GUI bereitgestellte Schnittstelle implementiert. Abbildung 15 stellt ein Klassendiagramm des Pfadservices dar.

Das Roblet® wird zum Aufbau einer RMI-Verbindung mit dem Roblet®-Server genutzt und stellt eine Methode getNewPath( ), zur Berechnung eines Pfades, bereit. Für die Berechnung eines Pfades werden innerhalb der Methode, falls noch nicht vorhanden, eine Funktion zum Laden der Planner-Unit aufgerufen. Durch das Laden der Unit wird eine Referenz auf die Instanz der Klasse zurückgegeben, so dass ein Zugriff auf die Methoden dieser Instanzen ermöglicht wird. Im „Planner“ wird die eigentliche Berechnung des Pfades durchgeführt. Die Methode plan( ), in der Planner-Klasse, bekommt alle nötigen Parameter zur Berechnung des Pfades und liefert, als Rückgabewert, ein Pfad zurück. Während der Pfadberechnung können Ausnahmefehler (engl. Exceptions) auftreten die eine Ausnahmebehandlung erfordern. Der berechnete Pfad wird dann an die Methode getNewPath( ) weitergeleitet.

Der berechnete Pfad (engl. Path) enthält eine Liste von Positionen (engl. Positions), die mit dem Datenläufer (engl. PathIterator) durchlaufen werden kann. Für nähere Informationen zu den Instanzen der Klasse „Path“ siehe Kapitel 3.1.3. Eine Position des Pfades

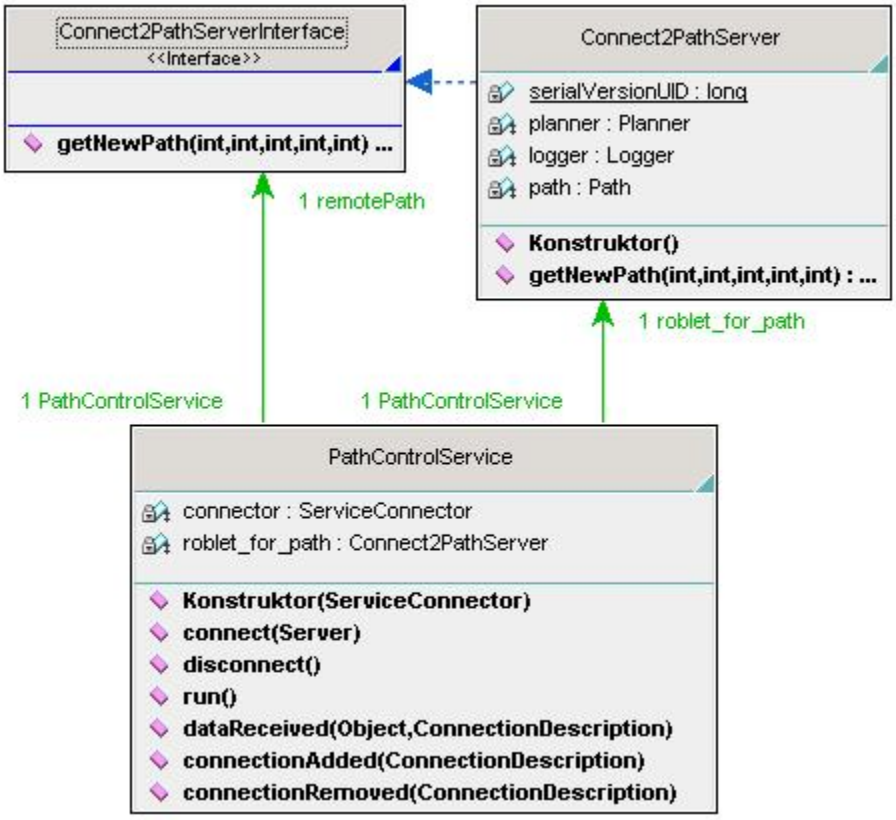


Abbildung 15: Klassendiagramm des Pfadservices. Die Klasse „PathControlServise“ bindet den Service in die graphische Steuerungsoberfläche ein. Die Instanzen der Klasse „Connect2PathServer“ repräsentieren Roblets®, die nach dem Versandt zum Pfad-Server das entfernte Objekt der RMI-Verbindung bildet. Die Schnittstelle „Connect2PathServerInterface“ bildet das „Remote Interface“ für die RMI-Verbindung zum Pfad-Server.

setzt sich aus zweidimensionalen Koordinaten zusammen. Die Koordinaten einer Position sind als öffentlich (engl. public) und endgültig (engl. final) deklariert, so dass anhand einer Referenz auf das Object, auf die Variablen zugegriffen werden kann.

3.9 Der Service für die Simulation des Roboters, des genRob®-Projektes

Der Service für die Simulation des genRob®-Projektes enthält drei Klassen, zwei Threads und eine Schnittstelle. Die Abbildung 16 zeigt ein Klassendiagramm des Roboterservices. Zunächst wird die Funktionalität aller im Service enthaltenen Komponenten betrachtet und anschließend auf die einzelne Funktionen der Komponenten eingegangen.

Die Service-Klasse repräsentiert die Klasse „RobotControlService“. Diese Klasse ist die Implementierung der Schnittstelle, die von der GUI zur Einbindung der Services bereitgestellt wird, und für die Anbindung des Services an die GUI verantwortlich ist. Die Schnittstelle beinhaltet sämtliche Methoden zum Aufbau einer Kommunikation mit dem

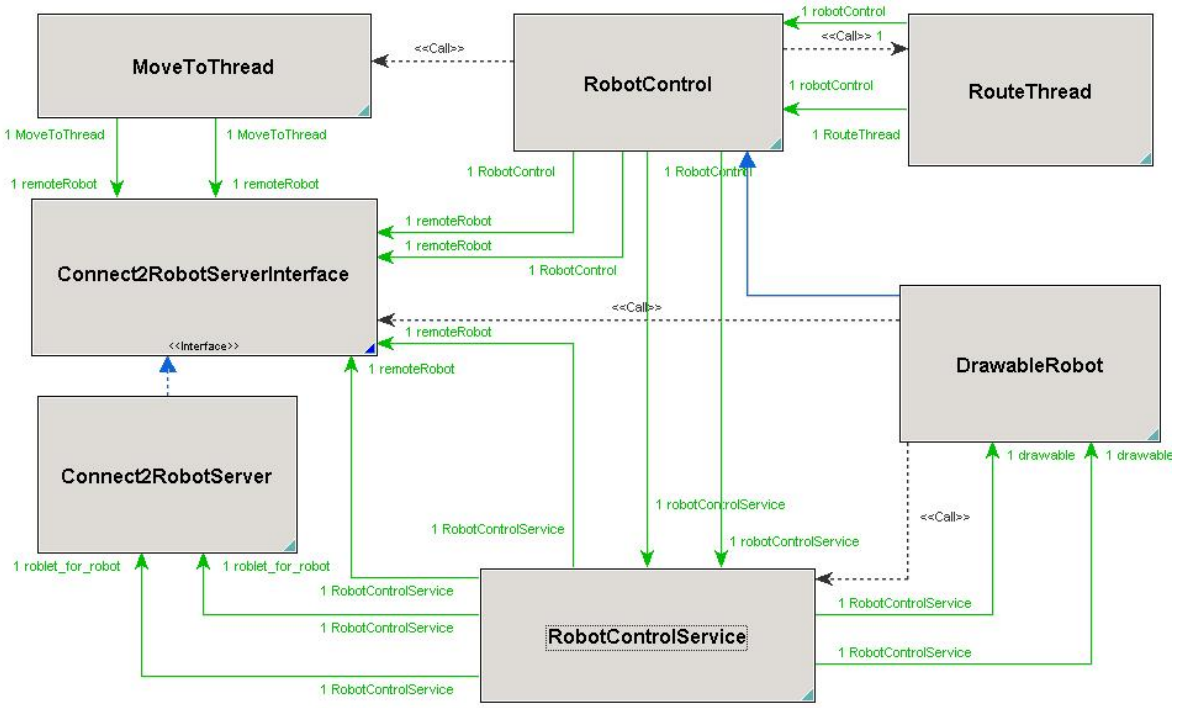


Abbildung 16: Klassendiagramm, zur visuellen Darstellung in der GUI und Realisierung der interaktiven Steuerung der Simulation. Die Klasse „RobotControllService“ repräsentiert die Service-Klasse, mit der der Service in die GUI eingebunden wird. Die Instanz der Klasse „Connect2RobotServer“ repräsentiert das Roblet®. Die Instanz der Klasse „DrawableRobot“ dient zur grafischen Darstellung und interaktiven Steuerung des Roboteres im Map-Plugin. Die Threads „MoveToThread“ und „RouteThread“ erhalten die Verbindungen zum Pfad-Server und Serviceroboter, in diesen Threads wird die Pfad-Berechnung auf dem Pfad-Server gestartet und das Ergebnis an den Roboter versandt. Die Schnittstelle „Connect2RobotServerInterface“ bildet das „Remote Interface“ für die RMI-Verbindung zum Serviceroboter.

Roblet®-Server, den Plugins beziehungsweise anderen Services. Die Methoden dieser Schnittstelle wurden im Kapitel 3.3 beschrieben und werden hier nicht weiter behandelt.

