Bachelorarbeit

Alexandra Revout

Entwicklung einer Architektur für die Nutzung von Diensten eines Serviceroboters via Web Services

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Alexandra Revout

Entwicklung einer Architektur für die Nutzung von Diensten eines Serviceroboters via Web Services

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Angewandte Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Kai von Luck
Zweitgutachter : Dipl. Inform. Birgit Wendholt

Abgegeben am 23. August 2005

Alexandra Revout

Thema der Bachelorarbeit

Entwicklung einer Architektur für die Nutzung von Diensten eines Serviceroboters via Web Services

Stichworte

Lose gekoppelte verteilte Systeme, Web Services, mobile Anwendungen, PDA, Serviceroboter, Pioneer, Saphira

Kurzzusammenfassung

In dieser Bachelorarbeit wurde ein lose gekoppeltes verteiltes System entwickelt. Das System ermöglicht, beliebigen Clients Dienste eines halbautonomen mobilen Roboters in Anspruch zu nehmen. Für das Anbieten der Dienste des Serviceroboters werden die Web-Service-Technologien verwendet. Das System besteht aus drei wesentlichen Komponenten: einem mobilen Client, für den ein PDA verwendet wird, einem Server, der die Verwaltungsaufgaben übernimmt und einem Serviceroboter. In dieser Arbeit wurde das Design der System-Komponenten und ihrer Bestandteilen ausgearbeitet und anschließend durch die Implementierung des Systems seine Tragfähigkeit bewiesen. Die einzelnen Teile des Systems wurden dafür auf verschiedenen Plattformen und in verschiedenen Programmiersprachen realisiert.

Alexandra Revout

Title of the paper

Development of an architecture for the use of services of a service robot via web services

Keywords

Loosely coupled distributed systems, Web Services, mobile Applications, PDA, Service robots, Pioneer, Saphira

Abstract

In this B.S. Thesis a loosely coupled distributed system was developed. The system makes it possible for any client to utilize the services of a semi-autonomous mobile robot. For offering the services of the service robot the web service technologies are used. The system consists of three main components: A mobile Client, for which a PDA is used, a server, which handles the administrative tasks and a service robot. In this thesis the design of the system components and their parts was developed and afterwards its viability and stability proven by the implementation of the system. For this the individual parts of the system were realized on various platforms and in various programming languages.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In den letzten Jahren konnte man die rasche Entwicklung der Computertechnik im Bereich der mobilen Geräten wie Handy und PDA (Personal Digital Assistant (engl.)) beobachten. Die Tendenz, dass solche Geräte einen festen Platz in unserem alltäglichen Leben einnehmen werden oder schon eingenommen haben, steigt, was in erster Linie auf die ständige Verbesserung ihrer technischen Möglichkeiten zurückzuführen ist. Die kompakte Größe und bequeme Handhabung dieser Geräte spielen dabei natürlich auch eine bedeutsame Rolle. In der Abbildung 1.1 auf der Seite 6 sind die Marktzahlen von mobilen Geräten im zweiten Quartal 2004 aufgelistet (vgl. [HA04]), die dies anschaulich verdeutlichen.

Auf der anderen Seite sind die Technologien der Web Services sehr populär geworden. Mit ihrer Plattformunabhängigkeit und der Sprachneutralität bieten sie eine ausgezeichnete Umgebung für lose gekoppelte verteilte Anwendungen, in denen Clients keine Implementierungsdetails des Servers zu kennen brauchen. Im Allgemeinen werden Web Services als Informationsdienste im World Wide Web benutzt, wie z.B. der Web Service von dem Internet-Suchdienst Google [GOO], der eine Möglichkeit für Anwendungen bietet, auf die Google-Funktionalität über Web Services zuzugreifen und sie so zu nutzen. Für Web Services kann aber auch ein anderer Einsatz in Frage kommen. Dessen Konzept liegt darin, mit Hilfe von Web Services keine Informationsvermittlung zu veranlassen, sondern einen realen Dienst anbieten, der von einem mobilen Roboter verrichtet wird. Dabei sind solche Dienste gemeint, bei denen der Roboter eine konkrete Aufgabe bekommt, aber im Stande ist, sie selbst zu erledigen, wie z.B. ein Getränke-Bringservice, bei dem der Roboter ein Getränk zu der Person, die es bestellt hat, abliefern kann. Be-

EMEA-Markt für Mobile Devices			
Sprach- und Datengeräte			
Anbieter	Stückzahl Q2/04	Marktanteil Q2/04	Wachstum Q2/03-Q2/04
Nokia	1.351.400	52,4%	61,0%
HP	254.300	9,9%	64,0%
PalmOne	200.920	7,8%	-6,0%
Sony Ericsson	149.500	5,8%	-9,0%
Medion	96.150	3,7%	103,0%
Andere	527.190	20,4%	90,0%
Gesamtsegment	2.579.460	100,0%	52,0%
Datenzentrische Geräte			
HP	254.300	28,3%	64,0%
PalmOne	177.790	19,8%	-16,0%
Medion	96.150	10,7%	103,0%
Andere	370.010	41,2%	73,0%
Gesamtsegment	898.250	100,0%	43,0%
Sprachzentrische Geräte			
Nokia	1.351.400	80,4%	61,0%
Sony Ericsson	149.500	8,9%	-9,0%
Siemens	61.490	3,7%	k.A.
Andere	118.820	7,1%	83,0%
Gesamtsegment	1.681.210	100,0%	57,0%

Abbildung 1.1: Marktzahlen von mobilen Geräten

sonders sinnvoll erscheint für solche Services der Einsatz von halbautonomen Robotern. Im Gegensatz zu autonomen Robotern, die eine allgemein gestellte Aufgabe von Anfang an kennen und diese komplett selbständig ausführen, was als ihr eigenes Ziel angesehen werden kann, haben die halbautonomen Roboter keine eigenen Ziele, sie stehen für ihre Nutzer zur Verfügung und bekommen jedes Mal eine konkret gestellte Aufgabe, die sie dann allerdings selbständig bewerkstelligen.

Die grundlegende Idee dieser Arbeit besteht darin, den alltäglichen Komfort von PDA und die Möglichkeiten von Web Services zu verbinden und dies für die Kommunikation mit einem halbautonomen Roboter zu nutzen.

1.2 Zielsetzung

An der Hochschule für Angewandte Wissenschaften Hamburg (HAW) werden zahlreichen Projekte mit den Robotern Pioneer 1, Pioneer 2 und Pioneer 3 der Firma ActivMedia [AM] durchgeführt. Dabei werden diese Roboter im Bereich Großraum-Serviceroboter eingesetzt. Unter „Serviceroboter“ versteht man mobile Roboter, die Dienstleistungen für Menschen übernehmen, allgemein gesprochen: Sie können eine bestimmte Aufgabenklasse ohne menschliche Überwachung ausführen. Beispielsweise sollen monotone, gefähr-

liche oder mühselige Arbeiten autonom durchgeführt werden, wie „Material- und Werkzeugtransport in Fabriken, Bringdienste im Krankenhaus, Reinigungsaufgaben im Heim, Inspektionen unter Wasser“(vgl. [KN03]).

Abbildung 1.2: Care-O-bot I

Abbildung 1.3: Care-O-bot II

Am Fraunhofer Institut für Produktionstechnik und Automatisierung (IPA) in Stuttgart [IPA] werden z.B. mehrere Serviceroboter entwickelt. Care-O-bot ist einer von ihnen. Er kann Menschen im Haushalt unterstützen und mit ihnen interagieren (siehe Abbildungen 1.2 und 1.3). Ein anderes Beispiel für einen Serviceroboter ist der Tourbot [TB], der ursprünglich aus dem RHINO-Projekt der „Intelligent Autonomous Systems Group“ am Institut für Informatik der Universität Bonn entstand [IASG]. Dieser Roboter ist dafür konzipiert, als ein interaktiver Museumsführer zu agieren, und wird seit einigen Jahren erfolgreich im Deutschen Museum Bonn eingesetzt.

Das Ziel dieser Arbeit besteht darin, eine Architektur für ein verteiltes, lose gekoppeltes System zu entwickeln, in dem ein PDA als Client fungiert und Dienste eines Serviceroboters über Abruf eines bestimmten Web Services in Anspruch nimmt. Der Client führt dabei keine Fernsteuerung des Roboters durch, sondern spricht ihn als einen halbautonomen Roboter an (siehe voriger Abschnitt). Der Zugriff auf die Funktionen eines Serviceroboters ist in der Regel nur über eine spezielle Roboter-Software möglich. Das zu entwickelnde System soll von dem konkreten Roboter-Zugriff abstrahieren

und so die Möglichkeiten für die Nutzung der Roboter-Dienste von beliebigen Client-Geräten aus bieten. Darüber hinaus soll das System ermöglichen, dass mehrere Clients den vom Roboter zur Verfügung gestellten Service in Anspruch nehmen können. Das System soll folglich die Multiuser-Fähigkeit garantieren. Web Services eignen sich dabei besonders gut als Kommunikationsmöglichkeit, da sie plattformunabhängig und interoperabel sind und so die lose Kopplung ermöglichen. Außerdem bieten sie von sich aus eine Ortsunabhängigkeit für den Client, da die Kommunikation mit Web Services über standardisierte Internet-Protokolle abläuft. Nicht zuletzt spricht die Einfachheit der Umsetzung von Web Services für sie. Sie weisen aber auch einen Nachteil auf: Die Kommunikation über Web Services ist für die Echtzeit-Übermittlung zu langsam. Da der Serviceroboter jedoch für solche Dienste eingesetzt werden soll, bei denen keine Echtzeit-Übermittlung notwendig ist, wie z.B. die Ablieferung von bestellten Getränken, ist dieser Aspekt für das zu entwickelnde System irrelevant.

Die Entwicklung des Serviceroboters selbst steht in dieser Arbeit nicht im Vordergrund und gehört nicht zu ihren Schwerpunkten.

1.3 Gliederung der Arbeit

Am Anfang des Kapitels „Analyse“ (2) wird eine Einführung in die moderne mobile Robotik gegeben, danach wird die Aufgabenstellung anhand eines Beispielszenarios definiert. Anschließend wird das zu entwickelnde System auf seine Bestandkomponenten und die Art der Verknüpfung zwischen ihnen analysiert. Letztendlich werden in diesem Kapitel Technologien und Hardware beschrieben, die für die Lösung der gestellten Aufgaben relevant sind, darüber hinaus wird die Wahl getroffen, welche von ihnen in der Arbeit eingesetzt werden.

Im Kapitel „Design und Realisierung“ (3) werden der Architektur-Entwurf des zu entwickelnden Roboter-Service-Systems und seine Komponenten vorgestellt, basierend auf dem in der Analyse beschriebenen Szenario und den gewählten Technologien. Abschließend werden die bei der Umsetzung des Entwurfs angewendeten Realisierungskonzepte beschrieben.

Im Kapitel „Zusammenfassung und Ausblick“ (4) werden die Ergebnisse der Arbeit vorgestellt und analysiert. Anschließend wird auf die Möglichkeiten der Weiterentwicklung des Roboter-Service-Systems eingegangen und es werden Vorschläge für System-Erweiterungen gemacht.

Kapitel 2

Analyse

2.1 Mobile Robotik

Unter Robotik wird eine wissenschaftliche Disziplin verstanden, die sich mit der Entwicklung und der Steuerung von Robotern beschäftigt.

Roboter sind heute aus der menschlichen Umgebung nicht weg zu denken. Sie werden in verschiedensten Bereichen eingesetzt, wie Industrie, Medizin oder auch Unterhaltung (siehe Abbildung 2.1). Ein bedeutendes Teilgebiet

Abbildung 2.1: AIBO von Sony [AIBO]

Abbildung 2.2: Mobiler Roboter „Arthur“

der Robotik ist die Forschung mit mobilen Robotern. Auf der Abbildung 2.2 ist ein Beispiel für einen solchen Roboter zu sehen. Er heißt Arthur und bildet die praktische Grundlage für das Forschungsprojekt „Sinnesorgane für mobile

Roboter“ der Fakultät für Informations- und Kognitionswissenschaften an der Universität Tübingen [ARTH].

Es ist zwischen vier Systemarchitekturen für Roboter zu unterscheiden: deliberative, reaktive, verhaltensbasierte und hybride Architekturen [MA05]. Für einzelne mobile Roboter sind eher hybride Architekturen geeignet, da „sie versuchen die besten Elemente der reaktiven und der deliberativen Architekturen zu kombinieren“ ([MA05]). Nach der Art, eine Aufgabe zu lösen, werden mobile Roboter allgemein in drei Kategorien unterteilt [SP05]:

- extern gesteuerte Roboter, für die jede Aktion als Anweisung kommt (z.B. *nach links*, *geradeaus*)
- halbautonome Roboter, die bestimmte komplexe Aktionen selbst ausführen können, die Anweisungen lauten in diesem Fall wie z.B. *fahre in einen bestimmten Raum*, der Roboter findet den Weg dorthin selbständig
- autonome Roboter, die völlig selbständig eine allgemein beschriebene Aufgabe lösen, z.B. *alle Gegenstände vom Fußboden sammeln*

In Abhängigkeit davon, wofür der Roboter eingesetzt werden soll, erscheint die eine oder eine andere Roboter-Kategorie am sinnvollsten. Als Beispiel kann hier der Einsatz von extern gesteuerten Robotern nach der Reaktorkatastrophe in Tschernobyl [TS01] angeführt werden. Die halb- (oder teil-) autonomen Roboter können dagegen dort verwendet werden, wo die Fernsteuerung des Roboters, aber auch die volle Autonomie, unerwünscht oder unmöglich ist. Der Roboter, der auf dem Mars eine Mission erfüllen soll, ist ein Beispiel dafür. Da Echtzeit-Steuerung in diesem Fall realitätsfern ist, kommt nur selbständige Arbeit des Roboters in Frage. Die Anweisungen, was er genau zu tun hat, bekommt er trotzdem in periodischen Abständen, z.B. jede Woche[1]. Danach richtet er dann sein Verhalten (siehe auch [NASA]). Die autonomen Roboter werden beim „Roboter-Fußball“ verwendet. Hier haben sie ein allgemeines Ziel: das Spiel zu gewinnen. Wie sie zu diesem Ziel kommen, wird nicht über Anweisungen gesteuert, sondern ist ihnen selbst überlassen (für weitere Informationen siehe auch [RC05]).

[1] Roboter, die Aufträge abarbeiten, werden auch Agenten genannt

2.2 Aufgabenstellung

2.2.1 Beispielszenario

Die Untersuchung und die Entwicklung der Architektur soll anhand eines Beispielszenarios erfolgen.

Als Grundlage für diese Arbeit wurde folgendes Szenario ausgewählt. Es wird angenommen, dass im Gebäude der HAW ein Service zur Verfügung steht. Dieser Service besteht darin, dass Studenten und Professoren (im Weiteren als Nutzer bezeichnet) gewisse Dienste eines autonomen Roboters in Anspruch nehmen können. Als ein Dienst kann das Bringen von verschiedenen Gegenständen angesehen werden, z.B. einer neuen Schachtel Tafelkreide oder einer Tasse Kaffee. Dieser Dienst kann mittels der den Nutzern zur Verfügung stehenden PDAs und einer darauf installierten Anwendung in Anspruch genommen werden, indem der gewünschte Service ausgewählt und aufgerufen wird. Der Service-Nutzer erhält darauf eine kurze Information darüber, ob die Anfrage angenommen wurde. Im Weiteren steht ihm die Möglichkeit zur Verfügung, den Status des Fortschritts der Anfrage abzurufen. Als Antwort erhält der Nutzer eine Auskunft darüber, ob sein Auftrag schon in der Bearbeitung ist und vom Roboter ausgeführt wird. Letztendlich kommt der Serviceroboter zum Benutzer mit der gewünschten „Bestellung“ an.

Als Ausgangspunkt für diese Arbeit wurde die Studienarbeit von Florian Boldt und Hauke Löhler [BL99] genommen. In ihrer Arbeit haben sie einen einfachen Büroboten SPOT entwickelt und seine Steuerung über eine Saphira-Client/Server-Anwendung realisiert. Auf den Abbildungen 2.3 und 2.4 sind der Bürobote in Aktion und seine Anwendungsumgebung zu sehen. In der von ihnen entwickelten Anwendung wird eine Remote-Verbindung zu

Abbildung 2.3: Roboter SPOT

Abbildung 2.4: Umgebung

dem Steuerrechner des Roboters aufgebaut. Der Austausch von Daten zwischen dem Client und dem Server erfolgt dabei über TCP-Sockets. Die grobe Skizze der entsprechenden Architektur ist in der Abbildung 2.5 dargestellt.