Die Instanz der Klasse „Connect2RobotServer“ repräsentiert ein Roblet®, das zu einem Roblet®-Server versandt und dort ausgeführt werden kann. Das in diesem Service eingesetztes Roblet® erbt die Eigenschaften der Klasse „RMIRoblet“ und implementiert die Schnittstelle „Connect2RobotServerInterface“, die die Spezifikationen der Schnittstelle „BasicRemoteRoblet“ erbt und das Einrichten einer RMI-Verdindung zwischen zwei Komponenten erlaubt. Für nähere Informationen zu „RMIRoblet“ und „BasicRemoteRoblet“ siehe Kapitel 3.5.

Die Schnittstelle „Connect2RobotServerInterface“ erbt, wie schon oben erwähnt, die Spezifikationen der Schnittstelle „BasicRemoteRoblet“ und bildet somit das „Remote Interface“ der RMI-Verbindung. Diese Schnittstelle erhält, nach dem Platzieren des „Remote Objects“ auf dem Server, eine Referenz auf das entfernte Objekt und erlaubt den Zugriff auf dessen Methoden.

Die Klasse „RobotControl“ beinhaltet Methoden zur Steuerung des Roboters. Die Funktionalität dieser Methoden wird entweder direkt, von den auf dem Roboter-Server zur Verfügung stehenden Methoden, übernommen oder zur Anpassung, an die aktuelle Umgebung beziehungsweise zur Erweiterung, der Funktionalität, modifiziert. Die vom Roboter zur Verfügung gestellten Funktionen wurden im Kapitel 3.1.1 beschrieben.

Die Threads „MoveToThread“ und „RouteThread“ erhalten als Parameter Referenzen zum Pfad-Server und Serviceroboter. In diesen Threads wird die Pfad-Berechnung auf dem Pfad-Server gestartet und das Ergebnis an den Roboter weitergereicht. Der Unterschied zwischen den beiden Threads ist, dass der Thread „MoveTo“ einen Zielpunkt und der „RouteThread“ eine Liste von Punkten, die als Zwischenpunkte in die Pfad-Planung miteinbezogen werden, als weiterer Parameter zu den Referenzen erhalten.

Nachdem die Funktionalität aller im Service enthaltenen Komponenten kurz veranschaulicht wurde, wird als nächstes auf die einzelnen Funktionen der Komponenten eingegangen.

Die Service-Klasse „RobotControlService“ beinhaltet nur die oben genannten Standard-Methoden, der von der GUI bereitgestellten Schnittstelle, und bedarf hier keiner weiteren Beschreibung.

Die Instanz der Klasse „Connect2RobotServer“ repräsentiert ein Roblet®. Dieses Roblet® dient, wie schon oben erwähnt, als entferntes Objekt auf dem Roblet®-Server für die RMI-Verbindung und erlaubt Zugriff auf die Methoden zur Steuerung und Abfrage der Sensorwerte des Roboters. Die Methoden der Robotersteuerung, auf dem Server, befinden sich in den Einheiten (engl. Units) eines Moduls. Das Modul der Robotersimulation wurde im Kapitel 3.1.1 behandelt. Bevor auf die Methoden der Robotersimulation zugegriffen werden kann, müssen die Einheiten, in denen sich die benötigten Methoden befinden, geladen werden. Das Laden wird mit einer, vom Server bereitgestellten, Methode realisiert. Die Klasse „Connect2RobotServer“ und somit auch die Instan-

zen dieser Klasse beinhalten einige modifizierte sowie von der Robotersimulation direkt übernommene Funktionen. Als nächstes werden einige Methoden mit modifizierter Funktionalität betrachtet.

Die Methode move( ) der Robotersimulation wurde so verändert, dass eine Liste von „Frames“ verarbeitet und der Roboter einen vollständigen Pfad abfahren kann. Dies wird mit Hilfe zweier Schleifen realisiert. Die erste Schleife ruft die Methode move() der Simulation auf und gibt als Parameter einen neuen „Frame“ mit. Die zweite Schleife fragt permanent die Position des Roboters ab, hat der Roboter die Position des „Frames“ erreicht, so wird die Kontrolle an die erste schleife wieder abgegeben. Dieser Vorgang wird so lange wiederholt, bis die vollständige Liste abgearbeitet ist. Die so abgeänderte Methode schließt die Funktion der ursprünglichen Methode der Simulation mitein, so dass das Ausführen der Original-Methode nicht mehr notwendig ist ohne die Mobilität des Roboters einzuschränken.

Die Methoden zur Rückgabe der Laserscans liefern, in der modifizierten Version, ein Array von Integern, mit alternierenden x,y-Koordinaten, so dass diese direkt in die Methoden von SWT als Parameter eingesetzt werden können. Auf die weiteren Methoden dieser Klasse wird hier nicht mehr eingegangen, da sie keine erwähnenswerte Veränderungen der Funktionalität der Original-Methoden vornehmen.

Die Methoden der Klasse „RobotControl“ bedienen sich an den Rückgabewerten und Funktionen der Methoden des entfernten Objekts auf dem Server und haben etwa den gleichen Umfang wie das Objekt selbst. Die Methoden zum Fahren des Roboters, die es in dieser Klasse zweimal gibt, erhalten verschiedene Parameter. Die erste Methode erhält als Parameter einen Zielpunkt, zu dem ein Pfad berechnet werden soll. Ihr Ergebnis wird an den Roboter weitergereicht. Die zweite Methode erhält eine Liste von Punkten, die als anzusteuernde Zwischenpunkte dienen, zum Abfahren eines festgelegten Fahrplans. Diese wird zum Abfahren eines, mit den RouteEditor-Plugin erstellten, Fahrplans benötigt. Das RouteEditor-Plugin wird später behandelt. Die zwei Move-Methoden starten zwei verschiedene Threads, die oben erwähnt wurden und im Anschluss genauer behandelt werden. Die mitgelieferten Parameter, beim Aufruf dieser Methoden, werden an die Threads weitergereicht und für die Pfadberechnung benötigt. Das Fahren des Roboters mit Einbezug des Pfad-Servers bietet die Möglichkeit den Roboter komplexe Strecken abfahren zu lassen. Die modifizierten Move-Methoden bringen aber den Nachteil, dass zum Fahren des Roboters ein Pfad-Server im Netz vorhanden und in den Service eingebunden sein muss.

Wie schon oben erwähnt werden zu den Roblet®-Servern RMI-Verbindungen aufgebaut. Die Aufrufe der Funktionen auf dem Server durch die RMI-Verbindung verhalten sich wie die Aufrufe der Funktionen auf dem lokalen Rechner, nur dass Netzwerk-Ausnahmefehler behandelt werden müssen. Wird eine Funktion auf den Roblet®-

Servern aufgerufen, so wird auch die Kontrolle an den Server abgegeben und erst dann wieder zurückgegeben, wenn die aufgerufene Funktion durchlaufen ist. Da die Berechnung oder das Abfahren eines Pfades einige Zeit in Anspruch nehmen kann, aber die Kontrolle zum Durchführen anderer Operationen benötigt wird, werden zeitintensive Prozesse in Threads ausgelagert.

Die Threads „MoveToThread“ und „RouteThread“ beauftragen den Pfad-Server Pfadberechnungen durchzuführen und reichen die Ergebnisse an den Roboter weiter. Abbildung 17 zeigt ein Ausschnitt des „MoveToThreads“. Bevor aber ein berechneter Pfad an den Roboter weitergereicht werden kann, müssen die Daten aufbereitet werden. Aufbereitet in dem Sinne, dass der Pfad-Server eine Liste von Koordinatenpaare als Ergebnis liefert. Der Roboter aber nur Instanzen vom Typ „Frame“ verarbeiten kann. Die Instanzen dieser Klasse beinhalten ein Koordinatenpaar und eine Orientierung in Robiant (siehe Homepage des genRob®-Projektes).

Die Orientierung des Roboters wird anhand zwei Geraden berechnet, dabei werden die vorherige und die aktuelle Geraden zur Berechnung miteinbezogen. Der berechnete Winkel zwischen den beiden Geraden wird mit Hilfe der Methoden der Robiant-Klasse in Robiant umgerechnet. Nachdem die Orientierung berechnet ist, wird mit dem Koordinatenpaar und der Orientierung als Parameter eine neue Frame-Instanz erzeugt. Diese Operationen werden für alle vom Pfad-Server gelieferte Koordinatenpaare in einer Schleife durchgeführt, so dass am Ende eine Liste von „Frames“ entsteht, die an den Roboter versandt wird. Da auf dem Roboter-Server alle Vorbereitungen zum Verarbeiten einer Liste von „Frames“ in der Move-Methode getroffen worden sind, verursacht das Verarbeiten von Listen auf dem Server keine Probleme.

Die Klasse „Drawable“ bildet die Schnittstelle zwischen dem Benutzer und der Steuerung des Roboters. Diese Klasse ist eine Implementierung der Schnittstellen „Usable“ und „Movable“, die von dem Map-Plugin bereitgestellt werden. Diese Schnittstellen enthalten Methoden zum Einzeichnen von grafischen Objekten in die Karte und Erstellen von PopUp-Menüs für diese Objekte.

Das PopUp-Menü bietet dem Benutzer Zugriff auf die vom Objekt angebotenen Funktionen. Für nähere Informationen siehe Kapitel 3.7. Das in diesem Service implementierte „Drawable“ zeichnet die Umrisse des Serviceroboters und erlaubt dessen Steuerung über die grafische Oberfläche. Die grafischen Steuerungselemente werden im Folgenden näher betrachtet.

Die Simulation des Serviceroboters bietet verschiedene Steuerungselemente, die durch den Service in die grafische Oberfläche mit eingebunden werden und durch das Map-Plugin eine grafische Repräsentation erhalten. Abbildung 18 zeigt die grafische Repräsentation des Serviceroboters in einer Umgebungskarte.

```java
package servicerobot.plugin.service.robotcontrol;

import genRob.genPath.unit.Path;

public class MoveToThread extends Thread {
    private final Float position;
    private final Connect2RobotServerInterface remoteRobot;
    private final Connect2PathServerInterface remotePath;

    public MoveToThread(Connect2RobotServerInterface remoteRobot,
            Connect2PathServerInterface remotePath, Float position) {
        this.remoteRobot =remoteRobot;
        this.remotePath = remotePath;
        this.position = position;
        setPriority(Thread.MIN_PRIORITY);
    }

    public void run() {
        int[] start;
        try {
            start = this.remoteRobot.getPosition();
            Path path = remotePath.getNewPath(600, start[0], start[1],
                                              (int)position.x, (int)position.y);
            PathIterator iter = path.iterator();
            final ArrayList list = new ArrayList();
            Frame frame = new Frame(start[0], start[1], 0);
            list.add(frame);
            while(iter.hasNext()){
                int x1=frame.x;
                int y1=frame.y;
                Position xyB = iter.next();
                int x2=xyB.x;
                int y2=xyB.y;
                double zaehl = (Math.sqrt(x2*x2)-Math.sqrt(x1*x1));
               double nen = (Math.sqrt(y2*y2)-Math.sqrt(y1*y1));
               double WinkInGrad = ((Math.acos(zaehl/
                       (Math.sqrt(zaehl*zaehl+nen*nen))))/Math.PI)*180;
               int angle = 0;
               if(xyB.x >= x1 && xyB.y > y1){
                   angle = Robiant.degree2robiant(WinkInGrad);
               }
               else
                   if(xyB.x < x1 && xyB.y > y1){
                       angle = Robiant.degree2robiant(WinkInGrad);
                   }
                   else{
                       if(xyB.x >= x1 && xyB.y < y1){
                           angle = Robiant.degree2robiant(-WinkInGrad);
                       }
                       else{
                           angle = Robiant.degree2robiant(-WinkInGrad);
                       }
                   }
               frame = new Frame(xyB.x, xyB.y, angle);
               list.add(frame);
            }
            frame = new Frame((int)position.x, (int)position.y, 0);
            list.add(frame);
            remoteRobot.moveTo(list);
        } catch (Exception e) {
            e.printStackTrace();
            return;
        }

    }
}
```

Abbildung 17: Ein Ausschnitt des Quelltextes des Threads „MoveToThread“. In diesem Ausschnitt sind die Berechnung des Winkels, zwischen zwei Geraden, und das und das Auffüllen einer Liste, mit „Frames“, zu erkennen.

Das in der Abbildung zu sehende PopUp-Menü beinhaltet Menüelemente, mit denen die mobile Einheit des Roboters gesteuert werden kann. Die Funktion „Reset“ setzt den

Roboter an den Ursprung des Koordinatensystems. Die Funktion „Reset to point“ kalibriert den Roboter an eine, vom Benutzer angegeben, Position in der Karte. Dafür muss der Benutzer diese Funktion mit der Maus auswählen und durch Anklicken einer Position in der Karte eine neue Position dem Roboter mitteilen. Die Funktion „hide Laserscan“ deaktiviert die Anzeige der Laserscans. Ist die Anzeige der Laserscans deaktiviert, so steht an dieser Stelle im Menü „show Laserscan“. Die Funktion „Load Route“ ermöglicht das Laden einer Datei, die mit dem RouteEditor-Plugin erstellt wird und einen Fahrplan beinhaltet. Um einen Roboter in der Umgebungskarte zu bewegen, wird der mit der Maus selektiert und ein Ziel vorgegeben.

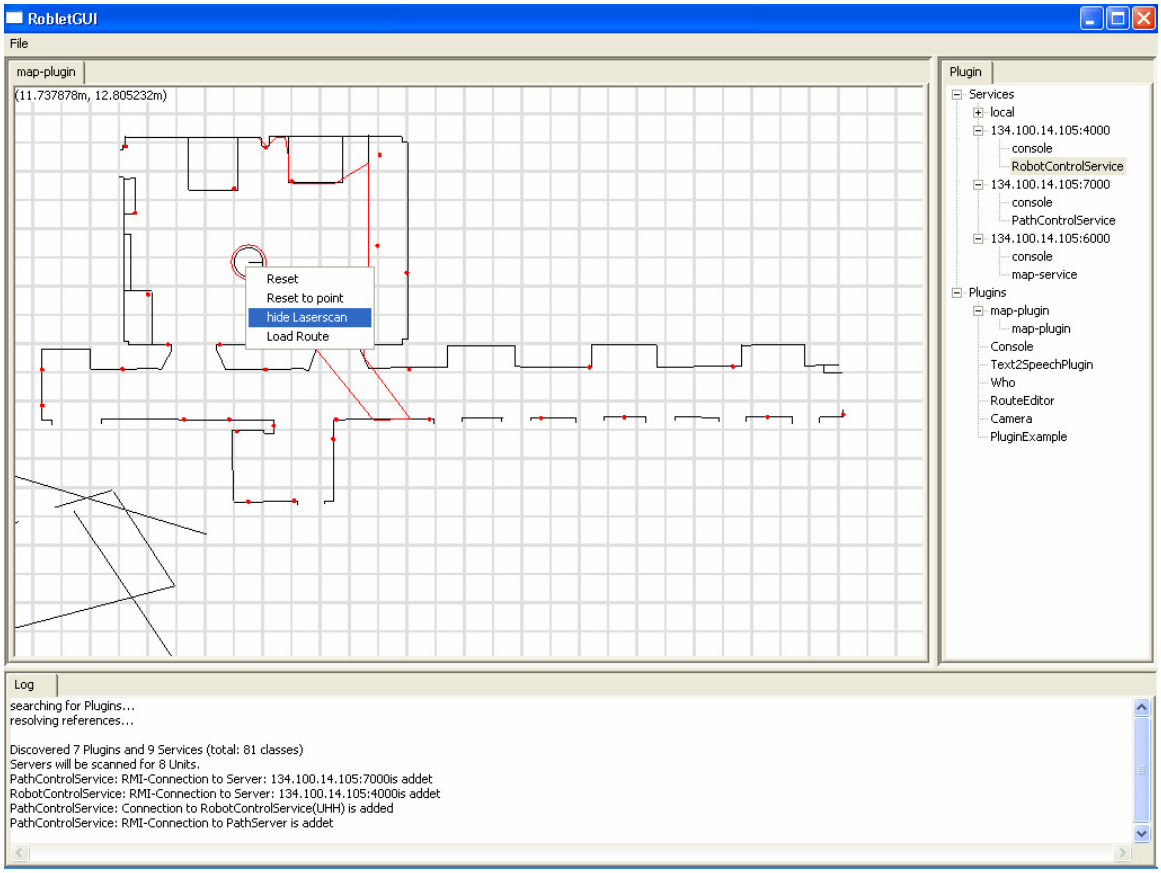


Abbildung 18: Grafische Oberfläche mit gestarteten „RobotControlService“ und sämtlichen anderen Komponenten. Im Hauptbereich der GUI ist eine Umgebungskarte mit eingezeichneten Umrissen eines Serviceroboters zu erkennen. Die roten Linien, die vom Roboter ausgehen, stellen die Laserscans in visualisierter Form dar. Des Weiteren ist am Roboter ein geöffnetes PopUp-Menü zu sehen, das ein Teil der, am Roboter ausführbare, Funktionen beinhaltet.

Da die graphische Steuerungsoberfläche als universelles Werkzeug für die Steuerung der Roboter beliebigen Typs gedacht ist, wird der oben beschriebener Service zum Testen der Anwendung eingesetzt und implementiert nur einige von der Robotersimulation angebotenen Funktionen.

Im nächsten Abschnitt wird ein weiterer Service, für den Roboter des Arbeitsbereiches TAMS, behandelt.