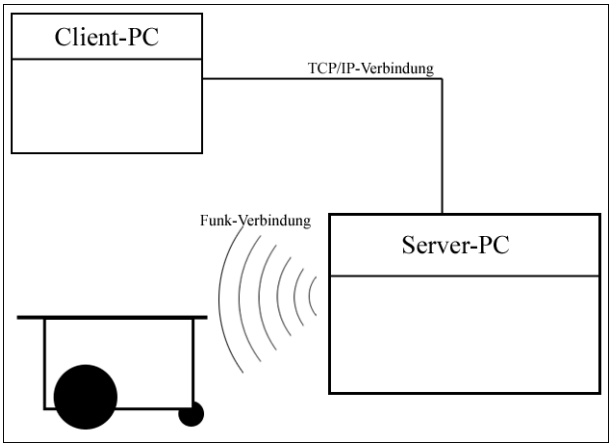

Abbildung 2.5: Architektur der Saphira-Client/Server-Anwendung von Florian Boldt und Hauke Löhler

Im Gegensatz zur Studienarbeit von Boldt und Löhler soll in dieser Arbeit die Kommunikation zwischen dem Client und dem Server auf der höheren Ebene, der Applikationsschicht, mittels Web Services stattfinden. Dabei soll die Funktionalität des Servers über eine einfache Steuerung des Roboters hinausgehen und auch Verwaltungsaufgaben beinhalten.

2.2.2 Analyse der Aufgabe

Wie aus dem oben beschriebenen Szenario ersichtlich ist, soll in dieser Arbeit ein verteiltes Roboter-Service-System entwickelt werden, das den Nutzern an der HAW zur Verfügung stehen wird. Aus dem Szenario können folgende Anwendungsfälle für das zu entwickelnde System abgeleitet werden:

- Aufgabe eines neuen Auftrages
- Anfrage über den Fortschritt der Bearbeitung des aufgegebenen Auftrages

In dem Diagramm 2.6 Seite 13 sind diese Anwendungsfälle anschaulich dargestellt.

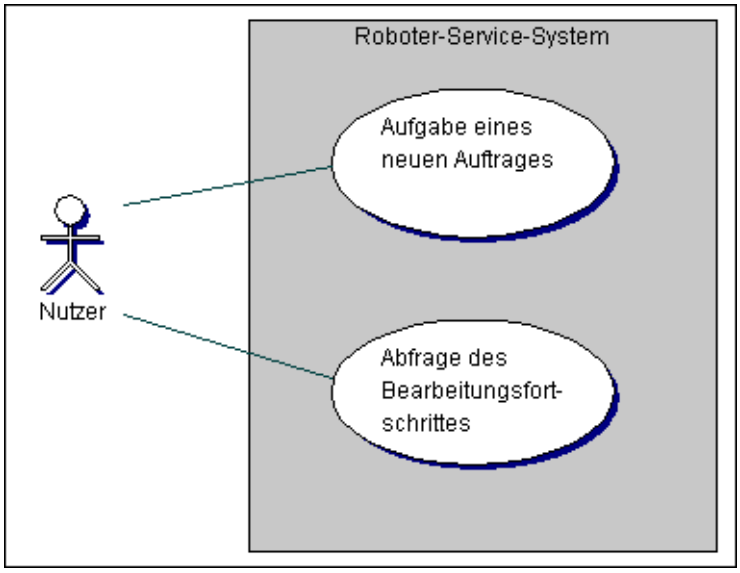

Abbildung 2.6: Anwendungsfälle

2.2.3 Analyse der Systemkomponenten und Anforderungen an sie

Aus dem Szenario wird erkennbar, dass das System mindestens aus drei Komponenten bestehen soll, an die folgende aus dem Szenario abgeleitete Anforderungen gestellt werden:

- PDA-Anwendung für den Nutzer:
  - Über die PDA-Anwendung wird der Nutzer von dem zur Verfügung stehenden Service Gebrauch machen können. Sie stellt also eine Schnittstelle zum Roboter-Service-System für den Nutzer dar.
- Server, der den Service anbieten wird:
  - Der Server ist das Verbindungsglied zwischen dem Client (der PDA-Anwendung) und dem Roboter. Er soll folgende Aufgaben übernehmen:
    - die Annahme der Aufträge für den Roboter;
    - die Übernahme seiner Steuerung, da keine Möglichkeit für den Client besteht, den Roboter direkt anzusprechen;
    - die Multiuser-Fähigkeit des Roboters, da wie aus dem Beispielszenario ersichtlich wird, mehrere Nutzer den Service in Anspruch nehmen können.

- Serviceroboter:
  - Der Roboter soll die an ihn gestellten Aufträge ausführen.

2.3 Analyse des zu entwickelnden Systems

In diesem Abschnitt wird eine Analyse des zu entwickelnden Systems in Bezug auf die Art der Kopplung des Systems durchgeführt, d.h. die Art der Verknüpfung, in der die einzelnen Komponenten des verteilten Systems zueinander stehen. Der Grad der Kopplung zwischen den Komponenten eines Systems (oder den Anwendungen, aus denen ein System besteht) kann unterschiedlich sein. Zwei Extreme bilden in diesem Sinne lose gekoppelte Systeme und eng gekoppelte Systeme. Zwischen diesen zwei Kategorien gibt es „Übergangsstufen“, in welche die Systeme gehören, die nicht streng eng oder lose gekoppelt genannt werden können.

Das Ziel dieser Arbeit ist die Entwicklung eines verteilten Roboter-Service-Systems, dessen Komponenten möglichst lose gekoppelt zueinander stehen. Deswegen wird im Folgenden zuerst der Begriff der lose gekoppelten Systeme konkretisiert. Im Laufe der Recherche hat sich herausgestellt, dass keine feste Definition dieses Begriffes existiert. Er wird in verschiedenen Kontexten und unter verschiedenen Sichtweisen auf ein verteiltes System gebraucht, angefangen mit der Betriebssystem-Sicht bis zur Anwendungsebenen-Sicht und der Abhängigkeit ihrer Komponenten untereinander. Dazu kommt noch, dass jeder, der diesen Begriff benutzt, es auf eigene Weise tut und für ihn oft eine verschwommene Erklärung gibt. Nach mehreren Definitionen aber kann eine der wichtigsten Eigenschaften der lose gekoppelten Systemen extrahiert werden. Nach [BR03] ist ein lose gekoppeltes System dadurch gekennzeichnet, dass die Teile dieses Systems unabhängig voneinander agieren und doch in einem notwendigen Maße interagieren können. Eine andere Beschreibung besagt, dass in einem lose gekoppelten System Komponenten nur bei (gelegentlicher) Nutzung in Verbindung zueinander stehen [HA05]. Also kann die Unabhängigkeit der Systemkomponenten voneinander und daraus folgend der hohe Grad ihrer Selbständigkeit als bedeutende Eigenschaft der losen Kopplung definiert werden. Deswegen wird das zu entwickelnde System in weiteren Abschnitten auf die Abhängigkeit der Systemkomponenten voneinander untersucht. Dabei wird dies von zwei Gesichtspunkten aus durchgeführt. Zuerst wird das System auf der funktionalen Ebene analysiert, in Bezug darauf, wie die Funktionalität der einzelnen Komponenten voneinander abhängt. Danach wird das Systems aus der Sicht der technischen Möglichkeiten für die Realisierung einer losen Kopplung untersucht.

2.3.1 Funktionale Ebene

Aus der funktionalen Sicht kann eine lose Kopplung folgendermaßen definiert werden: In lose gekoppelten Systemen kann eine einzelne Komponente im hohen Maße ihre Funktionalität weiter gewährleisten, auch wenn eine andere Komponente ausfällt, da von ihr keine tragende Funktionalität der ersten Komponente abhängt. Im Sinne dieser Überlegungen wird das in dieser Arbeit zu entwickelnde verteilte System analysiert. Wie in den vorigen Abschnitten beschrieben wurde, soll das System mindestens aus drei Komponenten bestehen[2]: dem PDA-Client, dem Server und dem Serviceroboter. Aus den im Abschnitt 2.2.3 beschriebenen Verantwortungsbereichen dieser System-Bestandteile geht hervor, dass der PDA-Client eigentlich dafür konzipiert werden soll, den Aufruf des Services zu tätigen, also ist diese Komponente auf den Server angewiesen, der diesen Service anbietet, und kann nicht agieren (also ihre Aufgabe erledigen), wenn der Service nicht zur Verfügung steht. Das bedeutet, dass, streng genommen, keine lose Kopplung zwischen diesen zwei Teilen des Systems besteht. Die Kopplung kann allerdings verringert werden, indem die Client-Anwendung bei einem Server-Ausfall die Möglichkeit erhält, dynamisch auf ein anderes Service-System „umzusteigen“. Bei einem Getränke-Bestellservice könnte dies z.B. ein E-Mail-System für ein menschliches Bedienungspersonal sein.

Der Server steht seinerseits in Beziehung zu dem Serviceroboter und ist auf dessen Funktionalität angewiesen, also gibt es hier auch keine lose Kopplung, denn ohne den Serviceroboter kann kein funktionierender Service angeboten werden. Aber auch hier können die Abhängigkeiten verringert werden, indem bei einem Roboter-Ausfall die Service-Aufforderungen z.B. in einer Message-Queue gesammelt werden, bis der Roboter wieder zur Verfügung steht. Die ankommenden Aufträge könnten vom Server auch an eine andere Ausführungsform umgeleitet werden. Das könnte wieder eine Person aus dem „menschlichen“ Servicepersonal sein.

Das System kann also aus der funktionalen Sicht nicht als lose gekoppeltes System genannt werden. Es ist aber auch kein eng gekoppeltes System, da mit den richtigen Ansätzen die Abhängigkeiten zwischen den Komponenten verringert werden können.

2.3.2 Realisierungsebene

Als Weiteres soll das System auf die Kopplung auf der Realisierungsebene untersucht werden. In diesem Kontext bedeutet „Kopplung“ das Ausmaß

[2]Das tatsächliche Design der Architektur wird im Kapitel 3 „Design und Realisierung“ ausführlich behandelt

von Änderungen, die an einer Anwendung (Komponente) durchgeführt werden müssen, wenn eine andere geändert wird. Aus dieser Sicht soll eine lose Kopplung keine Notwendigkeit solcher Anpassungen bedeuten. Das kann über ein funktional gutes Schnittstellen-Design der Systemkomponenten und den Einsatz von so genannten Middleware-Technologien erreicht werden, die für die Realisierung des Verteilungskonzeptes eines Systems eingesetzt werden.

2.3.2.1 Übersicht über die modernen Middleware-Technologien

Eine der bekanntesten Middleware-Architekturen, die beim Aufbau eines verteilten Systems angewendet werden, ist CORBA (Common Object Request Broker Architecture), die auf dem Paradigma der verteilten Objekte und der Kommunikation über wohldefinierte, schmale Schnittstellen basiert. CORBA gilt als ein ausgereifter Standard, der schon in mehreren Implementierungen existiert. Allerdings ist er recht kompliziert und wird nur von Teilen der Softwareindustrie unterstützt und setzt ein hohes Maß der Anpassung der eigenen Software-Architektur an die CORBA-Architektur voraus [DEGE]. Für weitere Informationen über CORBA wird an dieser Stelle auf [TAN2] verwiesen.

Eine andere bekannte Middleware für verteilte Systeme ist DCOM (Distributed Component Object Model) von Microsoft, deren Funktionalität von Windows-Anwendungen verwendet wird (für weitere Informationen siehe [TAN2]). Allerdings wurde DCOM von Microsoft zu Gunsten von .NET-Remoting aufgegeben, dessen Einsatz sich auf die .NET-Anwendungen beschränkt [HA04]. Wie sich schon vermuten lässt, ist diese Technologie nicht plattformunabhängig, sie wird in Windows-Umgebungen verwendet. Seit Kurzem aber stimmt das auch nicht mehr genau, da mit dem Open-Source-Projekt Mono eine plattformunabhängige Implementierung von .Net geschaffen wurde (siehe auch [MONO]). Dieses Projekt wurde von Novell 2003 übernommen und hat das Ziel gesetzt, die Möglichkeiten des .NET-Frameworks auch für Linux und andere Umgebungen zu öffnen [HE05].

Die dritte nennenswerte Technologie ist Java RMI (Java Remote Method Invocation) [RMI], die wie CORBA auf dem Konzept der verteilten Objekte basiert (siehe auch [CO02] und [TAN2]). Java RMI ist zwar plattformunabhängig, was durch die Java-Programmiersprache garantiert wird, aber nicht sprachunabhängig. Sie kann, wie der Name schon sagt, nur in Java-Umgebungen eingesetzt werden, was gewisse Einschränkungen mit sich bringt. Abhilfe bringt hier Java RMI-IIOP (Java Remote Method Invocation over Internet Inter-ORB Protocol) [RMI-IIOP], die die Interoperabilität der Java RMI mit der CORBA-Welt schafft.

Letztendlich können hier die Technologien der Web Services als Kommunikationsansatz zwischen verteilten Systemkomponenten aufgeführt werden. Web Services sind im Gegensatz zu CORBA, DCOM und RMI nicht objektbasiert, sondern dienstorientiert. Sie unterstützen keine verteilten Objekt-Referenzen, was die Serialisierung gegenstandslos macht. Web Services stützen sich auf etablierte Technologien. Das und die Tatsache, dass ihre Protokolle und Standards nicht auf binären Formaten, sondern auf XML basieren, macht ihre Interoperabilität aus. Web Services versprechen Sprach- und Plattformneutralität und sind sehr einfach zu handhaben. Im Abschnitt 2.4.3 werden diese Technologien ausführlich beschrieben.

Jede der beschriebenen Technologien kann die Entwicklung von verteilten Systemen intensiv unterstützen, indem sie eine nötige Transparenz und Unabhängigkeit der einzelnen Komponenten schafft. Mit dem Einsatz einer dieser Technologien wird jedoch nicht automatisch eine lose Kopplung garantiert. Dazu müssen bestimmte Richtlinien beim Design eingehalten werden, wie z.B. dynamische Bindung zur Laufzeit. Damit ist allerdings ein gewisser Overhead verbunden, da in diesem Fall die Komponente das Auffinden des benötigten entfernten Objektes oder Dienstes und dessen Anbindung übernehmen soll, was nicht immer wünschenswert oder realisierbar ist, besonders in sehr komplexen Systemen [DEGE]. Werden aber solche Richtlinien eingehalten, kann ein verteiltes System aus der Realisierungssicht als ein lose gekoppeltes System genannt werden.

2.3.2.2 Analyse der Abhängigkeiten zwischen den Systemkomponenten

Zwischen der PDA-Anwendung und dem Server wurden Web-Services-Technologien als Kommunikationsmöglichkeit gewählt, da die Gesamtheit ihrer Eigenschaften sie besonders attraktiv für den Einsatz im zu entwickelnden Roboter-Service-System macht. Mit der Verwendung von Web Services und dem richtigen Design des PDA-Clients und des Servers (siehe vorigen Abschnitt) kann eine lose Kopplung zwischen diesen beiden Komponenten gebildet werden.

Zwischen dem Server und dem Serviceroboter besteht eine andere Art der Verknüpfung. Die Kommunikation mit dem Serviceroboter erfolgt über eine spezielle Roboter-Software, die dem Benutzer – in diesem Fall dem Server – bestimmte Funktionen zur Verfügung stellt. Von der konkreten Abhängigkeit kann also in diesem Fall wegen der Spezifik der Roboter-Software nicht abstrahiert werden. Es besteht dementsprechend keine lose Kopplung zwischen diesen zwei Systemkomponenten. Wenn allerdings die Kommunikation zwischen den beiden Komponenten über wohldefinierte, schmale Schnittstellen

einer zusätzlichen Zwischenschicht stattfinden wird, die für eine gewisse Abstraktion zwischen dem Server und dem Roboter sorgt, wird hier auch keine enge Kopplung entstehen.