3.10 Der Service für den Serviceroboter am Arbeitsbereich TAMS

In diesem Abschnitt wird ein Service, für den Serviceroboter am Arbeitsbereich TAMS, behandelt. Da die Grundfunktionen zur Steuerung der mobilen Einheiten des Serviceroboters keine gravierende Unterschiede zu der oben beschriebenen Steuerung der Simulation aufweisen, wird hier im Wesentlichen nur auf die Unterschiede zwischen den

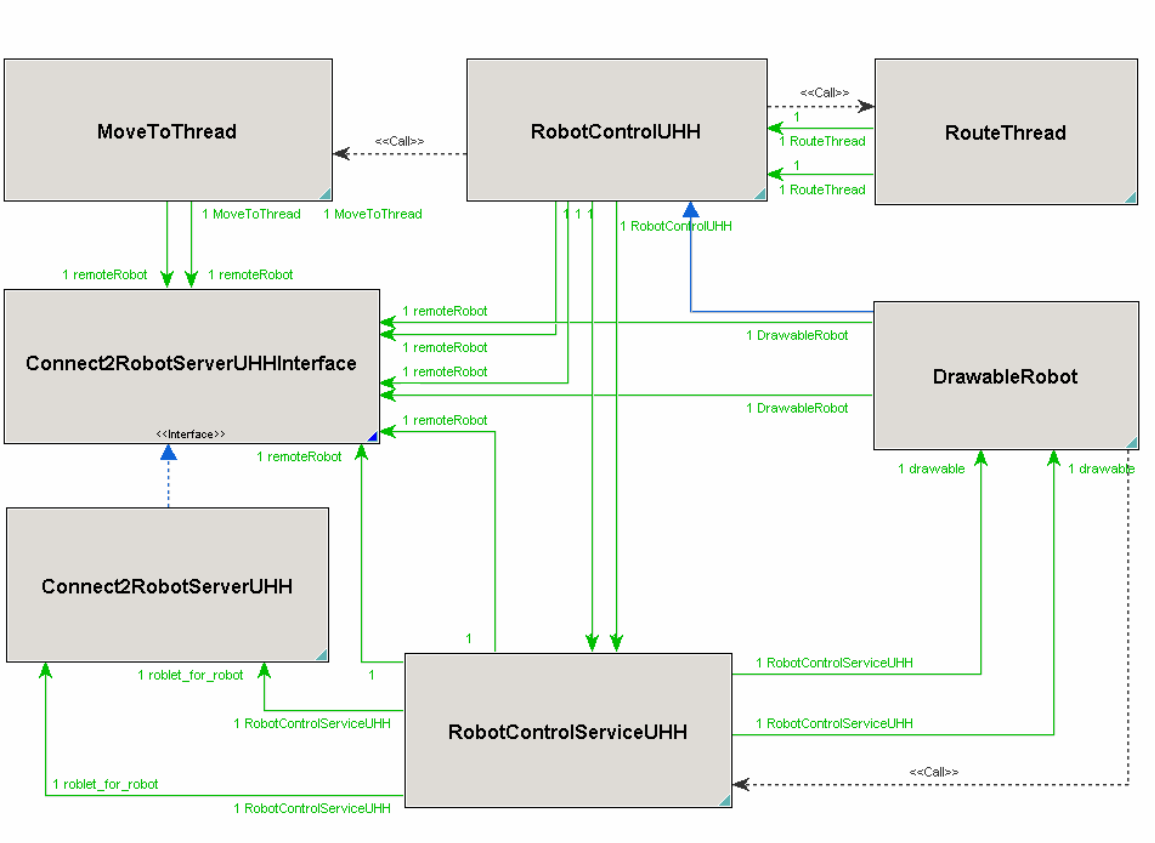


Abbildung 19: Ein Klassendiagramm, der Realisierung des Services für die graphische Steuerung des Serviceroboters am Arbeitsbereiches TAMS. Die Klasse „RobotControllServiceUHH“ repräsentiert die Service-Klasse, mit der der Service in die GUI eingebunden wird. Die Instanz der Klasse „Connect2RobotServerUHH“ repräsentiert das Roblet®. Die Instanz der Klasse „DrawableRobot“ dient zur grafischen Darstellung und interaktiven Steuerung des Roboters im Map-Plugin. Die Threads „MoveToThread“ und „RouteThread“ erhalten die Verbindungen zum Pfad-Server und Serviceroboter, in diesen Threads wird die Pfad-Berechnung auf dem Pfad-Server gestartet und das Ergebnis an den Roboter versandt. Die Schnittstelle „Connect2RobotServerInterfaceUHH“ bildet das Remote Interface für die RMI-Verbindung zum Serviceroboter.

beiden Services eingegangen. Allgemein bietet der Serviceroboter am Arbeitsbereich TAMS einen größeren Umfang an Funktionen zur Steuerung der mobilen Einheit. Die einzelnen Funktionen des Roboters wurden im Kapitel 3.1.2 näher behandelt.

Wie im Klassendiagramm aus der Abbildung 19 ersichtlich ist, beinhaltet der hier betrachtete Service „RobotControlServiceUHH“ die gleichen Komponenten, wie der

„RobotControlService“. Die Namensgebung der Komponenten unterscheidet sich nur geringfügig, so dass eine direkte Zuordnung möglich ist.

Die Anzahl der von Roboter angebotenen Funktionen ist wie oben erwähnt umfangreicher, so dass eine größere Menge in die grafische Steuerungsoberfläche miteinbezogen wird. Abbildung 20 stellt den Serviceroboter in einer Umgebungskarte dar. Am Roboter ist ein PopUp-Menü geöffnet, das die auf dem Roboter ausführbare Funktionen zeigt. Als nächstes werden diese Funktionen näher betrachtet.

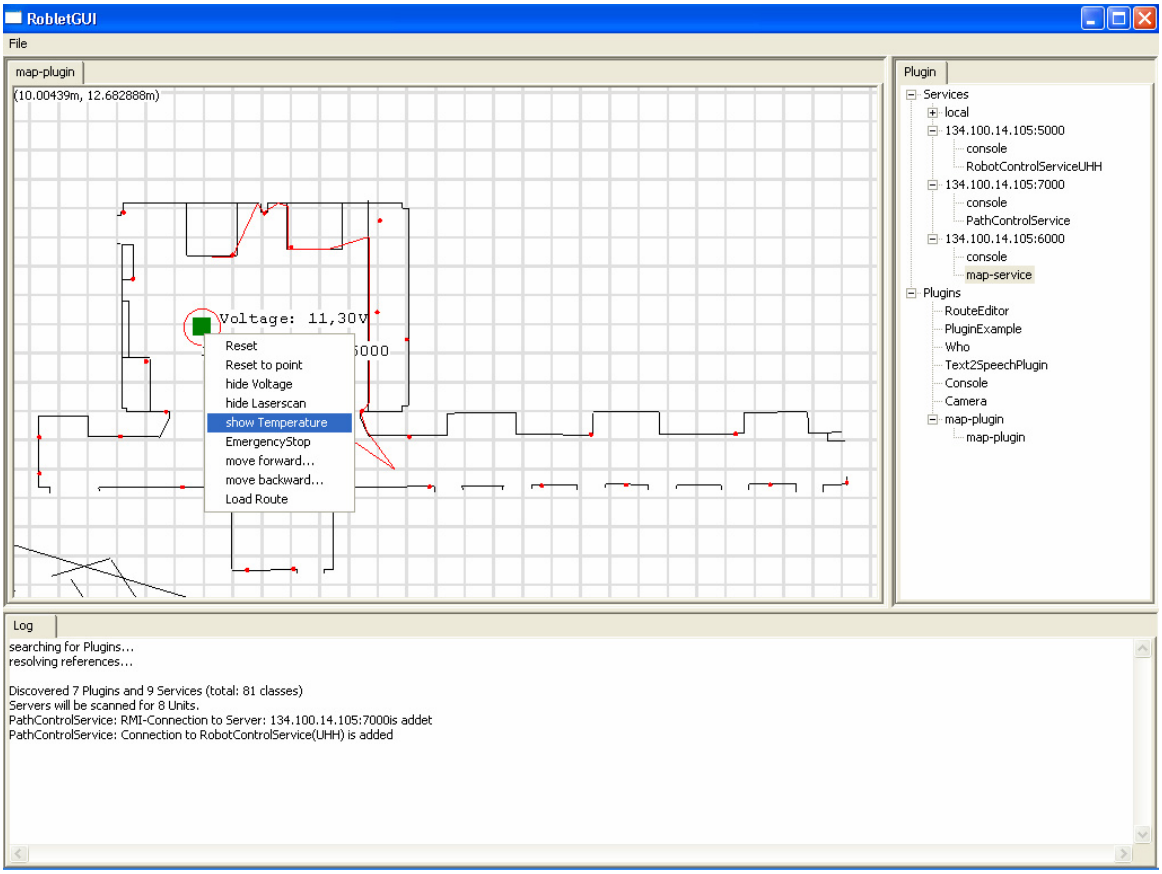


Abbildung 20: Grafische Oberfläche mit gestarteten „RobotControlServiceUHH“ und sämtlichen anderen Komponenten. In Fenster des Map-Plugins ist eine Umgebungskarte mit den graphischen Darstellung des Roboters zusehen. Am Roboter ist das PopUp-Fenster geöffnet, in dem die Funktionen des Roboters ausgewählt werden können.

Die Funktionen „Reset“ und „Reset to point“ bieten die Möglichkeit den Roboter an den Ursprung des Koordinatensystems beziehungsweise an eine festgelegte Position der Karte zu kalibrieren. Die Funktionen „show Voltage“ beziehungsweise „hide Voltage“ erlauben das Aktivieren beziehungsweise Deaktivieren der Spannungsanzeige für die Batterie. Die Funktionen „show Laserscan“ beziehungsweise „hide Laserscan“ bieten die Möglichkeit des Aktivierens beziehungsweise Deaktivierens der visuellen Darstellung des Laserscans. Die Laserscans eines Roboters werden als rote Linien, von Punkt zu Punkt der vom Scanner erfassten Koordinaten, angezeigt. Die Funktionen „show Temperature“ beziehungsweise „hide Temperature“ aktivieren beziehungsweise deaktivieren die Temperaturanzeige des Antriebs. Die Spannungs- und Temperaturanzeige

werden im aktivierten Zustand in unmittelbarer Nähe des Roboters angezeigt, so dass die wichtigen Sensordaten immer im Blickfeld des Benutzers sind. Die Funktion „EmergencyStop“ erlaubt ein Notfallstopp des Roboters, dabei werden allen Benutzern des Roboters die Rechte zum Bewegen der mobilen Einheit entzogen und ein Halt des Roboters durchgeführt. Die zu diesem Zeitpunkt aktiven Aufträge werden verworfen, so dass im Nachhinein der gewünschte Auftrag neu gesetzt werden muss. Die Funktionen „move forward“ beziehungsweise „move backward“ erlauben den Roboter vorwärts beziehungsweise rückwärts zu bewegen. Die zu fahrende Distanz wird beim Ausführen der Operation in einem Dialogfenster angegeben.

Im Folgenden wird ein Plugin betrachtet, das für die Serviceroboter Fahrplanungen ermöglicht.

3.11 Das RouteEditor-Plugin

Um die Funktionalität der Serviceroboter zu erweitern, wird ein Plugin implementiert, das erlaubt Fahraufträge für die Serviceroboter zu erstellen.

Als erstes wird auf den Zweck des Plugins eingegangen, um den Nutzen eines solchen Plugins aufzuzeigen.

Wird dem Serviceroboter ein Auftrag erteilt, einen bestimmten Punkt in der Karte anzufahren, so wird durch den Pfad-Server ein Pfad zu diesem Punkt berechnet und dieser an die Roboter weitergereicht. Der Benutzer der Roboter hat keinerlei Einflüsse wie der Pfad verlaufen wird. Das heißt, möchte der Benutzer bestimmte Punkte als Zwischenstationen für einen Roboter einplanen, muss dem Roboter erst ein Fahrauftrag bis zu der ersten Zwischenstation mitgeteilt werden. Ist der Roboter an diesem Punkt angekommen, wird dann ein weiterer Fahrauftrag bis zur nächsten Zwischenstation, oder wenn keine weitere Zwischenstation eingeplant sind, bis zum Endpunkt mitgeteilt. Es ist durchaus denkbar und für manche Szenarien sogar sinnvoll für die Roboter eine Warteschlange zu Implementieren, in der die oben genannten Zwischenstation gespeichert werden. Das hier zunächst beschriebene Plugin ist eher für die Planung konstanter Routen gedacht. Man kann sich zum Beispiel folgendes Szenario vorstellen, ein Serviceroboter dient als Wachroboter eines Industrie-Komplexes, der bestimmte Zwischenpunkte anfahren muss, wo er seiner Wachfunktion nachkommen soll. Die Wachroute kann mit dem RouteEditor-Plugin einmal geplant und gespeichert werden. Soll zu einer bestimmten Zeit der Serviceroboter seine Wachfunktion aufnehmen, kann der geplante Fahrauftrag geladen werden. Es sind noch weitere Szenarien, wie permanente Zulieferung von Bauteilen in Industrie, die an verschiedenen Stellen abgeholt werden müssen, denkbar, aber auf die soll hier nicht weiter eingegangen werden.

Im nächsten Abschnitt wird der strukturelle Aufbau der RouteEditor-Plugins betrachtet, dabei wird auf die einzelnen Klassen und deren Funktionalität näher eingegangen.

Das Plugin besteht aus drei Klassen, die verschiedene Aufgaben des Plugins übernehmen. Die Klasse „RouteEditorPlugin“ repräsentiert die Plugin-Klasse, durch die das Plugin in die GUI eingebunden wird. Welche Standard-Methoden diese Klasse beinhaltet und dessen Funktionalität, ist im Kapitel 3.3 behandelt worden. Des Weiteren beinhaltet diese Klasse Implementierungen sämtlicher Elemente für die graphische Erscheinung des Plugins in der GUI. Diese graphische Elemente greifen auf die implementierten Funktionen des Plugins zu. Dies sind Funktionen zum Auslesen, Speichern und Neuerzeugung eines Fahrplans, und Erzeugen, Löschen und Verschieben eines Punktes. Die genannten Funktionen sind in einer gesonderten Klasse untergebracht, um die Darstellung des Plugins von der ausführenden Einheit zu trennen. Die Instanzen der Klasse „Drawable“ dienen zum Einzeichnen von Fahrplanpunkten in die Karte des Map-Plugins. Auf die Instanzen dieser Klasse wird noch im weiteren Verlauf der Beschreibung des, hier behandelten, Plugins eingegangen. Für den Aufbau von „Drawables“ und deren Funktionalität, siehe Kapitel 3.7

Als nächstes werden die Anbindung, die Darstellung in der GUI und ein Überblick über die Funktionalität des RouteEditor-Plugins beschrieben.

Die Anbindung der Plugins ist ausführlich in dem Kapitel 3.3 behandelt worden und benötigt somit für das RouteEditor-Plugin keine besondere Behandlung. Es wird an dieser Stelle direkt mit der Darstellung in der GUI und der Funktionalität des Plugins begonnen. Wie in dem Kapitel 2.4 schon angesprochen wurde, soll das RouteEditor-Plugin diverse Funktion zum Bearbeiten von Fahrplänen enthalten. Die folgenden Abbildungen 21, 22 und 23 geben einen groben Überblick, über die im Plugin verfügbare Funktionalität. Der Zugriff auf die wichtigsten Funktionen geschieht über die Buttons des Plugins. Da das RouteEditor-Plugin direkt mit dem Map-Plugin interagiert und das Map-Plugin den mittleren Bereich der GUI belegt, wird das Fenster des RouteEditor-Plugin im Seitenbereich der GUI platziert. Im unteren Bereich der GUI werden die Log-Informationen angezeigt. Mit dieser Platzierung der Fenster hat der Benutzer, bei der Planung eines Fahrauftrages einen bequemen Zugriff auf alle wichtigen Funktionen und Überblick über die Gesamtsituation.

Die graphische Darstellung der Punkte der Fahraufträge wird durch Instanzen der Klasse „Drawable“ repräsentiert. Diese Klasse Implementiert die Schnittstellen „Usable“ und „Movable“ aus dem Map-Plugin. Durch das Implementieren der beiden Schnittstellen wird die Möglichkeit in der Umgebungskarte Objekte zu Zeichnen und Realisierung einer interaktiven Steuerung der Objekte gegeben. Für nähere Informationen zu den Schnittstellen siehe Kapitel 3.7. Die „Drawables“ werden im RouteEditor-Plugin erzeugt und durch die Send-Methode zum Map-Plugin versandt. Zur Kommunikation

zwischen den Services und Plugins siehe Kapitel 3.3. Durch den Aufruf der Methode draw(), der Instanzen „Drawable“ vom Map-Plugin, werden die Punkte der Fahrtplanung in die Umgebungskarte gezeichnet.

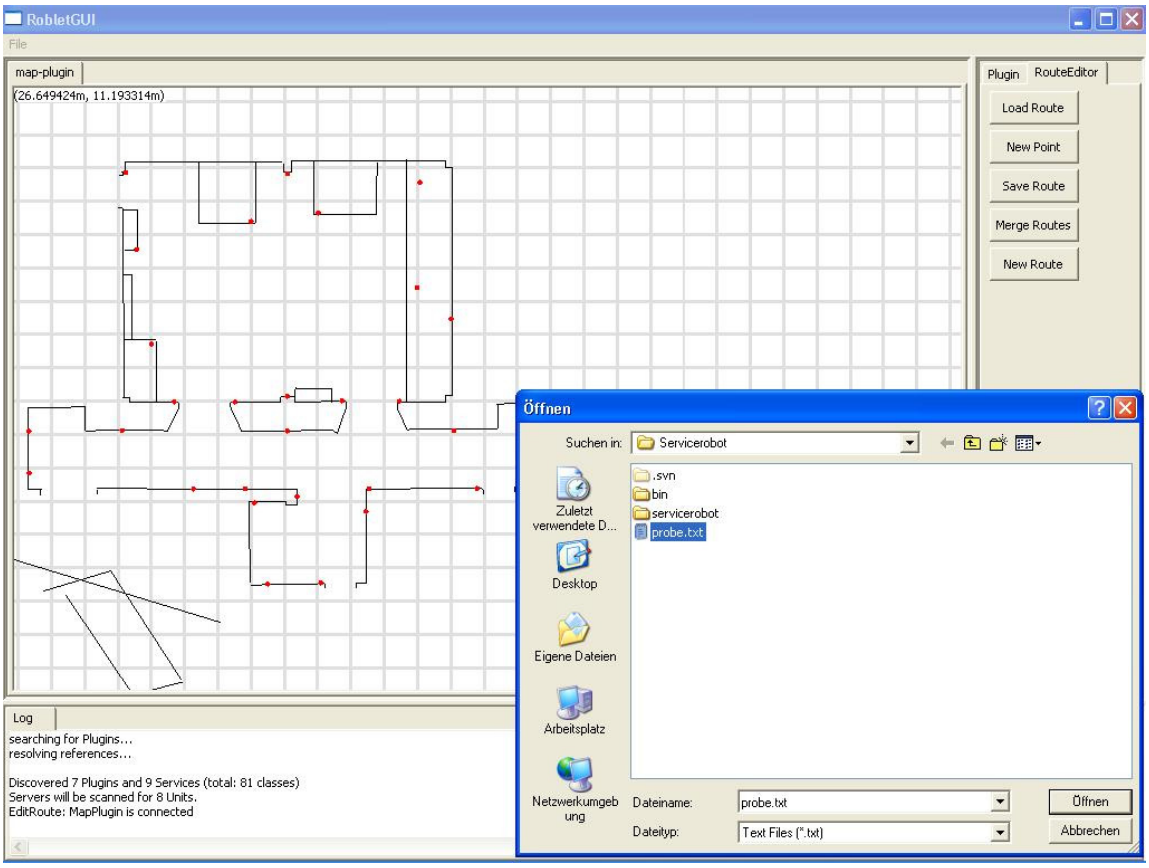


Abbildung 21: Grafische Oberfläche mit gestarteten RouteEditor- und Map-Plugin. Im Fordergrund ist ein Dialog zu sehen, der das Öffnen einer Datei erlaubt. Nachdem die Datei geladen ist, werden die Punkte der Fahrplanung in die Karte eingezeichnet.