2.4 Analyse der zur Verfügung stehenden Technologien

2.4.1 PDA

Mittlerweile gibt es auf dem Markt eine große Anzahl an PDA-Herstellern, die für ihre Geräte verschiedene Betriebssysteme einsetzen. Die bekanntesten sind z.B. Newton OS, PalmOS oder Microsoft Pocket PC (aktuelle Versionen werden auch als *Windows Mobile* bezeichnet).

Im Fall dieser Arbeit ist es wichtig, dass der PDA über eine Wireless-Funktion verfügt, die für die Kommunikation des mobilen Gerätes notwendig ist.

Im Roboter-Labor des Fachbereiches Elektrotechnik und Informatik der HAW Hamburg stehen PDAs HP iPAQ h5550 von der Firma Hewlett Packard [HP] zur Verfügung, mit dem Betriebssystem Microsoft Pocket PC 2003 darauf. Darüber hinaus verfügt der HP iPAQ h5550 über zwei Standardschnittstellen für eine funkbasierte Kommunikation: Bluetooth und WLAN 802.11b. Damit erfüllt dieser PDA die gestellten Anforderungen und wird in dieser Arbeit als mobiles Gerät eingesetzt. Die Client-Anwendung wird speziell für das auf dem PDA installierte Betriebssystem Pocket PC 2003 entwickelt. Die Frage ihrer Kompatibilität zu anderen Betriebssystemen wird hier absichtlich außer Acht gelassen, weil dies nicht zu den übergeordneten Zielen dieser Arbeit zählt.

2.4.2 Bluetooth, Wireless LAN

Da der einzusetzende PDA zwei Möglichkeiten für eine funkbasierte Kommunikation besitzt, werden in den folgenden Abschnitten die beiden Technologien kurz vorgestellt und die Wahl getroffen.

2.4.2.1 Bluetooth

Bluetooth ist ein Industriestandard für die drahtlose Vernetzung von mobilen Kleingeräten und auch Computern und Peripheriegeräten. Er gehört zu den Wireless Personal Area Networks (WPAN), was die Kurzstrecken-Funktechnik bezeichnet. Das Bluetooth-Modul, die Basis eines Bluetooth-

Systems, verbraucht wenig Strom und sendet im lizenzfreien ISM-Band (Industrial-, Scientific-, Medical-Band) zwischen 2,402 GHz und 2,480 GHz. Es gibt drei Klassen von Bluetooth-Geräten, die über ihre Sendeleistung und entsprechend die Reichweite definiert sind: Klasse I: 100 mW - 100 m Reichweite, Klasse II: 2,5 mW - 10 m und Klasse III: 1 mW - 10 cm. Gegenwärtig wird in PDAs meistens Bluetooth der Klasse II eingesetzt, auch in HP iPaq H5550, da es weniger Energie verbraucht. Die typische Anwendung von Bluetooth in den PDAs ist der Datenabgleich mit dem PC oder die Kommunikation mit einem Headset.

2.4.2.2 Wireless LAN

Wireless LAN oder kurz WLAN ist ein drahtloses lokales Netzwerk. Der verbreitetste Standard ist dabei die IEEE (Institute of Electrical and Electronics Engineers) [IEEE] 802.11-Familie. WLAN kann in zwei Modi betrieben werden: im Infrastruktur-Modus und im Ad-hoc-Modus. Im Infrastruktur-Modus existiert eine Basisstation, ein sogenannter Wireless Access Point, der für die Koordination der WLAN-Knoten und die Verbindung mit anderen Netzen zuständig ist. Im Ad-hoc-Modus sind alle WLAN-Knoten gleichberechtigt. Die Reichweite eines WLAN-Endgerätes beträgt 30 - 100 m, die Sendeleistung 100 mW.

2.4.2.3 Entscheidung in der Auswahl

Ein Vorteil des in PDAs eingesetzten Bluetooth gegenüber dem WLAN besteht im deutlich niedrigeren Energieverbrauch. Aber das hat auch seine Nachteile, da die Reichweite dabei auch deutlich kleiner ist. Dementsprechend verspricht die Nutzung von WLAN größere Freiheit und bessere Möglichkeiten für den Einsatz von PDAs als Consumer des an der HAW zur Verfügung stehenden Roboter-Services. Daraus folgend erscheint die Nutzung des WLANs in dieser Arbeit am sinnvollsten.

2.4.3 Web Services

2.4.3.1 Was sind Web Services

Web Services sind Dienste, auf die mit standardisierten Internet-Protokollen zugegriffen werden kann. Unter Web Services versteht man allgemein eine neue Art von Technologien für verteilte Anwendungen, mit denen sich eine service-orientierte Architektur (kurz SOA) [KR04] implementieren lässt. Web Services kennzeichnet vor allem die Interoperabilität, die durch die Verwendung von XML (Extensible Markup Language) [XML] als Datendarstel-

lungsschicht erreicht wird. Darüber hinaus charakterisieren Web Services unter anderem folgende Eigenschaften:

- lose Kopplung, was die Flexibilität und Wartbarkeit der Systemen erhöht;
- grobe Granularität der Dienste für bessere Performance der Geschäftsprozesse;
- sie basieren auf etablierten Standards, was eine einfache Anwendung und eine unkomplizierte Integration von Web Services ermöglicht;
- sie unterstützen keine verteilte Objekt-Referenzen;
- sie geben die Möglichkeit zur dynamischen Lokalisation und Einbindung eines Dienstes.

Ursprünglich basierte die Nachrichten-Übermittlung für Web Services auf RPC [TAN2], genau gesagt auf dem Protokoll XML-RPC, der von Dave Winer [DW], dem Gründer der Firma UserLand Software Inc., 1998 entwickelt wurde. Dieses Protokoll realisiert RPC über HTTP, als Nachrichten-Format wird dabei XML verwendet. Der große Vorteil des XML-RPC liegt in der Einfachheit dieses Protokolls, aber genau deswegen reicht XML-RPC oftmals nicht mehr aus (für weitere Informationen siehe [HA04]). Mittlerweile hat sich SOAP gegenüber XML-RPC durchgesetzt und fungiert als *der* Standard der Nachrichten-Übertragung für Web Services. Auch SOAP beherrscht die RPC-basierte Übermittlung, allerdings bleibt es nicht darauf beschränkt und unterstützt ebenfalls den Dokumentenaustausch.

Web Services stützen sich auf drei zentrale standardisierte Technologien:

- SOAP[3] als Kommunikationsprotokoll für Web-Service-Anwendungen, das den Aufbau und das Format von Nachrichten bei einem Web-Service-Aufruf definiert;
- WSDL (Web Service Description Language) für die standardisierte Beschreibung von Schnittstellen eines Web Services mit der Definition der Methoden des Web Services, deren Parameter, Transportmechanismen und Nachrichtenformaten;
- UDDI (Universal Description, Discovery and Integration) als weltweiter Standard für Verzeichnisse von Web Services, in denen die Web Services registriert, identifiziert und verwaltet werden.

[3]Der Name stand ursprünglich für Simple Object Access Protocol, ab SOAP 1.2 stellt es kein Akronym mehr dar.

Alle drei Technologien sind XML-basiert und ermöglichen somit die Interoperabilität von Web Services.

2.4.3.2 Möglichkeiten zur Umsetzung

Für die Realisierung eines Web Services stehen mehrere Möglichkeiten zur Verfügung. Zurzeit sind zwei Zweige der Web-Service-Entwicklungsplattformen sehr populär. Zum einen ist es das .NET-Framework von Microsoft, das eine eigene Umgebung für die Entwicklung von Web Services mit Hilfe von solchen Programmiersprachen wie Visual Basic, C++ oder C# anbietet. Den anderen Zweig bilden mehrere Plattformen, die auf Java basieren, sowohl kommerzielle Produkte als auch Open-Source-Projekte. Als Beispiele können hier folgende Plattformen für die Entwicklung und das Betreiben von Web Services aufgeführt werden: WebSphere-Plattform von IBM, Java Web Services Developer Pack von Sun Microsystems oder WebLogic von BEA Systems. Als wichtiger Vertreter der Open-Source-Gemeinde im Bereich der Web-Service-Entwicklung wird das Axis-Projekt von Apache Software Foundation angesehen. Axis ist die Implementierung des SOAP-Standards, die die Entwicklung von Clients und Servern, weiteren Bibliotheken und Werkzeugen und das Hosting von Web Services ermöglicht. Für den server-seitigen Einsatz von Axis wird ein Servlet- und JSP-fähiger Web-Container benötigt, um die Möglichkeit zu haben, Web Services zur Verfügung zu stellen.

Für die Demonstration der Interoperabilität der Web Services werden in dieser Arbeit zwei verschiedene Web-Service-Plattformen benutzt. Für die Entwicklung des Web-Service-Clients wird .NET-Framework eingesetzt. Diese Plattform ist im Bereich der Anwendungsentwicklung für die mobilen Pocket-PC-Geräte besonders ausgereift und bietet wie keine andere umfangreiche Möglichkeiten und sehr hohen Komfort für die Erstellung von mobilen Anwendungen, einschließlich Web-Service-Clients. So verfügt die .NET-Entwicklungsumgebung z.B. über einen Pocket-PC-Simulator, der die Möglichkeit gibt, die zu entwickelnde Anwendung jederzeit problemlos zu testen.

Für die Entwicklung des Providers[4] wird in dieser Arbeit Axis benutzt. Dieses Framework weist sehr gute Qualität auf und bietet umfangreiche Features. Es wird schon in vielen kritischen Praxisprojekten erfolgreich eingesetzt. Außerdem sprechen für Axis dieselben Gründe, die allgemein für den Einsatz von Open-Source-Software sprechen: Unabhängigkeit vom Hersteller, Code-Qualität, Interaktion mit den Entwicklern und eine schnelle Reaktion auf Fehlermeldungen (vgl. [WA04]). Als Laufumgebung für Axis wird

[4]Ein Service Provider implementiert einen Web Service und stellt ihn im Netz zur Verfügung.

der Apache Tomcat Web-Container verwendet, der ebenso ein Open-Source-Projekt der Apache Software Foundation ist.

2.4.4 Pioneer, Saphira, Colbert

Wie in der Einleitung schon erwähnt wurde, verfügt die HAW Hamburg über mehrere Roboter Pioneer 1, Pioneer 2 und Pioneer 3 von der Firma ActivMedia [AM]. Pioneers sind kleine, intelligente mobile Roboter, deren Architektur ursprünglich von Dr. Kurt G. Konolige [PM99] entwickelt wurde. Sie besitzen alle Grundkomponenten für die Navigation in einer realen Umgebung, einschließlich Batterie, Antriebsmotoren und Räder, Positions- und Geschwindigkeits-Geber, Sensoren und andere Accessoires. Diese Komponenten werden von dem Bord-Mikrocontroller des Roboters und der Roboter-Server-Software gesteuert. Der Roboter-Server ist eine Plattform für mobile Roboter, die eine Reihe von Dienstleistungen in einem standardisierten Format zur Verfügung stellt. Er ist für die „low-level"-Operationen verantwortlich: das Betreiben der Motoren, basierend auf den Kommandos des Clients [5], und die Abfrage der Sensoren und das Senden der Resultate dieser Abfragen zum Client.

In dieser Arbeit wird die Client-Software auf einem entfernten Host-PC laufen und mit dem Roboter-Server mit Hilfe eines Paares Radio-Modems kommunizieren (siehe Abbildungen 2.7 und 2.8).

Abbildung 2.7: Pioneer 2 mit dem angeschlossenen Radio-Modem

Abbildung 2.8: Pioneer 2: Saphira-Verbindung über Radio-Modems

[5] Unter dem Client wird in diesem Abschnitt eine Steuerungssoftware für den Roboter verstanden.

Beim Pioneer-Roboter gibt es aber auch die Möglichkeit, den Client auf einem auf dem Roboter aufgestellten Laptop laufen zu lassen, angeschlossen über ein Kabel, wie auf den Abbildungen 2.9 und 2.10 zu sehen ist. Wo sich die Client-Software befindet, hat dabei keine Aussagekraft über die Autonomie des Roboters. Die Autonomie wird unabhängig davon definiert (siehe dazu Abschnitt 1.1).

Abbildung 2.9: Pioneer 3 mit Saphira auf dem Laptop: Vorderansicht

Abbildung 2.10: Pioneer 3 mit Saphira auf dem Laptop: Rückansicht

An der HAW Hamburg steht zum Agieren mit dem Pioneer-Roboter-Server der Saphira-Client zur Verfügung. Saphira ist eine Software-Umgebung für die Steuerung von mobilen Robotern und für die Entwicklung von Client-Applikationen für diese. Ihre Architektur gehört zu den hybriden Systemarchitekturen für Roboter (siehe Abschnitt 1.1) und wurde im Artificial Intelligence Center [AIC] am SRI International unter der Leitung von Dr. Kurt G. Konolige entwickelt [KO96]. Das Saphira-System besteht aus zwei Architekturen, die aufeinander aufbauen. Die *System-Architektur* beinhaltet Routinen für die Kommunikation mit und das Controlling des Roboters von einem Host-Computer. Diese Architektur dient zur unkomplizierten Erstellung von Client-Robot-Anwendungen. Die *Robot-Control-Architektur* ist das Interface zum Roboter. Sie umfasst das Design für die Kontrolle des mobilen Roboters auf hardware-naher Ebene im Bereich der mit der Navigation verbundenen Probleme, ausgehend von der Kontrolle von Motoren und Sensoren bis zur Planung und Erkennung von Gegenständen.

Zum Erstellen von Roboter-Controlling-Programmen unterstützt Saphira die einfache, C-ähnliche Programmiersprache Colbert. Die Semantik von Colbert basiert auf der Theorie der endlichen Automaten. Der Grundkonstrukt der Sprache sind so genannte *Aktivitäten*, Prozeduren, die eine Reihe

von koordinierten Handlungen beschreiben, die der Roboter durchführen soll. Eine der wichtigsten Eigenschaften von Colbert ist die Nebenlaüfigkeit der Prozesse, was für die komplizierte Steuerung eines Roboters in einer realen Umgebung von großer Bedeutung ist. Mehrere Aktivitäten werden in der Regel gleichzeitig ausgeführt. Sie haben eine hierarchische Struktur und können miteinander kommunizieren, außerdem kann jeder Aktivität eine Priorität zugewiesen werden. Diese Merkmale ermöglichen die Koordination von nebenläufigen Aktivitäten.

Für detaillierte Informationen zu Saphira und Colbert wird an dieser Stelle auf offizielle Dokumentationen hingewiesen: [SA99] und [CO99] von Kurt G. Konolige.

2.4.5 Problem der gemeinsamen Ressourcen-Nutzung

Das am Anfang des Kapitels beschriebene Beispielszenario sieht unter anderem die Möglichkeit vor, dass mehrere PDA-Clients den Roboter-Dienst in Anspruch nehmen können. Eines der wichtigsten Probleme in Hinblick auf mehrere Client-Anfragen ist die Simultanität dieser Anfragen. Der Roboter kann hier als eine System-Ressource angesehen werden, auf die mehrere Clients miteinander konkurrierend zugreifen wollen. Das konkrete Problem in dieser Arbeit besteht darin, dass der Saphira-Client eine Steuerungssoftware für den Roboter ist und dementsprechend über keinen Mechanismus für die Verwaltung von mehreren gleichzeitigen Anforderungen an den Roboter verfügt, anders ausgedrückt: Bis ein Job von dem Roboter zu Ende ausgeführt wurde, dürfen keine weiteren Befehle an ihn weitergeleitet werden, andernfalls wird es zum unerwünschten Verhalten des Roboters kommen. Um dieses Problem zu lösen, muss ein Konzept entworfen werden, das die Verwaltung von mehreren gleichzeitigen Anforderungen an den Roboter beinhalten soll. Dieses Konzept wird im Kapitel „Design und Realisierung“ ausführlich behandelt.

Kapitel 3

Design und Realisierung

Im folgenden Kapitel wird der Entwurf des Systems vorgestellt, basierend auf dem in der Analyse beschriebenen Szenario und den gewählten Technologien.

3.1 Grober Überblick

Wie aus dem im Abschnitt „Beispielszenario“ beschriebenen Szenario (siehe Kapitel „Analyse“ Abschnitt 2.2.1 auf Seite 11) und aus dem darauf folgenden Abschnitt „Analyse des zu entwickelnden Systems“ ersichtlich ist, soll das zu entwerfende System hardware-mäßig aus drei großen Teilsystemen bestehen. Zum einen ist es die Client-Anwendung auf dem PDA, zum anderen der Server, der die Anfragen an den Roboter aufnimmt und sie bearbeitet und zum letzten der Roboter selbst mit seiner Laufumgebung. Die Abbildung 3.1 auf der Seite 26 stellt die grobe Skizze dieser Bestandteile und ihres Zusammenspiels dar.

Hier soll die Skizze der Architektur kurz erläutert werden. Ein oder mehrere PDA-Clients können den Roboter-Web-Service aufrufen. Welcher Service das ist, ist hier nicht relevant, aber als ein anschauliches Beispiel wird die Aufforderung benutzt, einen Gegenstand in einen bestimmten, vom Client vorgegebenen Raum zu bringen. Diese Anfragen werden von dem Server entgegen genommen. Unter dem Server wird hier ein Computer verstanden, der mehrere Funktionen in sich vereinigt. In erster Linie soll er im Stande sein, mehrere Client-Anfragen entgegenzunehmen und diese an einen oder mehrere Roboter weiter zu reichen (in der Abbildung 3.1 ist das anschaulich dargestellt). Die Funktionalität des Servers wird ausführlicher in weiteren Abschnitten erläutert.

Die Schnittstelle, die der Server dem Client anbietet, reduziert sich dabei

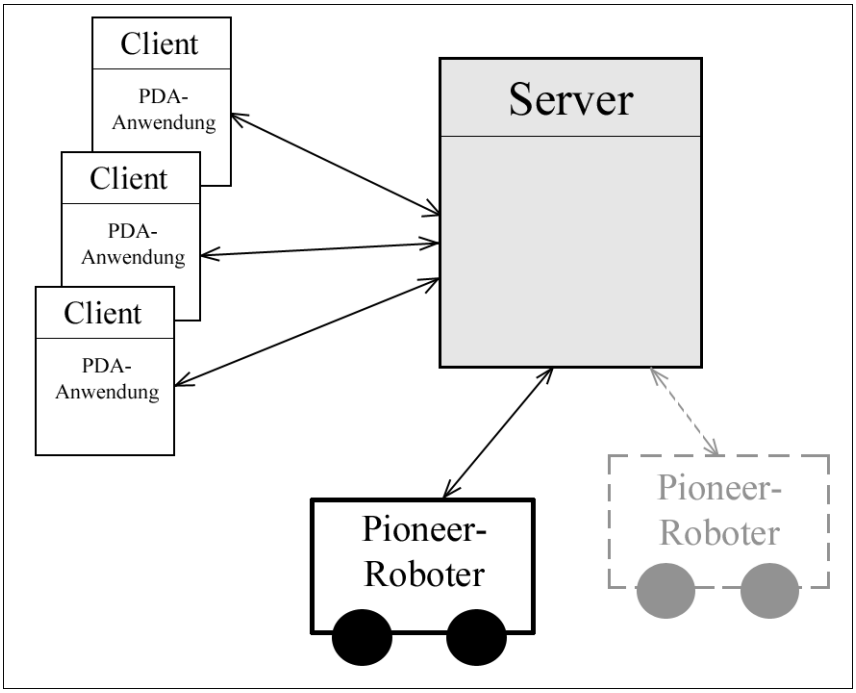

Abbildung 3.1: Architektur: grobe Skizze

auf den Roboter-Web-Service, über den die ganze Kommunikation zwischen dem Client und dem Server stattfinden wird. Die vom Server entgegengenommenen Anfragen werden bearbeitet und daraufhin werden bestimmte Aktivitäten des Roboters angestoßen. Der Server tauscht dabei mit dem Client nach seiner Nachfrage die Zustandsinformationen aus. Unter den Zustandsinformationen werden die Nachrichten verstanden, die eine Aussage über den Fortschritt der Anfragebearbeitung machen, z.B. ob der Roboter die Aufgabe schon erhalten hat und unterwegs ist.

Auch mit dem Roboter tauscht der Server Informationen aus, um ihm die Aufgabe zu erteilen und um nach Bedarf eine Auskunft über den Fortschritt der Ausführung zu erhalten. Die Funktionalität des Roboters ist sehr umfangreich (siehe Kapitel „Analyse“ Abschnitt 2.4.4). Um den Zugriff auf diese Funktionalität so unkompliziert wie möglich zu halten und die Kopplung zwischen dem Server und dem Roboter zu reduzieren, wird hier eine Fassade verwendet und für den Server eine einfache Schnittstelle als Zugriffsmöglichkeit auf das Roboter-Teilsystem angeboten (siehe Abbildung 3.2 Seite 27). In diesem Sinne ist der Server ein Kommunikationspartner für den Client und für den Roboter.

In den weiteren Abschnitten werden die einzelnen Bestandteile der Ar-

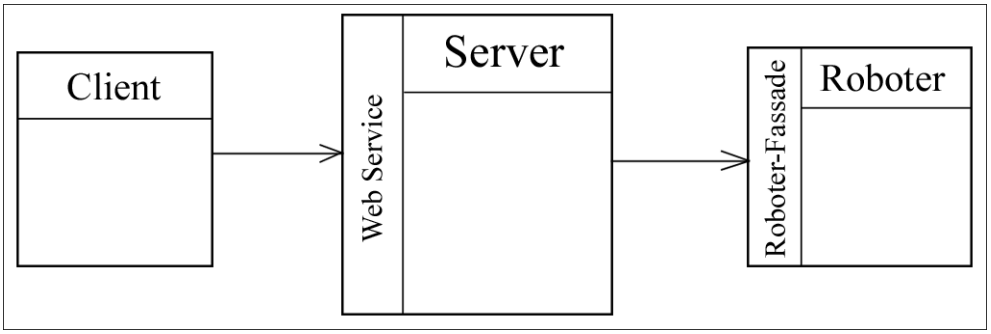

Abbildung 3.2: Architektur: Schnittstellen

chitektur ausführlicher beschrieben.

3.2 Modulare Sicht

3.2.1 Server

Aus dem oben Beschriebenen wird ersichtlich, dass der Server den Kern des Systems darstellt. Zu seinen Funktionalitäten zählt in erster Linie das Anbieten und der Betrieb des Web Services. Um diese Funktionalität zu gewährleisten, muss auf dem Server der Apache Tomcat Web-Container mit der darauf installierten Axis-Webanwendung laufen[1]. In der Abbildung 3.3 auf der Seite 28 ist das anschaulich dargestellt. Der in der Axis-Umgebung installierte Roboter-Web-Service bildet eines der Module des Servers, in Abbildung 3.3 als „Roboter-Web-Service“ zu sehen.

Um das im Kapitel „Analyse“ angesprochene Problem der gemeinsamen Ressourcen-Nutzung (siehe Abschnitt 2.4.5 Seite 24) zu lösen, muss der Server als eine weitere Funktion die Verwaltung von ankommenden Aufträgen übernehmen. Dies wird dadurch gewährleistet, dass für die entgegengenommenen Client-Anfragen eine Queue benutzt wird. Auf diese Weise wird die Funktionalität des Web Services auf die Leitung der neuen Aufträge in die Queue eingeschränkt. Die sich in der Queue befindenden Aufträge werden von einer anderen Anwendung einzeln übernommen, die sie dann weiter bearbeitet. Das Holen eines Auftrages aus der Queue soll die eigentliche Ausführung einer neuen Aufgabe für den Roboter initiieren. Aus der Queue und der weiteren Bearbeitung der Aufträgen setzt sich ein weiteres Modul zusammen, in

[1] Die Gründe für diese Wahl wurden im Kapitel „Analyse“ Abschnitt 2.4.3 ausführlich erläutert.

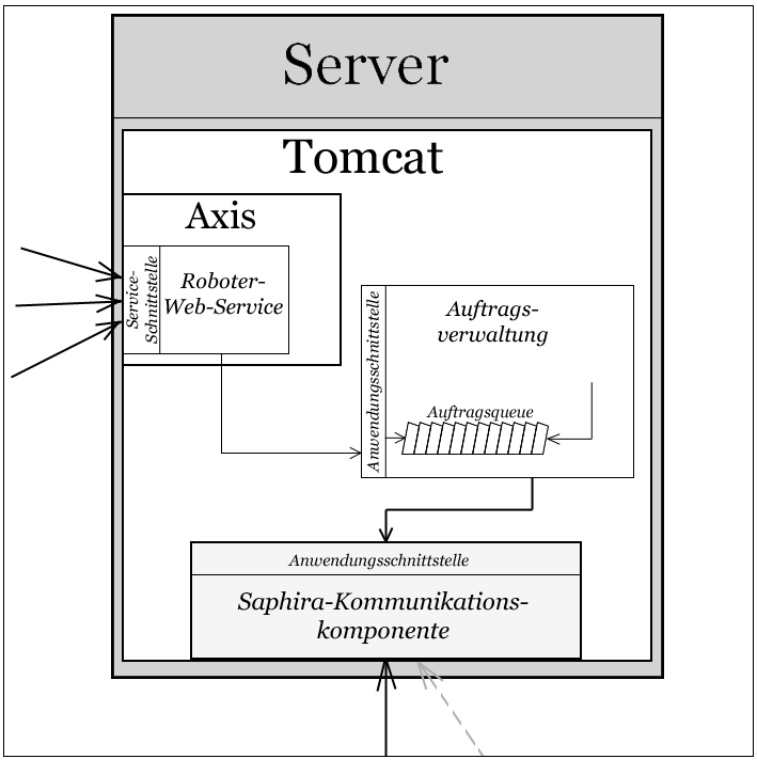

Abbildung 3.3: Server

Abbildung 3.3 als „Auftragsverwaltung“ dargestellt.

Eine der Funktionen des Web-Service-Moduls ist der Austausch der Zustandsinformationen mit dem Client, wie es schon oben beschrieben wurde. Hier taucht ein Problem auf, das mit der Web-Service-Technologie zusammenhängt: Web Services haben in der Regel keinen Zustand und sind nicht Session-behaftet. Deswegen muss, um die beschriebene Funktionalität zu gewährleisten, ein eigener Zustandsmechanismus ausgearbeitet werden. Wenn jeder neue Auftrag beim Eintragen in die Queue eine eindeutige „Auftragsnummer“ verliehen bekommt und diese dann an den Client zurückgegeben wird, wird der Server im Stande sein, den Bearbeitungsfortschritt der Aufgabe für jeden anfragenden Client zu liefern.

Das Web-Service-Modul stellt also für den Client Funktionalitäten zur Verfügung, die über eine Schnittstelle folgendermaßen beschrieben werden können:

- Gegenstand bringen(*Raum*), Rückgabe: Auftragsnummer
- Zustand geben(*Auftragsnummer*), Rückgabe: Zustand

Das Auftragsverwaltung-Modul stellt für das Web-Service-Modul folgen-

de Schnittstelle bereit:

- Neuen Auftrag in die Queue eintragen, Rückgabe: Auftragsnummer
- Zustand geben(*Auftragsnummer*), Rückgabe: Zustand

Der Saphira-Client kann auf mehrere Weisen in der Verbindung mit dem Roboter stehen (siehe Abschnitt 2.4.4 auf Seite 22). Aus der Sicht der „Auftragsverwaltung“-Anwendung gibt es zwei Varianten, auf Saphira zuzugreifen: lokal, wenn Saphira auf demselben Host-Computer läuft, und remote, wenn Saphira auf einem auf dem Roboter befestigten Laptop installiert ist. Um die Flexibilität zu ermöglichen, wird sich die „Auftragsverwaltung“-Applikation über ein Interface mit Saphira in Verbindung setzen. Dieses Interface soll grob granulare Funktionalitäten beschreiben, wie z.B. „bringe Gegenstand in Raum(*Raum*)“, ohne dabei auf die Feinheiten einzugehen. Diese Schnittstelle kann dann von einer weiteren Anwendung implementiert werden, die mit der lokalen oder mit der entfernten Saphira arbeiten wird.

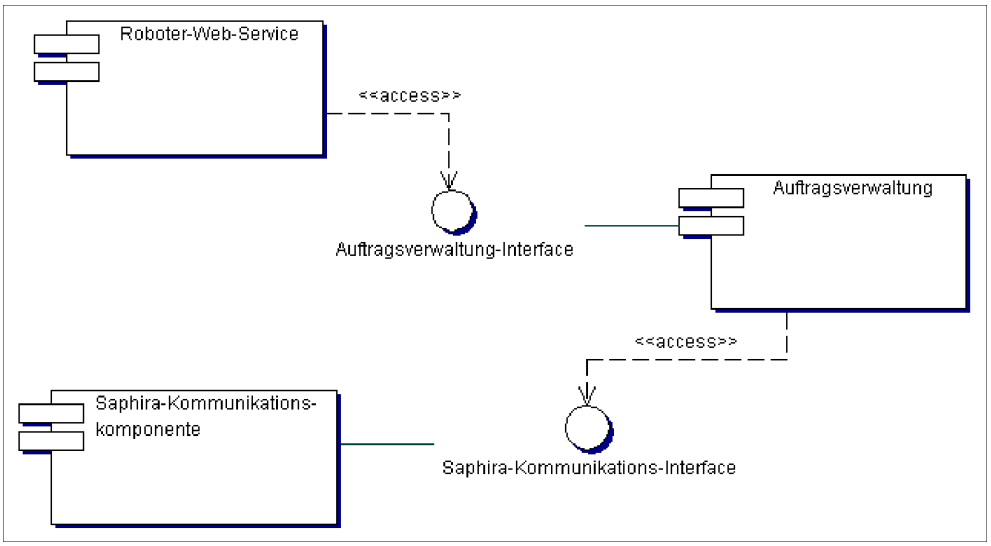

Abbildung 3.4: Server: Komponentendiagramm

Somit wird die Kopplung der verschiedenen Module verringert und die Flexibilität erhöht. Der entfernte Zugriff auf den Saphira-Client wurde schon von Florian Boldt und Hauke Löhler [BL99] realisiert, deswegen wird in dieser Arbeit auf die Realisierung eines remote-Zugriffs nicht weiter eingegangen und nur ein lokaler Saphira-Client als Möglichkeit der Kommunikation mit dem Roboter betrachtet. Diese Anwendung bildet dann das dritte System-Modul, auf dem Bild 3.3 als „Saphira-Kommunikationskomponente“ dargestellt. Da außer der Auftragserteilung noch die Abfrage der Zustandsinforma-

tionen der Bearbeitung möglich sein soll, soll die Schnittstelle der „Saphira-Kommunikationskomponente“ auch diese Funktion beschreiben. Diese Schnittstelle kann also durch folgende wesentlichen Funktionen charakterisiert werden:

- bringe Gegenstand in den Raum(*Raum*)
- gib den Zustand des Auftrages, Rückgabe: Zustand

In der Abbildung 3.4 Seite 29 sind alle drei Module des Servers und ihre Verbindungen zueinander anschaulich dargestellt.

3.2.2 PDA-Client

Der PDA-Client besteht aus einem einzigen Modul – der Anwendung, die den Roboter-Service aufruft. Die Funktionalitäten dieser Anwendung sollen sich deswegen im Wesentlichen auf die Schnittstelle beziehen, die der Web Service zur Verfügung stellt, also das Erteilen eines neuen Auftrags für den Roboter und die Abfrage des Fortschritts der Aufgabendurchführung.

Darüber hinaus soll der Client im Stande sein, den Provider, der den Web Service anbietet, zu lokalisieren. Dazu gibt es mehrere Möglichkeiten. Als Erstes könnte UDDI (siehe Kapitel „Analyse“ Abschnitt 2.4.3 auf Seite 19) dazu benutzt werden, den Web Service ausfindig zu machen. Aber im Falle dieser Arbeit wird davon ausgegangen, dass der Service in einer lokalen, abgeschlossenen Umgebung zur Verfügung gestellt wird, wo der Web Service selbst und seine Beschreibung bekannt sind, während UDDI meistens dazu dient, Web Services nach einem bestimmten Kriterium ausfindig zu machen, wobei deren konkrete Beschreibung zur Zeit der Suche noch nicht bekannt ist. Aus diesem Grund wird in dieser Arbeit auf die Nutzung dieses Mechanismus verzichtet, um den damit verbundenen Umfang möglichst klein zu halten.