Zum Editieren der Fahraufträge werden einige Funktionen benötigt, die als nächstes beschrieben werden.

Die Namensgebung der Funktionen wird am Beispiel der Standard-Editoren gehalten, so dass sich jeder Benutzer, der mit einem Editor schon mal gearbeitet hat, sich schnell und intuitiv zu Recht finden kann. Die Funktion „Load Route“ startet ein Fenster zum öffnen einer Datei, aus den eingelesenen Daten werden Drawable-Instanzen gebildet und zum Map-Plugin versand, das die grafische Repräsentation der Instanzen realisiert. Abbildung 21 zeigt die GUI mit einem Dialog, zum Öffnen einer Datei, im Vordergrund.

Die Funktion „New Point“ fügt einen neuen Punkt zum Fahrplan hinzu. Nach der Betätigung des Buttons wird eine neue Drawable-Instanz gebildet und eine Booleschvariable gesetzt, die dem Map-Plugin signalisiert, dass die Instanz auf einen Eingabewert zur Positionierung des Punktes wartet. Die Eingabe der Positionierungs-Koordinaten wird Anhand eines Mausklicks in der Karte, wo sich der Punkt befinden

soll, realisiert. Die Funktion „Save Route“ dient zum Speichern von Fahraufträge, die gespeicherte Datei enthält dann alle Koordinaten der Punkte. Die Funktion „MergeRoutes“ ermöglicht mehrere Fahrauftrage zusammenzuführen, dabei muss der erste Fahrauftrag mit der Funktion „Load Route“ geladen werden und dann mit der Funktion „Merge Routes“ alle weitere Fahraufträge, die zusammengeführt werden sollen, nachgeladen. Das Ergebnis kann nach Wunsch dann als neuer Fahrauftrag gespeichert werden. Die Funktion „New Route“ ermöglicht das beginnen einer neuen Fahrtplanung. Wurde eine Fahrtplanung davor begonnen und nicht abgespeichert, so erscheint ein Warnungs-Fenster, das den Benutzer zum Abspeichern auffordert. Abbildung 22 zeigt die GUI mit einem Warnungs-Fenster im Vordergrund. Ist das abspeichern nicht gewünscht, so kann die Fahrtplanung verworfen werden.

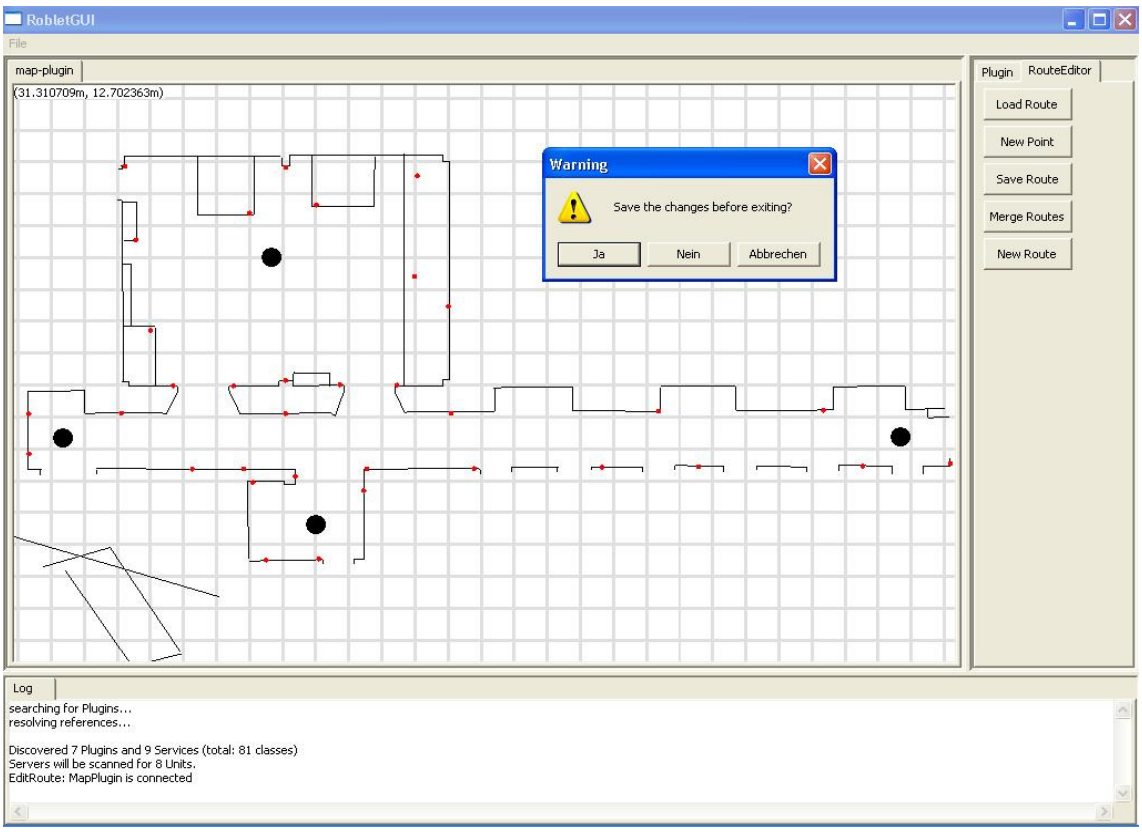


Abbildung 22: Grafische Oberfläche mit gestarteten Map- und RouteEditor-Plugins. Im Vordergrund ist Warnungs-Fenster zu sehen, das den Anwender nach Abspeichern der Daten fragt, ob er einen neuen Fahrauftrag planen möchte.

Das direkte Editieren einzelner Punkte einer Fahrtplanung, wird in einem PopUp-Menü der Punkte ermöglicht. Wird ein Punkt in der Umgebungskarte mit der rechten Maustaste angeklickt, so erscheint ein PopUp-Menü in dem die einzelnen Funktionen ausgewählt werden können. Die Funktion „Remove“ dient zum entfernen eines Punktes aus der Fahrtplanung. Die Funktion „Move“ verschiebt einen Punkt an eine neue Position. Das Mitteilen der neuen Position des Punktes wird, wie bei der Funktion „New Point“, durch anklicken der Position, an die der Punkt verschoben werden soll, realisiert. Die Reinfolge der einzelnen Punkte ist durch den Index der Liste, in der die Punkte verwal-

te verwaltet werden, festgelegt, so dass die Reinfolge der Generierung einen direkten Einfluss auf Abfahrreihenfolge hat.

Soll die generierte Fahrtplanung an einen Roboter weitergereicht werden, so muss diese erst abgespeichert und dann auf dem Roboter geladen werden. Wie das Laden einer Fahrtplanung auf den Roboter realisiert ist, wird in den Kapiteln 3.9 und 3.10 beschrieben.