Um trotzdem eine gewisse Flexibilität zu erhalten, wird als eine weitere Funktion des Clients die Möglichkeit der Angabe der Provider-Adresse definiert. Damit wird gewährleistet, dass der Service von einem beliebigen Host angeboten werden kann.

3.2.3 Roboter

Das Roboter-Modul stellt die Fassade zu der Funktionalität der Saphira-Umgebung dar, erweitert um die Aktivitäten, die das Stellen von komplexeren Aufgaben für den Roboter, wie z.B. zu einem vorgegebenen Raum zu fahren, ermöglichen.

Wie im Kapitel „Einleitung“ schon angesprochen wurde, wird die Entwicklung des Serviceroboters nicht als eine der Aufgaben dieser Arbeit angesehen (siehe Abschnitt 1.2). Deswegen wird als Roboter-Modul eine Ser-

viceroboter-Anwendung genommen, die im Rahmen des Wahlpflichtmoduls „Verhaltensbasierte Systeme / Agentensysteme“ im Sommersemester 2005 an der HAW Hamburg von den Studenten Alexander Seizeller und Kien Nghia Tran [ST] entwickelt wurde. Die Anwendung enthält folgende Funktionalität:

- fahren zu einem vorgegebenen Raum, ggf. um einen Gegenstand abzuholen;
- zurückkommen zu einem anderen vorgegebenen Raum, ggf. mit dem abgeholten Gegenstand.

Diese Anwendung wird im Weiteren als eine Art „Blackbox“ behandelt und auf die Details ihres Aufbaus wird nicht weiter eingegangen. Der Programm-Quellcode befindet sich im Anhang dieser Arbeit.

Das Roboter-Modul soll dem Server also folgende Funktionalität zur Verfügung stellen:

- Initialisierung der Saphira-Umgebung;
- Funktionalität des externen Programms von Alexander Seizeller und Kien Nghia Tran.

Dieses Modul wird als ein Beispiel benutzt, um die Kommunikation zwischen dem Server und dem Roboter zu demonstrieren.

3.3 Design der Module

In den weiteren Abschnitten werden Design-Entscheidungen der einzelnen Module ausführlich erläutert, einschließlich Klassen- und Ablauf-Modelle.

3.3.1 Auftragsverwaltung

3.3.1.1 Allgemeine Probleme der Verwaltung

Die Verwaltung von Aufträgen gehört in den Bereich der Ablaufplanung und der Ablaufkoordination. In Abhängigkeit davon, wie das System aufgebaut ist und welche Ansprüche an die Verwaltung gestellt werden, können verschiedene Ansätze für die Lösung der damit verbundenen Probleme verwendet werden. Wenn der oder die im System einzusetzende(n) Serviceroboter so konstruiert sind, dass sie zu einem Zeitpunkt nur einen Gegenstand transportieren können, dann können sie als dem System zur Verfügung stehende Ressourcen angesehen werden. Die Verwaltung der Aufträge oder „Jobs“ ist in diesem Fall auf die Verwaltung von Systemressourcen zurückzuführen und kann über den Scheduling-Ansatz, wie es in Betriebssystemen

für die Verwaltung von konkurrierenden Prozessen eingesetzt wird, gelöst werden (für weitere Informationen über Scheduling siehe [TAN1]).

Wenn der Roboter mehrere Gegenstände gleichzeitig transportieren kann, hat die Verwaltung komplexere Aufgaben. In diesem Fall kann die Ausführung der einzelnen Aufträge optimiert werden. Der Roboter könnte z.B. mehrere Aufträge auf einer Fahrt erledigen. Dazu sollte aber eine möglichst optimale Route für die Fahrt bestimmt werden. Das Optimierungsproblem tritt auch dann ein, wenn der Roboter verschiedene Arten von Aufgaben erledigen soll, wie z.B. nicht nur einen „bringe"-, sondern auch einen „hole"-Service. In diesem Fall sollte die Roboter-Route auch optimal berechnet werden, um überflüssige Fahrten des Roboters zu vermeiden. Die aufgeführten Probleme fallen in Bereich des berühmten „Traveling Salesman"-Problems (siehe auch [GO03]), das zu den Problemen gehört, mit denen sich Operations Research beschäftigt. „Unter Operations Research (OR) wird allgemein die Entwicklung und der Einsatz quantitativer Modelle und Methoden zur Entscheidungsunterstützung verstanden."([GOR]) Operations Research ist durch die Zusammenarbeit von Mathematik, Wirtschaftswissenschaften und Informatik geprägt. Zu ihren wichtigsten Verfahren zählen „die lineare Programmierung u. a. Formen der Programmierung, die Spieltheorie, die Wahrscheinlichkeitsrechnung, die Simulation und die Theorie der Warteschlangen" (vgl. [BELE]).

In dieser Arbeit wird davon ausgegangen, dass der einzusetzende Serviceroboter über keine Möglichkeit zum Transport von mehreren Gegenständen verfügt. Darüber hinaus wird nur die „bringe einen Gegenstand"-Aufgabe als ein Dienst des Roboters angesehen, da sonst der damit verbundene Aufwand den Rahmen dieser Arbeit sprengen würde. Deswegen wird die Optimierung nicht als eine der Aufgaben des „Auftragsverwaltung"-Moduls angesehen.

3.3.1.2 Design des Moduls

Die Anforderungen an das Modul „Auftragsverwaltung" können folgendermaßen definiert werden:

- Das Modul soll die Annahme der Aufträge und ihre Weiterleitung an die Saphira-Kommunikationskomponente übernehmen;
- Auf dem Server soll während seines Lebenszyklus nur eine einzige Instanz des Moduls existieren, um die Konsistenz der Verwaltung zu garantieren;
- Das Modul soll eine einfache einheitliche Schnittstelle anbieten;
- Die Flexibilität der Implementierung soll gewährleistet werden.

Die Persistenz der Aufträge wird in dieser Arbeit außer Acht gelassen. Allerdings ist es eine sehr wichtige Anforderung angesichts der Wahrscheinlichkeit eines Server-Ausfalls. Um die Aufträge persistent zu machen, könnte z.B. solche Technologie wie Java Message Service (kurz JMS) [JMS] eingesetzt werden, die das Senden, das Empfangen und das Lesen von asynchronen Nachrichten ermöglicht.

Um die Konsistenz der Verwaltung zu erreichen, wird hier das Singleton-Pattern [GA04] angewendet. Als Singleton-Klasse wird die Klasse „AssignmentAdministration“ entworfen (siehe Klassendiagramm 3.5). Gleichzeitig repräsentiert diese Klasse die Schnittstelle, die das Modul für den Web Service zur Verfügung stellt, und ist damit die Fassade des Moduls (Fassade-Pattern [GA04]), sie leitet die funktionalen Aufrufe an die Klassen weiter, die diese Funktionalität implementieren. Die „AssignmentAdministration“

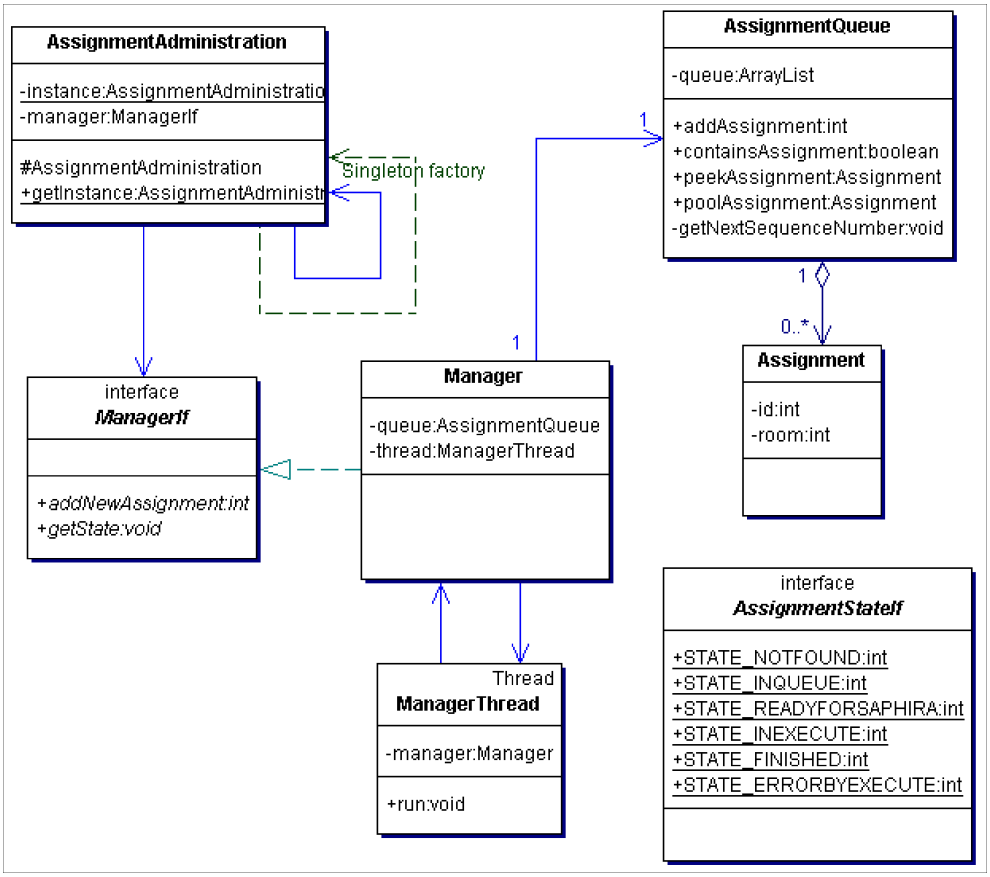

Abbildung 3.5: Klassendiagramm: Auftragsverwaltung

hält eine Referenz auf das Interface „ManagerIf“ und nicht auf eine konkre-

te Klasse. Das Interface kann in Bezug auf verschiedene Realisierungen der Funktionalität mehrmals implementiert werden. Welche Implementierung genommen wird, kann in einer Konfigurationsdatei, die im XML-Format erstellt wird, festgelegt und beim Starten des Moduls auf dem Server geladen werden. Damit wird eine hohe Flexibilität der Realisierung dieses Moduls erreicht.

Für die eingehenden Aufträge wird eine Queue benutzt (siehe „AssignmentQueue“ in dem Diagramm 3.5), die nach dem einfachen FIFO-Prinzip arbeitet. Die Verwaltung der Aufträge kann allerdings komplexer sein, wie es schon im Abschnitt 3.3.1.1 erwähnt wurde. In diesem Fall könnte eine Prioritätsqueue eingesetzt werden, in der einzelne „Jobs“ eine Priorität nach ihrer Abhängigkeit voneinander bekommen (siehe Abschnitt 3.3.1.1).

Die Aufträge werden als „Assignment“-Objekte in die Queue eingetragen, die Eindeutigkeit der Aufträge wird durch ihre ID festgelegt. Der Status eines Auftrages kann dann über seine ID ermittelt werden. Die möglichen Status werden in einem Interface als statische Konstanten festgelegt (im Diagramm 3.5 als „AssignmentStateIf“ zu sehen).

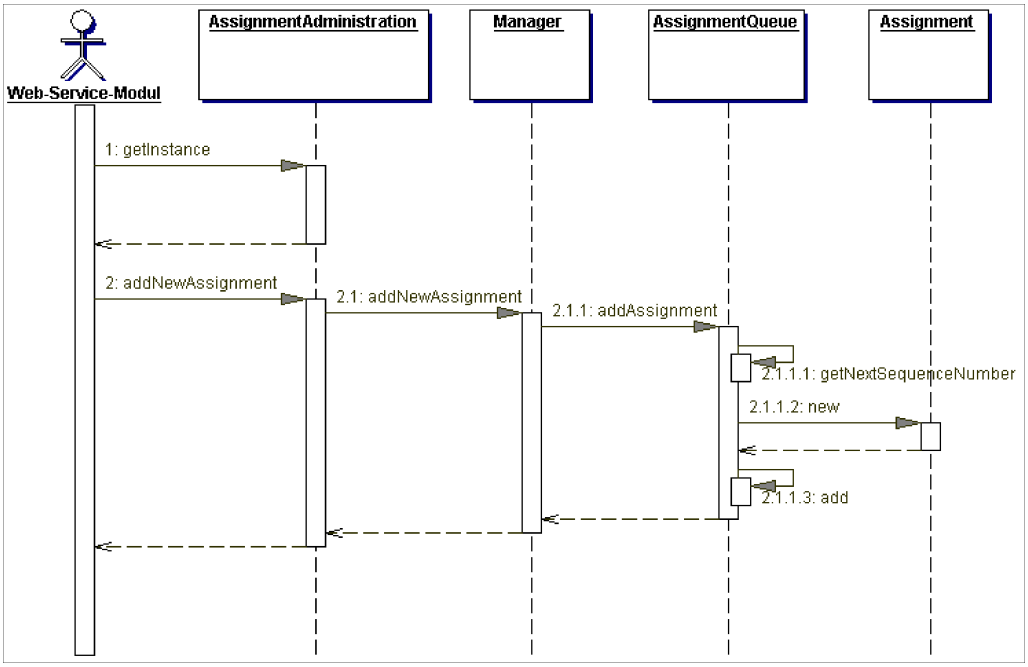

Abbildung 3.6: Sequenzdiagramm: Auftragsverwaltung - Neuer Auftrag

Das dynamische Zusammenspiel der Modul-Objekte ist in den Sequenzdiagrammen 3.6 und 3.7 anschaulich dargestellt. In diesen Diagrammen ist zu sehen, welche Interaktionen zwischen verschiedenen Objekten des „Auf-

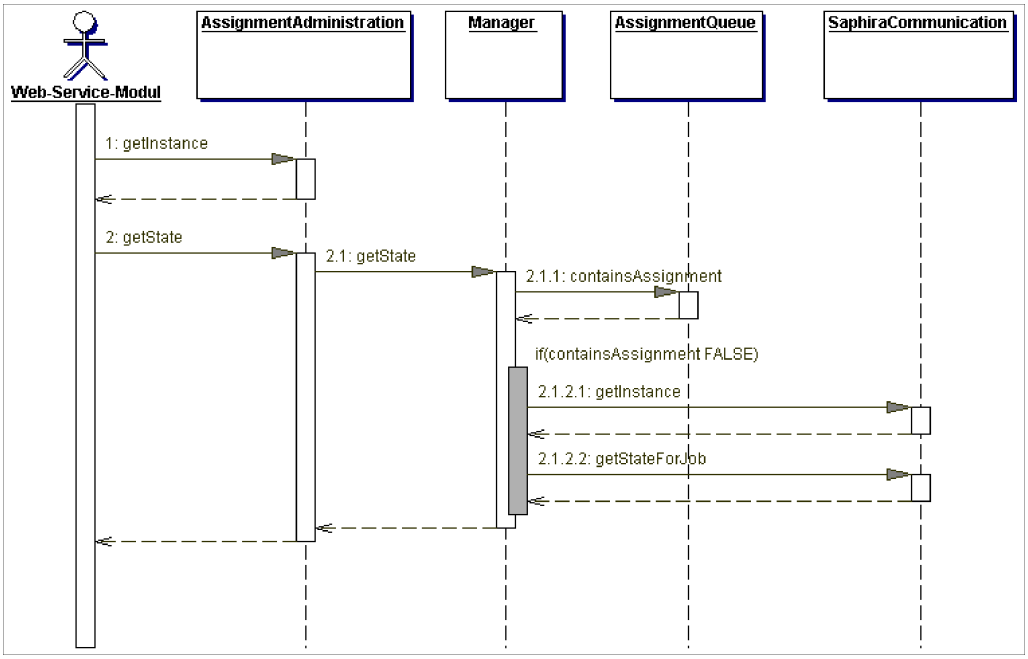

Abbildung 3.7: Sequenzdiagramm: Auftragsverwaltung - Zustand ermitteln

tragsverwaltung“-Moduls und Modul-übergreifende Interaktionen stattfinden, wenn ein neuer Auftrag ins System eingeht (Diagramm 3.6) und wenn eine Anfrage nach dem Zustand eines schon vorhandenen Auftrages erfolgt (Diagramm 3.7). Das Aktivieren von Operationen des „SaphiraCommunication“-Objektes gehört dabei zu den Modul-übergreifenden Interaktionen, da dieses Objekt ein Bestandteil des Moduls „Saphira-Kommunikationskomponente“[2] ist. Ein Zugriff auf dieses Modul kommt nur dann zu Stande, wenn der Zustand eines Auftrages ermittelt werden muss und der gesuchte Auftrag dabei nicht in der „AssignmentQueue“ gefunden wurde, dieser Fall ist im Diagramm 3.7 mit „*if(containsAssignment FALSE)*“ gekennzeichnet. In dieser Situation muss nachgeschaut werden, ob der Auftrag schon in der Ausführung ist. Wenn sich auch das als erfolglos erweist (der Auftrag mit der vorgegebenen ID wurde nicht gefunden), wird an den anfragenden Akteur, in diesem Fall ist es das „Roboter-Web-Service“-Modul, der Status „NOT FOUND“ zurückgesendet.

Die Verwaltung der Bearbeitung der Aufträge wird von der Klasse „ManagerThread“ übernommen. Diese Klasse soll parallel zum restlichen Prozess als ein Thread gestartet werden, um eine kontinuierliche Verarbeitung von

[2]Eine ausführliche Beschreibung dieses Moduls in weiteren Abschnitten.

Queue-Einträgen zu garantieren. Die Funktionalität dieser Klasse besteht aus der Weiterleitung der Aufträge an die „Saphira-Kommunikationskomponente“. Im Sequenzdiagramm 3.8 ist der Ablauf der Arbeit des ManagerThreads zu

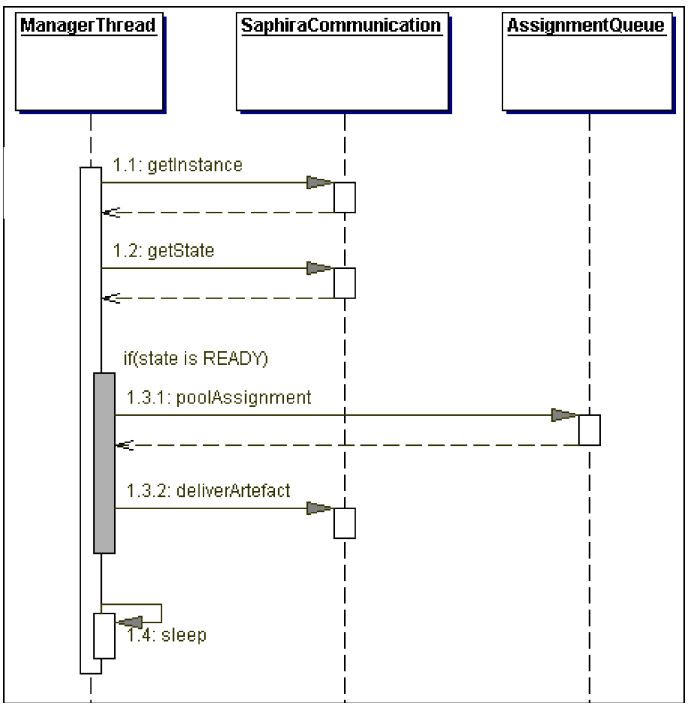

Abbildung 3.8: Sequenzdiagramm: ManagerThread

sehen. Aus dem Diagramm ist ersichtlich, dass das Holen eines Auftrages aus der „AssignmentQueue“ und das Weiterreichen dieses Auftrages an das Modul „Saphira-Kommunikationskomponente“ nur dann geschieht, wenn dieses Modul bereit für den nächsten Auftrag ist, was über den Status des „SaphiraCommunication“-Objektes mit der Operation *getState* bestimmbar ist. Dabei ist diese Operation nicht mit der *getStateForJob* (siehe Diagramm 3.7) zu verwechseln: Die erste gibt die Auskunft über den Zustand des Moduls und die zweite über den Zustand des einzelnen Auftrages. Wenn die „SaphiraCommunication“ bereit ist (im Diagramm durch „*if(state is READY)*“ erkennbar), wird an sie ein neuer Auftrag übergeben.

Der hier dargestellte Zyklus wiederholt sich in regelmäßigen Abständen (siehe die Operation „sleep“ im Diagramm), der zeitliche Abstand zwischen zwei Zyklen ist dabei variierbar und kann auch in der oben erwähnten Konfigurationsdatei festgelegt werden.

3.3.2 Saphira-Kommunikationskomponente

Das Modul „Saphira-Kommunikationskomponente“ ist für die Kommunikation mit dem Saphira-Client verantwortlich, der den Zugriff auf den Serviceroboter ermöglicht (siehe 2.4.4). Im Abschnitt 3.2.1 (Seite 29) wurde seine Schnittstelle, die dieses Modul für die „Auftragsverwaltung“ bereit stellt, allgemein beschrieben. In diesem Abschnitt wird der interne Aufbau des Modules und seine Funktionalität ausführlich behandelt.

Die Kommunikation mit dem Saphira-Client kann auf verschiedene Weisen erfolgen. Einer der Gesichtspunkte wurde schon in den vorigen Abschnitten erläutert, nämlich in Bezug darauf, wo Saphira gestartet wird. Ein anderer Aspekt, der beim Entwurf berücksichtigt werden sollte, ist die Tatsache, dass nur eine Instanz des Saphira-Clients zu einem Zeitpunkt im Betriebssystem existieren kann, was auf die Eigenschaften von Saphira [SA99] zurückzuführen ist. Beim Entwurf des Systems wurde die Entscheidung getroffen, dass das Starten des Saphira-Clients aus dem Modul heraus durchgeführt wird. Das erleichtert den Zugriff auf Saphira und die Kontrolle über sie. Basierend auf diesen Fakten können die Grundanforderungen an die „Saphira-Kommunikationskomponente“ wie folgt festgelegt werden:

- Das Modul soll das Weiterreichen der Aufträge an den Serviceroboter-Modul übernehmen;
- Das Modul soll eine einheitliche Schnittstelle anbieten;
- Die Flexibilität der Umsetzung soll gewährleistet werden;
- Das Anstoßen der Initialisierung von Saphira soll nur einmal geschehen, um während des Lebenszyklus des Moduls die Referenz auf dieselbe Saphira-Instanz zu erhalten.

Entsprechend diesen Anforderungen wurde eine Klasse als eine Schnittstelle des Moduls „Saphira-Kommunikationskomponente“ unter der Anwendung des Fassade- und des Singleton-Patterns [GA04] entworfen, die im Klassendiagramm 3.9 auf Seite 38 als „SaphiraCommunication“ zu sehen ist. Der Aufbau dieser Klasse ist ähnlich dem der Klasse „AssignmentAdministration“ aus dem Modul „Auftragsverwaltung“: Sie hält eine Referenz auf das Interface „SaphiraClientIf“ und ermöglicht somit die Flexibilität des Designs des Moduls. An dieser Stelle kommt wieder die schon im Abschnitt „Auftragsverwaltung“ erwähnte Konfigurationsdatei zum Einsatz: Die letztendlich zu initialisierende Klasse wird in dieser Datei festgelegt, und bei der Initialisierung der „SaphiraCommunication“ wird eine entsprechende Instanz erzeugt.

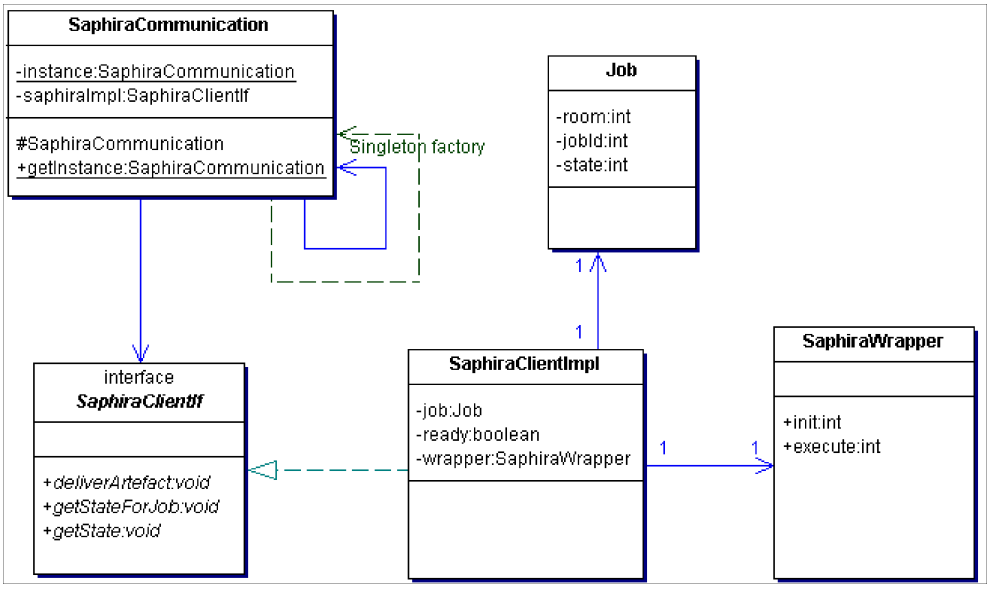

Abbildung 3.9: Klassendiagramm: Saphira-Kommunikationskomponente

In dieser Arbeit implementiert die Klasse „SaphiraClientImpl“ das „SaphiraClientIf“-Interface und setzt damit die Funktionalität des Moduls um.

Der Saphira-Client kann zu einem Zeitpunkt nur einen Auftrag bearbeiten. Diesen Auftrag hält die „SaphiraClientImpl“ als eine Referenz auf ein Objekt von der Klasse „Job“, das die Eigenschaften des aktuellen, an dieses Modul übergebenen Auftrages enthält, wie die ID des Auftrages und der Status der Ausführung. Die Klasse „SaphiraWrapper“ stellt die Verbindung zum nächsten System-Modul dar: dem „Pioneer-Roboter“ (siehe Abbildung 3.1 auf Seite 26). Die Funktionen dieser Klasse bestehen aus dem Aktivieren der Initialisierung des Saphira-Clients und dem Überreichen eines Auftrages an diesen.

Die Funktionalität der „Saphira-Kommunikationskomponente“ kann sich auf simples Weiterreichen der Befehle an Saphira beschränken oder auch Verwaltungsfunktionen beinhalten, wenn z.B. dem System mehrere Serviceroboter zur Verfügung stehen. In solchem Fall können die Roboter als ein Pool von Ressourcen angesehen und ihre Koordinierung von diesem Modul übernommen werden. Wie aus dem Diagramm 3.9 ersichtlich wird, wurde die Funktionalität der „Saphira-Kommunikationskomponente“ in dieser Arbeit einfach gehalten, aber aufgrund des flexiblen Designs sollten keine Schwierigkeiten bestehen, sie im Bedarfsfall zu erweitern oder die bestehende Komponente durch eine andere zu ersetzen.

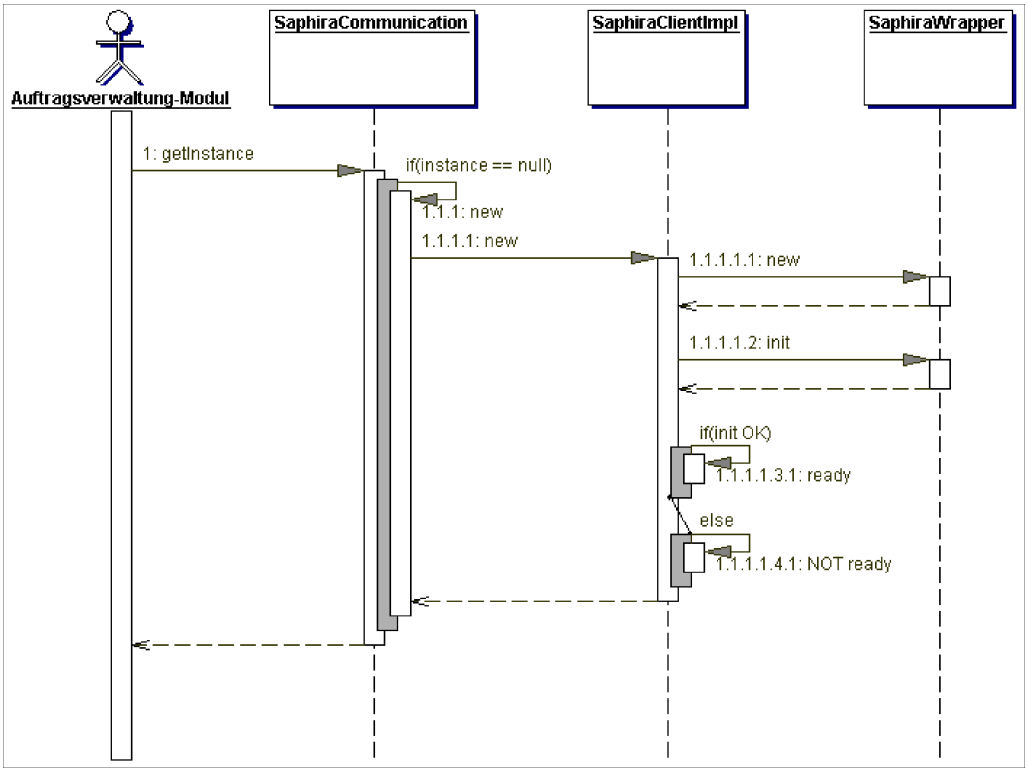

Abbildung 3.10: Sequenzdiagramm: Saphira-Kommunikationskomponente - Initialisierung

Das dynamische Verhalten der Objekte dieses Moduls wird mit Hilfe von drei folgenden Sequenzdiagrammen beschrieben. Im Diagramm 3.10 ist der Initialisierungsablauf des Moduls anschaulich dargestellt. Im Diagramm ist zu sehen, dass bei der Initialisierung des „SaphiraCommunication“-Objektes gleichzeitig auch andere tragende Objekte des Moduls initialisiert werden. Sogleich wird der Saphira-Client gestartet (über die Operation *init* im Diagramm). Im Fall einer erfolgreichen Saphira-Initialisierung bekommt der „SaphiraClientImpl“-Objekt den Status *READY*, was signalisieren soll, dass der Saphira-Client für die Ausführung von Aufgaben bereit ist. Dieser Ablauf ist im Diagramm mit *if(init OK)* gekennzeichnet. Die vom „ManagerThread“ aus dem Modul „Auftragsverwaltung“ angestoßene Operation *deliverArtefact* aktiviert die Ausführung eines von ihm neu übergebenen Auftrages (Sequenzdiagramm 3.11 Seite 40). Das geschieht allerdings nur in dem Fall, wenn der Status der „Saphira-Kommunikationskomponente“ *READY* ist (siehe Abbildung 3.8 Seite 36). Für die neue Aufgabe wird von der „SaphiraClientImpl“ ein „Job“-Objekt erzeugt, dessen Referenz sie bis zum Ende der Ausführung

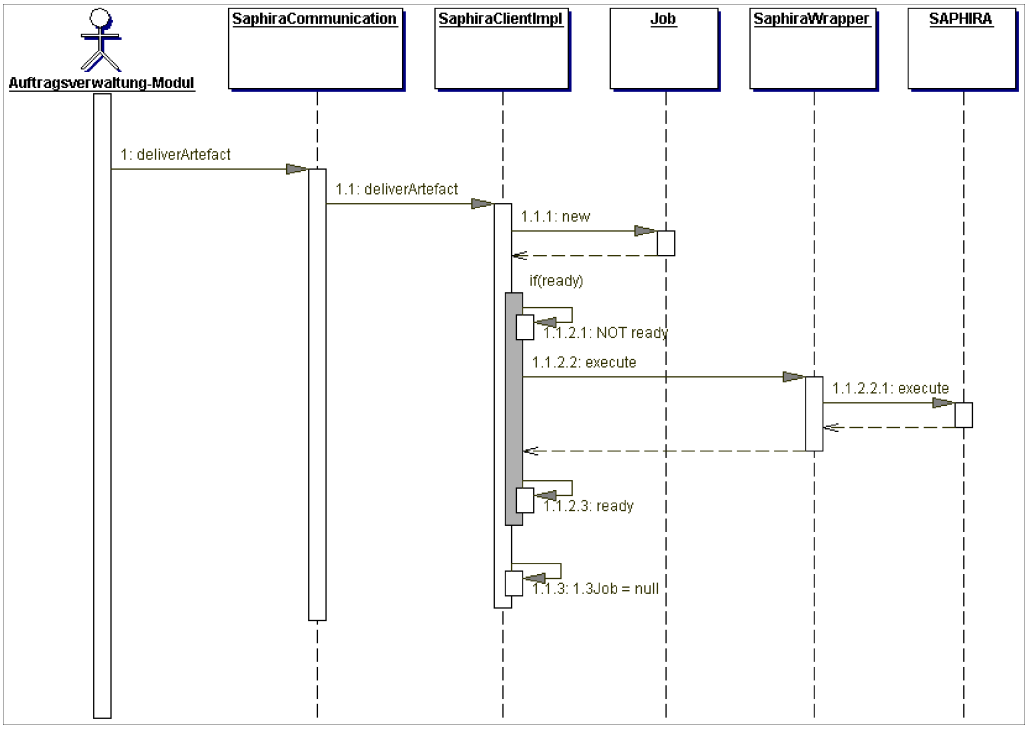

Abbildung 3.11: Sequenzdiagramm: Saphira-Kommunikationskomponente - Neuer Auftrag

der Aufgabe behält. Vor dem Ausführen der Aufgabe wird der Zustand des Moduls als *NOT READY* gekennzeichnet. Nach dem Ausführen wird er wieder auf *READY* geändert. Im Sequenzdiagramm 3.12 auf Seite 41 ist der Ablauf des Abfragens des Status der Bearbeitung der Aufgabe geschildert. Der Ansprechpartner in diesem Fall ist das Objekt „Job“, das unter anderem die Information über den Zustand des Auftrages beinhaltet. Die Aktualität dieser Information wird dadurch gewährleistet, dass dieses Objekt beim Aktivieren der Operation *execute* des Roboter-Moduls (siehe Abbildung 3.11) an ihn als Parameter übergeben wird. Der Kontrollmechanismus, der in dem Diagramm 3.12 mit *if(AuftragID == jobID)* gekennzeichnet ist, garantiert die Prüfung der Identität des Auftrages. Wenn die übergebene ID nicht mit der von „Job“ übereinstimmt, wird auf die Anfrage des „Auftragsverwaltung“-Moduls mit *NOT FOUND* geantwortet.