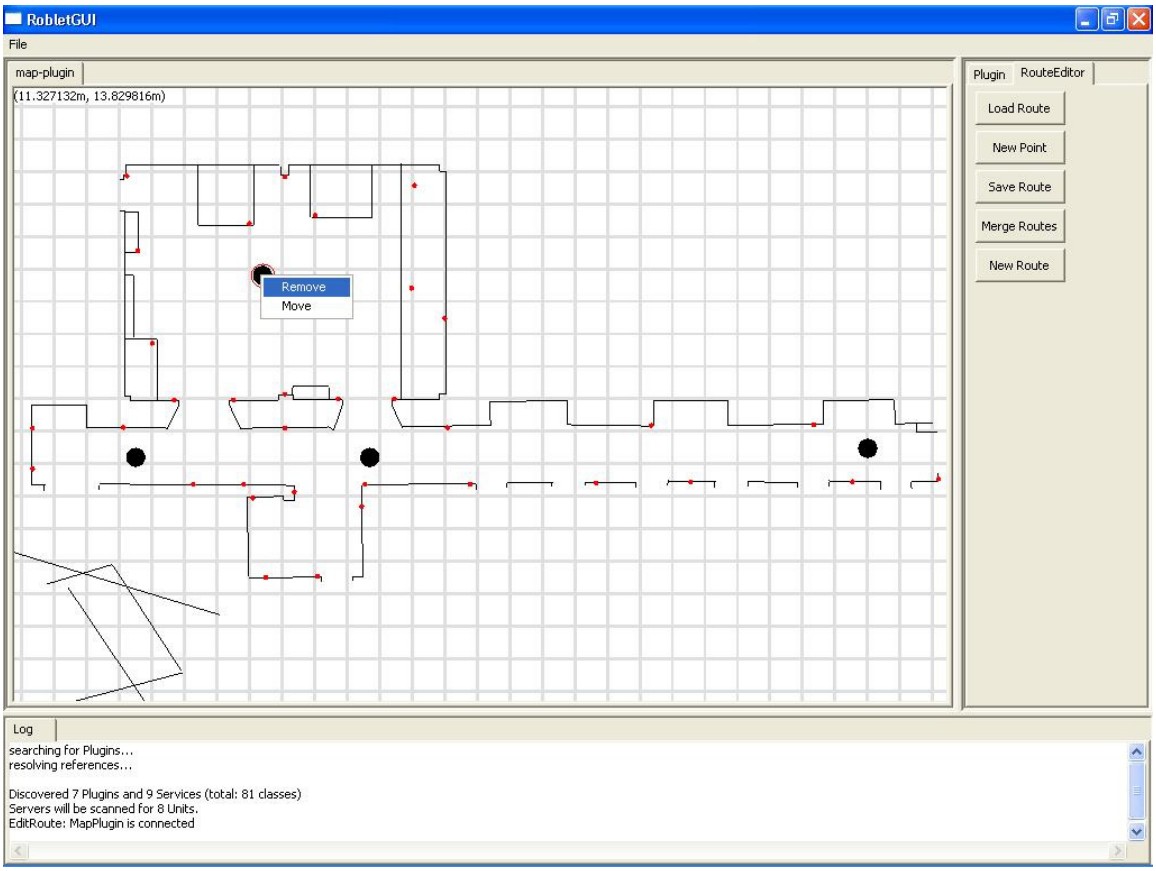


Abbildung 23: Zeigt die grafische Oberfläche mit gestarteten Map- und RouteEditor-Plugins. In der Umgebungskarte sind die Punkte eines Fahrplans zu sehen. An einem der Punkte ist durch anklicken, mit der rechten Maustaste, ein PopUp-Menü geöffnet worden. Dieses Menü bietet Funktionen zum Löschen und Verschieben des Punktes.

3.12 Erweiterungsexperiment der graphischen Steuerungsoberfläche

Im Vordergrund dieses Projektes stand die Entwicklung einer graphischen Steuerungsoberfläche, die vielen möglichen Szenarien angepasst werden kann. Nach dem die graphische Oberfläche implementiert worden ist, lag der Gedanke nah, die Anpassungsfähigkeit der graphischen Steuerung zu testen.

Dafür wurde ein Szenario ausgewählt, in dem ein Roboter definierte Punkte einer räumlichen Umgebung anfährt. Solche Szenarien werden heutzutage schon umgesetzt, zum Beispiel werden zur Bewachung von Räumlichkeiten Wachroboter [WachRob] oder in

oder in der Industrie, zum Transport von Gegenständen [TranspRob] zwischen zwei oder mehreren festen Station eingesetzt.

Damit ein Serviceroboter mit einer Wachfunktion erweitert werden kann, müssen Routen durch den Benutzer festgelegt werden. Zum Realisieren des oben genannten Szenarios müssen dem Benutzer solche Funktionen wie festlegen, editieren oder löschen einer Route zur Verfügung gestellt werden. Eine Route wird durch den Benutzer als Orientierungspunkte festgelegt, die einzeln angefahren werden müssen. Der Weg zwischen diesen Punkten wird ohne Einfluss des Benutzers durch den Pfad-Server berechnet. Weitere Aufgaben sind Routen zu speichern, zu laden und schließlich dem Roboter zu übergeben. Diese Erweiterungen werden in Form eines Plugins implementiert und in die graphische Steuerungsoberfläche eingebunden.

Die graphische Steuerungsoberfläche wird, wie im Abschnitt „Das RouteEditor-Plugin“ beschrieben, durch ein GUI-Plugin erweitert und erlaubt die Speicherung und das Laden von Routen. So können mehrere feste Routen gespeichert werden, spätere Änderung und die Hinzufügung weiterer Routen ist jeder Zeit möglich.

Durch das eingebundene Plugin wird nicht nur die Oberfläche erweitert, sondern auch der Bedienkomfort des Wachroboters angenehmer gestaltet. So können mehrere von einander unabhängige Routen festgelegt und gespeichert werden. Dem Bediener bleibt nur die Route zur Laden, alles andere passiert ohne seinen Eingriff, voll automatisch. Roboter fährt, die ihm übergebene Route, selbständig ab. Auch die Kollisionsvermeidung geschieht ohne Eingriff des Benutzers, die ausgewählte Route, also eine Ansammlung von Koordinatenpaaren, wird in Paare von Punkten zerlegt. Diese Paare werden zusammen mit dem Radius des Roboters an Pfad-Server versandt und ein sicherer, also ein kollisionsfreier Weg, wird berechnet. Danach werden die Koordinatenpaare, einzeln nach einander an den Roboter übergeben und durch permanentes abfragen abgewartet, bis der Roboter den nächsten Punkt erreicht. Danach wird weiteres Koordinatenpaar übergeben.

Die Auswahl und das Abfahren unterschiedlicher Strecken nacheinander ist im gleichen Plugin realisiert worden. Mehrere Routen werden durch die Funktion „MergeRouts“ zusammengefügt und können als eine Gesamtroute gespeichert werden, die auf den Roboter geladen und von ihm abgefahren werden kann. Dadurch wird die Qualität des Roboters als Wächter verbessern.

Abbildung 24 stellt die einzelnen Koordinaten einer geladenen Route in Form von schwarzen Punkten dar. Ein Roboter, hier als grünes Quadrat angezeigt, fährt die übergebene Route vom Punkt zum Punkt ab.

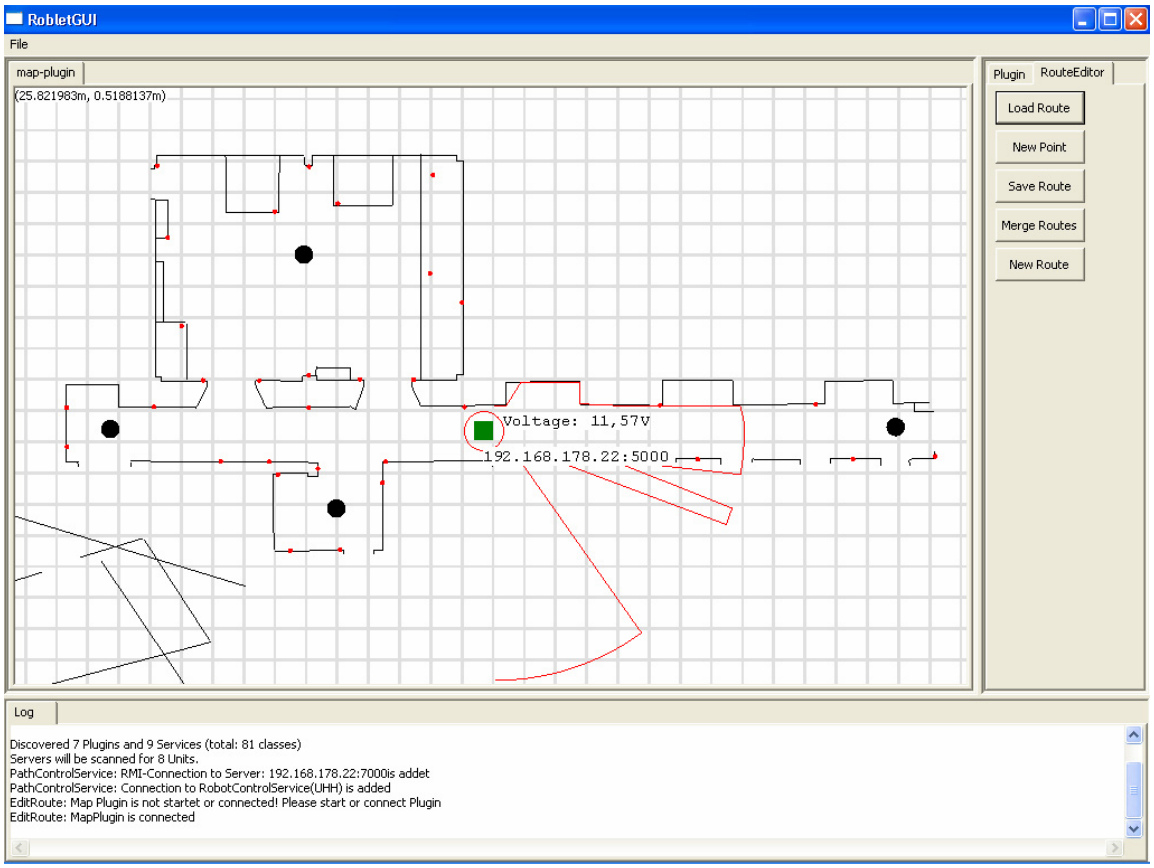


Abbildung 24: Umgebungskarte mit der geladenen Route, hier in Form schwarzer Punkte angezeigt. Der Roboter fährt die eingezeichnete Route ab.

Dieses Experiment verdeutlicht, dass die Überlegungen, die diesem Projekt zur Grunde lagen, richtig waren. Das Konzept der graphischen Steuerungsoberfläche, die durch Plugins und Services realisiert wird, erweist sich auch in der Praxis als benutzerfreundlich, leicht anpassbar und erweiterungsfähig.

Im nächsten Abschnitt findet eine Auswertung dieses Projektes mit der anschließenden Diskussion statt. Des Weiteren wird ein Ausblick gegeben.

4 Fazit

Das Ziel dieser Arbeit war es eine graphische Steuerungsoberfläche zu entwickeln. In erster Linie sollte die Steuerungsoberfläche leicht erweiterbar sein. Deswegen basiert der Projektentwurf auf verteilter Architektur, die ihrerseits auf Java und der Java Virtual Maschine aufbaut. Dadurch wird Plattformunabhängigkeit garantiert und die Ressourcenverteilung verbessert.

Um diese Idee weiter auszubauen wurde die graphische Steuerungsoberfläche als ein Rahmenwerk (engl. Framework) realisiert. Dieses Rahmenwerk ist der einzige feste Bestandteil der graphischen Oberfläche. Alle Funktionen der Steuerung oder graphischen Darstellung werden durch den Benutzer als Plugins beziehungsweise Services eingebunden. Damit wird Anpassungsfähigkeit und leichte Erweiterbarkeit garantiert, siehe Kapitel 2.3. Diese Eigenschaften wurden im Kapitel 3.12 durch ein Experiment. Die Ergebnisse stützen die obigen Behauptungen.

Im Weiteren wurde Anhand dieses Experiments nachgewiesen, dass durch die gewählte Architektur der graphischen Steuerungsoberfläche, die Benutzerfreundlichkeit verbessert werden kann. Zum Beispiel wurde beschrieben, dass alle möglichen Routen vorab gespeichert werden können und somit der Benutzer eine Route nur noch auf einem Roboter laden muss.

Abschließend lässt sich feststellen, dass nach einem Semester eine graphische Steuerungsoberfläche implementiert und einsatzbereit ist. Es wurden nicht alle, die zum Anfang dieses Projektes, gesetzte Ziele erreicht. Dennoch durch den modularen Aufbau kann die graphische Steuerungsoberfläche weiterhin verbessert und an das gewünschte Ergebnis angepasst werden.

Auch dem Ziel graphische Steuerungsoberfläche möglichst vielen Szenarien anpassen zu können, wurde die entwickelte Oberfläche gerecht. Beliebige Funktionen können in die GUI eingebunden und als Steuer- oder Darstellungselemente eingesetzt werden. In der Praxis wurde dieses Ziel, durch im Kapitel 3.12 dargestelltes Experiment, verifiziert.

In diesem Projekt ist deutlich geworden, dass das Model eines auf einer modularen Architektur aufgebauten Rahmenwerkes, das durch Plugins einem Szenario angepasst werden kann, enorme Vorteile bietet. Damit das Rahmenwerk durch Funktionen erweitert werden kann, die als Plugins realisiert werden, müssen Plugins bestimmte Schnittstellen implementieren. Ein Rahmenwerk muss theoretisch nie geändert werden. Durch den technischen Fortschritt und die Entwicklung weiterer Ideen und Algorithmen kön-

können bessere Modele entwickelt werden, dennoch ist dieses Model zukunftsweisend und wird noch viele Entwicklungsteams beschäftigen.

Im nächsten Kapitel wird ein Ausblick über mögliche Erweiterungen und Verbesserungen dieses Projektes gegeben.

5 Ausblick

Das vorliegende Projekt dauerte über ein Semester, so dass Ziele nur in gewisse Grenzen, erreicht werden können. An dieser Stelle wird ein Ausblick, über die Verbesserungen beziehungsweise Erweiterungen der graphischen Steuerungsoberfläche, gegeben. Dies könnte die graphische Steuerung der Roboter noch einfacher gestallten beziehungsweise seine Funktionalität erweitern.

Die graphische Steuerungsoberfläche implementiert nicht den gesamten Umfang an Funktionen, den die Serviceroboter anbieten. Somit kann der Funktionsumfang der Roboter in der graphischen Steuerungsoberfläche noch erweitert werden.

Die in der graphischen Steuerungsoberfläche angebotenen Steuerungselemente der Roboter sind auch in einem eigenen Plugin denkbar. Das Erteilen eines Befehls an einen Roboter könnte somit nicht nur durch das Selektieren der Roboter in der Umgebungskarte, sondern durch die Auswahl eines Tabs geschehen. Diese könnten die Funktionen und die Anzeige der Sensorenwerte eines Roboters beinhalten. Dies ist aber beim Abschluss des Projektes noch nicht möglich gewesen, weil die Namen der Plugins in der Konfigurationsdatei festgelegt werden und während der Laufzeit statisch sind. Daraus resultiert, dass mehrere gestartete Plugins, vom gleichen Typ, unter dem gleichen Namen in der Anwendung erscheinen und nicht zu unterscheiden sind.

Das Verbinden der Plugins mit den Services wird in der aktuellen Version der Anwendung manuell vorgenommen, dass zwar Rechnerressourcen schont, jedoch das Einrichten der Steuerungsoberfläche umständlich macht. Der Einrichtungsprozess könnte erleichtert werden, wenn die Steuerungsoberfläche mit einem teilautomatischen Verbindungsaufbau erweitert wird. Der automatische Aufbau könnte zum Beispiel anhand eines Parameters in der Konfigurationsdatei festgelegt werden.

Die Anbindung der Services an die grafische Steuerungsoberfläche erlaubt nur eine Verbindung mit den Roblet®-Server. Es würde sich nützlich erweisen, zu mehreren Server Verbindungen aufbauen zu können. Dies erlaubt komplexe Prozesse innerhalb einer in sich geschlossener Einheit durchzuführen, ohne auf andere Services zurückzugreifen, die die benötigten Server einbinden.

Der Serviceroboter am Arbeitsbereich TAMS beinhaltet Bewegungsfunktionen, die mit deaktivierter Kollisionskontrolle durchgeführt werden. Der Benutzer sollte vor dem Zugriff auf diese Funktionen darauf aufmerksam gemacht werden. Diese Erweiterung wäre in Form eines Warnungsdialogs denkbar.

Literaturverzeichnis

Westhoff, Zhang, Stanek, Scherer, Knoll: „Mobile Manipulatoren und ihre aufgabenorientierte Programmierung“, atp - Automatisierungstechnische Praxis 10/2004, Oldenbourg Industrieverlag GmbH, Munich, Germany, 2004

[c’t]: http://www.heise.de/ct/ftp/projekte/ct-bot/ (Datum des Zugriffs: 29.04.2006)

[genRob®]: http://genrob.com/de/ (Datum des Zugriffs: 10.05.2006)

[genRobSyst]: http://genrob.com/system/ (Datum des Zugriffs: 11.05.2006)

[Heute]: http://www.heute.de/ZDFheute/inhalt/27/0,3672,2262459,00.html (Datum des Zugriffs: 12.04.2006)

[MobRob]: http://tams-www.informatik.uni-hamburg.de/personal/westhoff/de/robotik/genRob/mobilerobot/index.html (Datum des Zugriffs: 11.05.2006)

[Philips]: http://www.philips.de/PV_Article-14779.html (Datum des Zugriffs: 5.05.2006)

[RMI-Robl]:http://tams-www.informatik.uni-hamburg.de/personal/westhoff/de/robotik/genRob/utils/index.html (Datum des Zugriffs:10.04.2006)

[Roblet®]: Westhoff, Stanek „Das genRob®-Projekt, Einführung in die Roblet® Technologie“ Version von 24.10.2005

[RobBibl]: http://roblet.org/library/2.0/doc/overview.html (Datum des Zugriffs: 8.05.2006)

[Rocon]: http://www.informatik.uni-ulm.de/rs/projekte/core/pearl99.ps (Datum des Zugriffs: 27.04.2006)

[ServRob98]: Rolf Dieter Schraft und Gernot Schmierer „Serviceroboter“

[ServRobVis04]: Rolf Dieter Schraft, Martin Hägele und Kai Wegener „Service Roboter Visionen“

[TAMS]: http://tams-www.informatik.unihamburg.de/personal/westhoff/de/robotik/-robotik.html (Datum des Zugriffs: 7.5.2006)

[TranspRob]: http://www.servus.info/servus-system/servustransportroboter/ (Datum des Zugriffs: 9.05.2006)

[Utils]: http://tams-www.informatik.uni-hamburg.de/personal/westhoff/de/robotik/genRob/utils/index.html (Datum des Zugriffs: 6.05.2006)

[WachRob]: http://www.igs-hagen.de/mosro.htm (Datum des Zugriffs: 8.05.2006)

[WeltBev]: http://www.weltbevoelkerung.de/pdf/histEntwWB.pdf
(Datum des Zugriffs:10.04.2006)

[Wiki:Eclipse]: http://de.wikipedia.org/wiki/Eclipse_%28IDE%29
(Datum des Zugriffs: 2.04.2006)

[Wiki:RMI]: http://de.wikipedia.org/wiki/Remote_Method_Invocation/
(Datum des Zugriffs: 14.04.2006)

[Wiki:SVN] http://de.wikipedia.org/wiki/Subversion_%28Software%29
(Datum des Zugriffs: 2.04.2006)

[Wiki: SWT]: http://de.wikipedia.org/wiki/Standard_Widget_Toolkit

(Datum des Zugriffs: 3.04.2006)

Erklärung

Hiermit erkläre ich, Andre Stroh, dass ich die vorliegende Baccalauriatsarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ort, Datum Unterschrift

Hiermit erkläre ich, Denis Klimentjew, dass ich die vorliegende Baccalauriatsarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ort, Datum Unterschrift

Aufteilung der Gruppenarbeit

Da diese Arbeit von Andre Stroh und Denis Klimentjew gemeinsam erstellt wurde, werden im Folgenden einzelne Kapitel dem jeweiligen Autor zugeordnet.

Vom Andre Stroh erstellte Kapitel: 3.1 - 3.11, 5.

Vom Denis Klimentjew erstellte Kapitel: 1, 2, 3.12, 4.