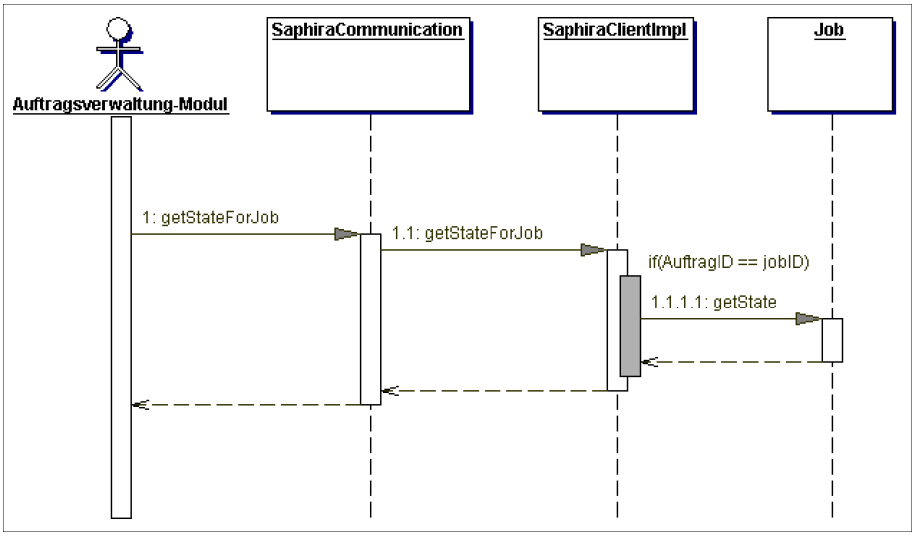

Abbildung 3.12: Sequenzdiagramm: Saphira-Kommunikationskomponente - Status eines Auftrages

3.3.3 Roboter-Web-Service

Das Modul „Roboter-Web-Service“ ist relativ schmal gehalten, was der Absicht entspricht, keine Geschäftsabläufe in diesem Modul zu platzieren und es als Fassade für die Geschäftslogik, die in zwei anderen Server-Modulen enthalten ist (siehe Abschnitte 3.3.1 und 3.3.2), und als Kommunikationsschnittstelle für Clients zu verwenden.

Das „Roboter-Web-Service“-Modul wäre die richtige Komponente für den Entwurf eines Sicherheitsmechanismus, wenn solcher für eine sichere Kommunikation zwischen dem Client und dem Server gefordert wird. Bei dieser Kommunikation kommen folgende Sicherheitsaspekte in Frage: die Authentifikation, die Autorisation und der sichere Datenaustausch. Die Authentifikation würde garantieren, dass der Client wirklich der ist, für den er sich ausgibt, und die Autorisation des Clients, dass er befugt ist, das System zu nutzen. Diese zwei Mechanismen könnten über so genannte Tickets realisiert werden, wie z.B. im Kerberos-Verfahren. Der sichere Datenaustausch sollte garantieren, dass die Vertraulichkeit der Nachrichten, die zwischen dem Client und dem Server versendet werden, erhalten bleibt. Dazu könnte z.B. ein Verschlüsselungsverfahren umgesetzt werden (für weitere Informationen zum Thema Sicherheit siehe [EC04]).

In dieser Arbeit wird die Sicherheit außer Acht gelassen, da das Eingehen auf dieses Thema den Rahmen der Arbeit sprengen würde. Dementsprechend

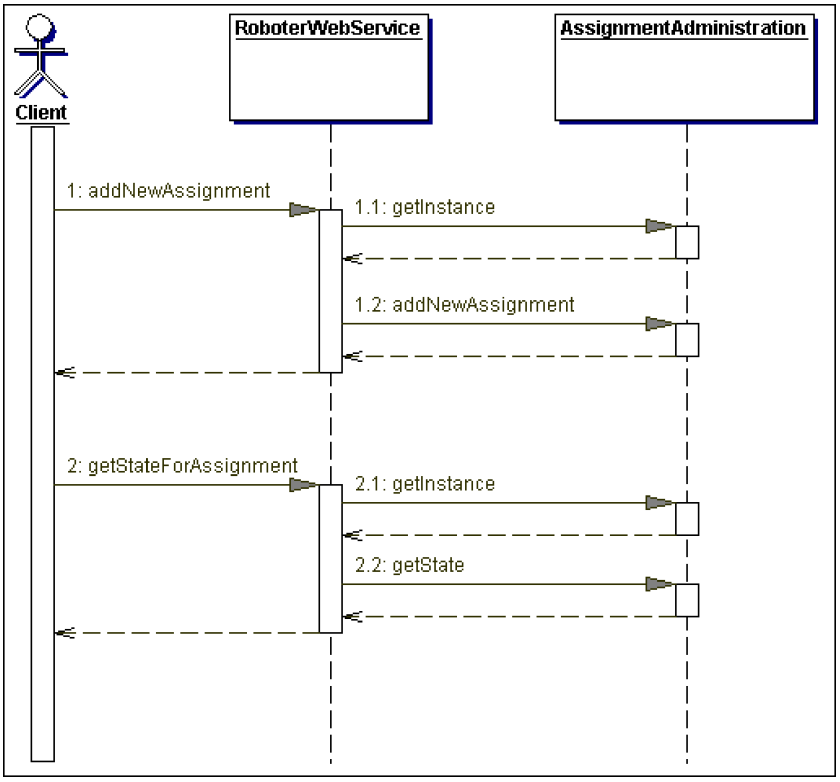

Abbildung 3.13: Sequenzdiagramm: Roboter-Web-Service

besteht das Modul nur aus einer Klasse: „RoboterWebService“, die die Basis für den zu veröffentlichenden Web Service bildet und auf den Web Container installiert werden soll. Wie in dem Abschnitt 3.2.1 schon beschrieben wurde, zählen zu seinen Funktionalitäten die Aufnahme eines neuen Auftrages für den Roboter und die Ausgabe der Information über den Fortschritt der Bearbeitung dieses Auftrages. Diese Funktionen werden als zwei Operationen des Web Services zur Verfügung gestellt.

Das Sequenzdiagramm 3.13 zeigt den dynamischen Ablauf von Aufrufen der entsprechenden Operationen des Web Services. Wie aus dem Diagramm zu sehen ist, leitet der „Roboter-Web-Service“ die Client-Aufrufe an die „AssignmentAdministration“-Fassade weiter, was die Kopplung zwischen den beiden Modulen minimiert und die Flexibilität erhöht. Beim Aufruf des Services bekommt der Client als Rückgabewert die ID des neuen vom System aufgenommenen Auftrages. Dieser Ablauf ist im Diagramm mit *AddNewAssignment* gekennzeichnet. Danach hat der Client jederzeit die Möglichkeit zu erfahren, wie weit der Fortschritt der Bearbeitung des Auftrages ist. Dazu

braucht er die ihm vorher übertragene Auftrags-ID (auf dem Diagramm als *getState*-Operation zu sehen).

3.3.4 PDA-Anwendung

Wie schon im Abschnitt 3.2 beschrieben wurde, soll sich die Funktionalität der PDA-Anwendung auf die Funktionalität des Roboter-Web-Services richten. Darüber hinaus soll in dieser Anwendung eine Möglichkeit für die Anpassung der mit dem Web Service verbundenen Einstellungen gegeben werden. Dieses Modul wird dementsprechend eine überschaubare Funktionalität besitzen. Da diese Anwendung für die Kommunikation mit dem Endbenutzer entwickelt wird, ist einer der wichtigsten Punkte bei dem Entwurf dieses Moduls eine ergonomische Gestaltung der Benutzeroberfläche. Die Grundanforderungen an die PDA-Anwendung können also folgendermaßen definiert werden:

- Das Modul soll die Funktionalität des Roboter-Web-Services anbieten.
- Die Flexibilität in Bezug auf das Anbinden des Providers des Web Services soll gewährleistet werden.
- Eine ergonomische Gestaltung der Oberfläche soll eingehalten werden.

Der Design der grafischen Oberfläche wird sich an die bekannten Richtlinien der Software-Ergonomie halten (siehe dazu [BA99] und [WE98]).

Die Funktionalität der Anwendung kann in zwei Teile getrennt werden: die Initialisierung des Services bildet einen Teil und die Verfolgung der Ausführungsfortschritts einen anderen. In dieser Sicht stellt sich die Frage, wie sich der PDA-Client während der Ausführung eines Auftrages verhält. Dabei stehen zwei mögliche Handlungsweisen zur Auswahl. Nach dem Aufruf des Services bleibt die Anwendung entweder weiterhin geöffnet, bis der Roboter den Service ausgeführt hat, oder die Anwendung wird nach dem Aufruf geschlossen und zum Abfragen des Auftragszustandes neu gestartet. In solcher Situation sollte das Modul dafür sorgen, dass die an ihn übergebene Auftrags-ID bis zum Ende der Bearbeitung erhalten bleibt, z.B. durch das Speichern in einer temporären Datei. In dieser Arbeit wird davon ausgegangen, dass die PDA-Anwendung geöffnet bleibt, bis der Auftrag abgeschlossen wurde. Das wäre auch der Fall, wenn der Roboter-Web-Service in einem umfangreichen „Ferien-Club“-System eingesetzt würde. Dann wäre dieses Modul nur ein Teil einer größeren Client-Anwendung, die permanent auf dem PDA läuft. Allerdings käme hier die Frage der Erhaltung der Daten für den Fall, dass der PDA selbst ausgeschaltet wird. Für einen solchen Fall wäre die Speicherung des Auftrags-ID ebenfalls sinnvoll.

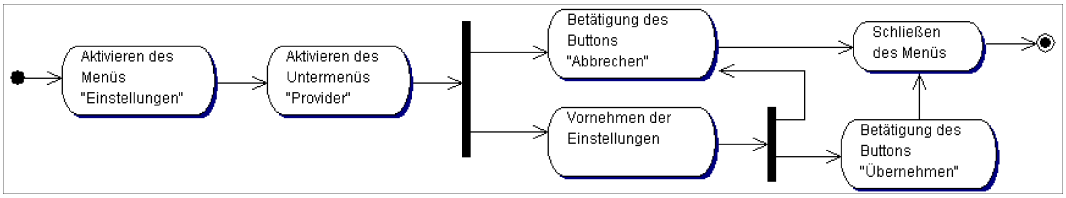

Abbildung 3.14: Aktivitätsdiagramm: PDA-Anwendung - Administrationsbereich

Nach der oben erwähnten „Ferien-Club“-Vision soll es für die PDA-Anwendung zwei Typen von potenziellen Benutzern geben: so genannte Administratoren, die für die Lauffähigkeit der Anwendung zuständig sind, und Endbenutzer, die die eigentliche Funktionalität der Anwendung in Anspruch nehmen. Aus diesem Grund sollen in der PDA-Anwendung zwei Bereiche existieren: der Administrationsbereich und der Nutzerbereich. In dem Administrationsbereich soll es möglich sein, die Einstellungen, wie Provideradresse, vorzunehmen, und im Nutzerbereich soll die Web-Service-Funktionalität zur Verfügung stehen.

In dem Aktivitätsdiagramm 3.14 sind die Handlungen auf der Benutzeroberfläche der PDA-Anwendung im Admin-Modus dargestellt. Die dazu gehörige Funktionalität ist in dieser Arbeit auf die Konfiguration des Providers beschränkt, aber sie kann nach Bedarf erweitert werden. Wie aus dem Diagramm hervorgeht, ist das Vornehmen der Einstellungen nicht unbedingt zwingend, es können auch Standardeinstellungen für den Web-Service-Anbieter übernommen werden, die in einer Konfigurationsdatei schon vorhanden sind.

Die möglichen Aktivitäten im Nutzer-Modus sind in dem Diagramm 3.15 Seite 45 abgebildet. Nach dem Starten der Anwendung hat der Benutzer die Möglichkeit, einen neuen Auftrag aufzugeben, im Diagramm als „Betätigung des Buttons ’Neue Bestellung abgeben...’“ zu sehen. Nach dem Betätigen des entsprechenden Buttons erscheinen zwei weitere grafische Elemente: das Eingabefeld für den Zielraum und der Button „OK“. Wenn der Service erfolgreich aufgerufen wurde, wird eine zusätzliche funktionale Möglichkeit visuell aktiviert: die Abfrage des Auftragszustandes (auf dem Diagramm als „Visuelle Aktivierung des Buttons ’Zustand abfragen’...“ dargestellt). Danach kann der Zustand der Ausführung abgefragt werden. Wenn der Service nicht gefunden wurde oder andere Schwierigkeiten aufgetreten sind, wird ein Fenster mit dem Text eingeblendet, der die Gründe für den fehlerhaften Ablauf angibt. In diesem Fall kann der Nutzer versuchen, den Service erneut aufzurufen, oder er beendet die Anwendung und meldet die aufgetretene Störung

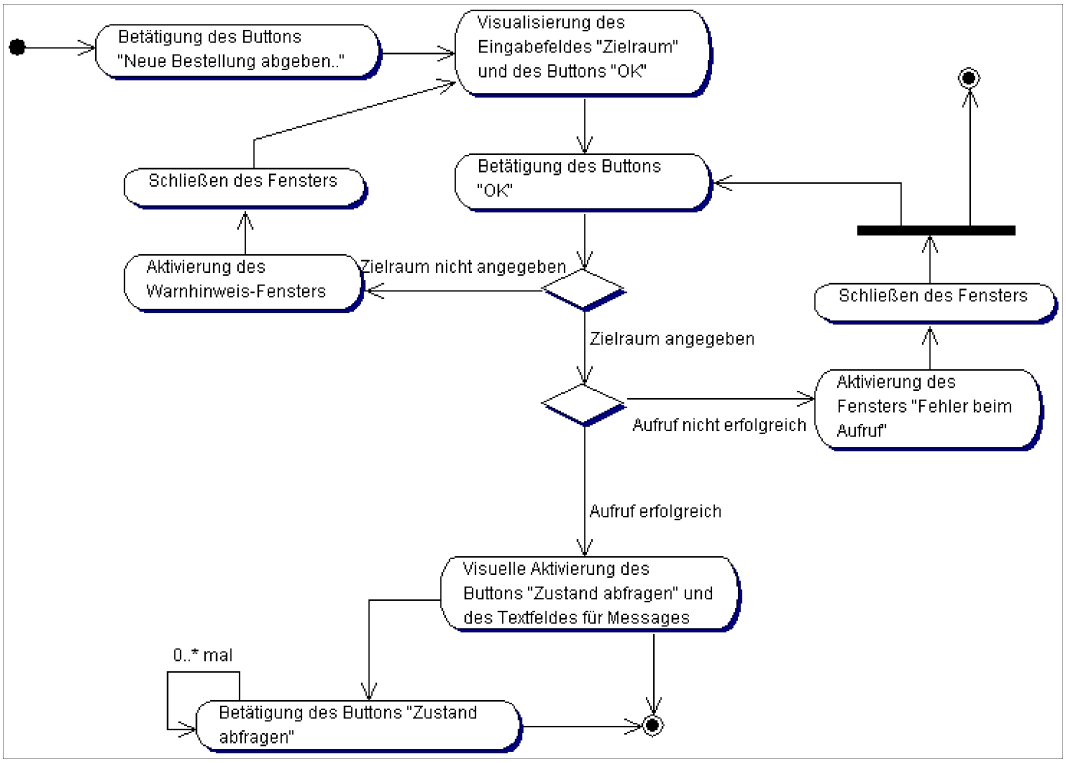

Abbildung 3.15: Aktivitätsdiagramm: PDA-Anwendung - Nutzerbereich

einer Service-Dienststelle (auch Administration genannt).

3.4 Realisierung

Der folgende Abschnitt beschreibt Realisierungskonzepte, die bei der Umsetzung des in den vorigen Abschnitten beschriebenen Architektur-Entwurfs des Roboter-Service-Systems angewendet wurden. Die Realisierung dient als Prüfung der Tragfähigkeit des entworfenen Designs.

3.4.1 Programmiersprachliche Sicht

Aus der Sicht der Programmiersprachen besteht das System aus drei Teilen. Aus Gründen, die im Abschnitt 2.4.3.2 ausführlich erklärt wurden, werden die PDA-Anwendung mit der .NET-Entwicklungsumgebung und die Server-Module in Java entwickelt. Für die Entwicklung der PDA-Anwendung wird dabei C# verwendet (siehe Abbildung 3.16 auf Seite 46), da sie eine aus-

gereifte, sehr mächtige Programmiersprache ist, die in Verbindung mit den Möglichkeiten der .NET-Entwicklungsumgebung einen hohen Komfort für die Entwicklung von Pocket PC-Anwendungen bietet.

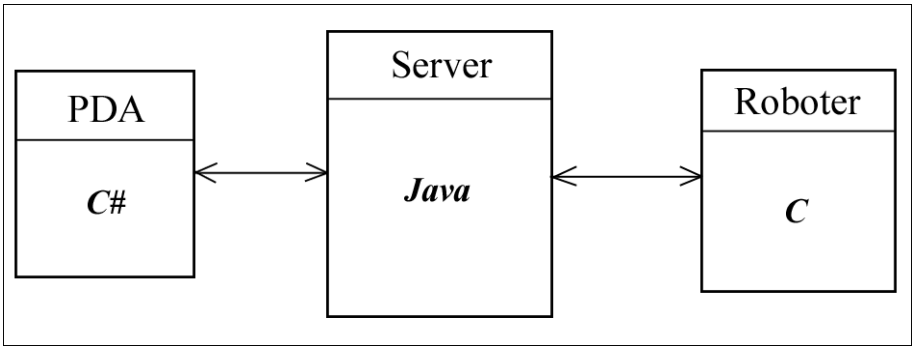

Abbildung 3.16: Programmiersprachliche Sicht

Der Saphira-Client wurde in C realisiert und seine Funktionalität steht in Form einer *dynamic link library* (engl.), kurz DLL, zur Verfügung. Das Roboter-Modul soll also auch in C entwickelt werden (siehe Abbildung 3.16).

Wenn für die Entwicklung eines Software-Systems mehrere Programmiersprachen eingesetzt werden, besteht immer das Problem der Verbindung der verschiedenen Teilen miteinander.

Der PDA-Client und der Server kommunizieren miteinander nur über den Roboter-Web-Service. Da Web Services von sich aus die Interoperabilität anbieten (siehe Kapitel „Analyse“ Abschnitt 2.4.3 auf Seite 19), bestehen keine Schwierigkeiten, diese beiden Teile miteinander zu verbinden und den Übergang C# - Java zu realisieren. Außerdem bietet die .NET-Entwicklungsumgebung einen sehr bequemen Mechanismus für die automatisierte Anbindung der Methoden eines Web Services zu einer Anwendung anhand seiner WSDL-Beschreibung [MSDN].

Das Server-Modul „Saphira-Kommunikationskomponente“ (siehe Abbildung 3.3 Seite 28) soll auf die Funktionalität des Roboter-Moduls zugreifen können,um dem Roboter Anweisungen zu geben. Da dieses Modul in Java implementiert wird, soll an dieser Stelle die Verbindung zwischen Java und C erfolgen. Dazu wird Java Native Interface (JNI) eingesetzt, das ermöglicht, von Java aus einen in einer anderen Programmiersprache geschriebenen Programmcode aufzurufen. Dieser Programmcode muss dafür als DLL vorliegen, allerdings müssen die Methoden einer speziellen Namenskonvention entsprechen, deswegen kann die Saphira-DLL nicht direkt in Java eingebunden werden. Um Saphira-Funktionen trotzdem nutzen zu können, wird daher ein in C

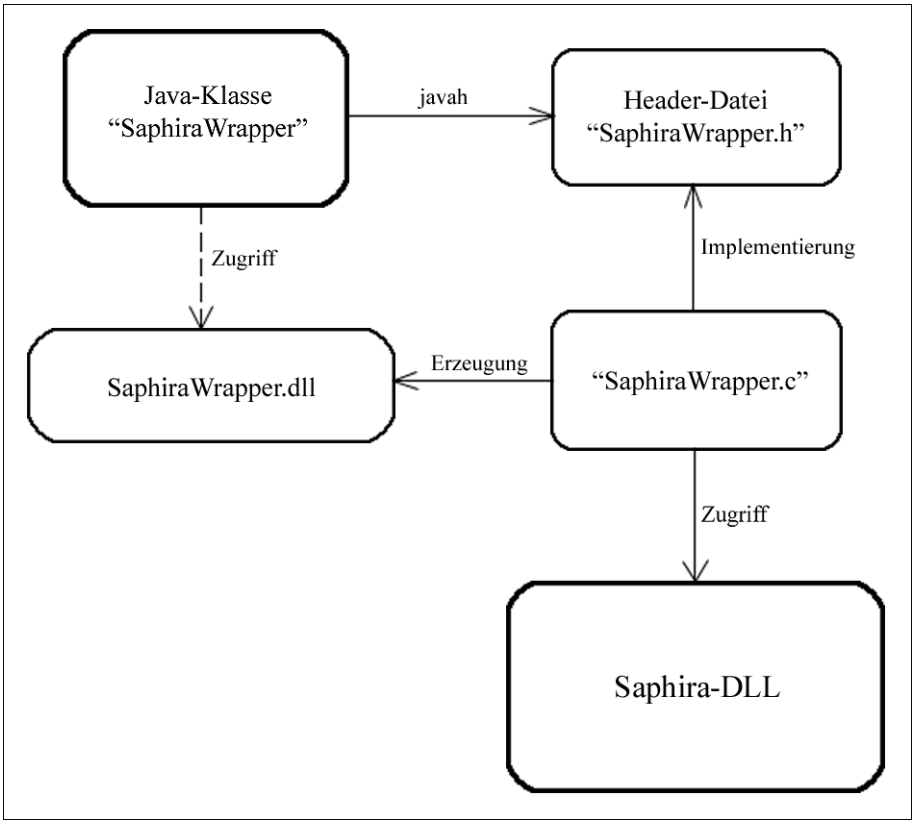

Abbildung 3.17: Verbindung Java - C

geschriebener Wrapper eingesetzt. Die Vorgehensweise ist schematisch in der Abbildung 3.17 dargestellt. In einer Java-Klasse, in der Abbildung als „SaphiraWrapper“ bezeichnet, werden so genannte *native* Methoden deklariert, die in einer anderen Sprache implementiert werden. Aus dieser Klasse wird mit Hilfe des Generators *javah* die Header-Datei SaphiraWrapper.h erzeugt, die nativen Methoden bekommen dabei die für JNI erforderlichen Signaturen. Daraufhin sollen die in der Header-Datei aufgelisteten Methoden in C implementiert werden, wo die notwendigen Saphira-Funktions-Aufrufe stattfinden. Daraus wird eine DLL erstellt, die dann in Java eingebunden und benutzt werden kann.

3.4.2 Umgebung für die Entwicklung

In dem vorigen Abschnitt wurde auf die Verbindung zwischen den Programmiersprachen eingegangen, mit denen das System realisiert wurde. In

diesem Abschnitt werden die für die Entwicklung eingesetzten Umgebungen und Sprach-Versionen ausführlich beschrieben.

Für die Implementierung in Java wurde die zur Entstehungszeit dieser Arbeit aktuellste Version 3.1.0 der Eclipse-Entwicklungsumgebung und Java J2SE 1.4.2 verwendet. Für die Web-Service-Anwendung wurden die letzte Version von Apache Tomcat (5.5.9) und Apache Axis Version 1.1 eingesetzt. Die Tomcat-Anwendung wurde mit Hilfe des Eclipse Tomcat Launcher Plugins Version 3.1beta von der Firma Sysdeo entwickelt. Zur Zeit der Realisierung war Java J2SE in der Version 5.0 zwar schon verfügbar, aber wegen der Unstimmigkeiten zwischen dieser Version und dem Deploy-Mechanismus von Axis wurde auf ihre Nutzung verzichtet.

Für die Entwicklung der PDA-Anwendung in C# und der Library für das Roboter-Modul in C++ wurde die Microsoft Entwicklungsumgebung 2003 eingesetzt, die zur Zeit der Realisierung aktuell war. Für die PDA-Anwendung wurden die Bibliotheken des Compact Frameworks (kurz CF) in der zurzeit letzten Version 1.0 SP3 von Microsoft verwendet, da nur dieser auf dem Pocket PC zur Verfügung steht. Für das Roboter-Modul wurden die .NET-Standard-Bibliotheken benutzt. Als PDA-Simulator wurde Pocket PC 2003 Emulator eingesetzt.

3.4.3 Server-Realisierung

3.4.3.1 Projektaufbau

Der Server wurde in zwei Projekten realisiert. Im Projekt „AssignmentAdministration“ wurden die Server-Module „Auftragsverwaltung“ und „Saphira-Kommunikationskomponente“ umgesetzt und im Tomcat-Projekt „Roboter-Server“ - das Modul „Roboter-Web-Service“. Um die Funktionalität des „AssignmentAdministration“-Projektes im „RoboterServer“ nutzen zu AssignmentAdministration“-Objektes und den Start des Web Services umgesetzt. Der können, wird aus dem ersten Projekt eine JAR-Datei generiert, die dann im zweiten Projekt eingebunden wird. Dieses Projekt wird seinerseits auf dem Tomcat-Web-Container installiert. Nach dem Einspielen auf den Tomcat soll die Anwendung gleich gestartet werden, um die Funktionalität des Servers zu gewährleisten. Dies wird durch die Initialisierung des „Web-Service-Start wird über den Deploy-Mechanismus von Axis mit Hilfe der Datei „deploy.wsdd“ realisiert (siehe Listung in Abbildung 3.18 auf Seite 49), die Informationen über den Namen des Services und seine Methoden sowie den Format der SOAP-Nachrichten enthält. In dieser Arbeit wird der RPC-Stil für die Nachrichtenübermittlung verwendet. Zum Löschen des Web Services aus dem Web-Container steht die Datei „undeploy.wsdd“ zur Verfügung. Bei-

```
<deployment name = "RoboterService"
            xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
            xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">

    <service name="RoboterWebService" provider="java:RPC">
        <parameter name="className" value="service.RoboterWebService"/>
        <parameter name="allowedMethods" value="*" />
        <parameter name="scope" value="Application"/>
    </service>
</deployment>
```

Abbildung 3.18: WSDD-Datei

de Dateien befinden sich im „RoboterServer“-Projekt.

Um die Initialisierung des „AssignmentAdministration“-Objektes zu gewährleisten, wurde im „RoboterServer“-Projekt ein Servlet implementiert, in dem eine Instanz von „AssignmentAdministration“ erzeugt wird. Dieses Servlet wird gleich nach dem Deployen des Projektes auf dem laufenden Tomcat oder beim Hochfahren des Web-Containers gestartet. Das gleiche Servlet sorgt auch dafür, dass beim Herunterfahren des Web-Containers die Anwendung ordnungsgemäß geschlossen wird.

Die Grundkonfiguration der Module des „AssignmentAdministration“-Projektes wird in der XML-Datei *config.xml* festgehalten, die in der Listung in Abbildung 3.19 auf Seite 50 zu sehen ist. Aus dieser Datei bekommen die einzelnen Module ihre Initialisierungsinformationen. Wie schon im Design beschrieben wurde, ist eine der Anforderungen, die an die Modulen „Auftragsverwaltung“ und „Saphira-Kommunikationskomponente“ gestellt wurden, die Flexibilität der Implementierung (siehe Abschnitte 3.3.1 und 3.3.2). Diese wird über die Nutzung der *config.xml*-Datei umgesetzt. So werden hier z.B. die Namen der Klassen angegeben, die in den Fassade-Klassen der entsprechenden Module über die Java Reflection API initialisiert werden. Außerdem steht in dieser Datei die Information über die Art der Verbindung zum Roboter. In dieser Arbeit werden zwei Verbindungsmöglichkeiten angenommen: die Verbindung zu einem Roboter-Simulator, z.B. zum Zweck des Testens der Roboterfunktionalität, oder die Verbindung zu einem Pioneer-Roboter. Letztendlich enthält die *config.xml* die Mapping-Informationen über die Räume, in denen der Roboter eingesetzt werden soll. Der Roboter besitzt eine spezifische Darstellung seiner Umgebung, die ihm von Saphira mit Hilfe einer Map-Datei vermittelt wird. Um den Web-Service-Client unabhängig von dieser spezifischen Darstellung zu machen, werden reale Räume auf die für den Roboter verständliche Nummerierung abgebildet.

```
<?xml version="1.0" ?>

<config>
    <manager>
        <!-- Definition der Implementierungsklasse für Reflektion API -->
        <name>administration.manager.Manager</name>
    </manager>

    <saphiraclient>
        <!-- Definition der Implementierungsklasse für Reflektion API -->
        <name>saphiraclient.client.SaphiraClientImpl</name>
    </saphiraclient>

    <assignmentqueue>
        <limit>128</limit>
    </assignmentqueue>

    <robot>
        <!-- Was für ein Roboter: 1 - Simulator, 2 - echter Pioneer -->
        <case>1</case>
    </robot>

    <roommapping>
        <!-- Mapping der Räume für den Roboter:  -->
        <!-- room - realer Raum, -->
        <!-- door - Map-Eitrag für den Roboter -->
        <map>
            <room>1180a</room>
            <door>1</door>
        </map>

        ...

        <map>
            <room>1164</room>
            <door>16</door>
        </map>
    </roommapping>

</config>
```

Abbildung 3.19: Konfigurationsdatei

Der Zugriff auf die *config*-Datei und darin enthaltenen benötigten Informationen erfolgt über die Methoden der Klasse „XMLConfigReader“, in denen das Lesen aus der XML-Datei mit Hilfe der Bibliotheken JDOM [JDOM] und der im gleichen Packet enthaltenen XPath-Implementierung geschieht.

3.4.3.2 AssignmentQueue

Eines der wichtigsten Elemente des Servers bildet die „AssignmentQueue“. Sie wird mit der ArrayList realisiert, da in der Klassenbibliothek von Java 1.4 keine Warteschlangen-Implementierungen vorhanden sind (was sich allerdings in Java 1.5 geändert hat). Die Größe der Queue ist begrenzt und in der Konfigurationsdatei *config.xml* definiert. Diese Einstellung wird bei der Initialisierung einer Queue-Instanz ausgelesen. Die begrenzte Anzahl der Einträge soll garantieren, dass es nicht zu unkontrollierten System-Fehlern führt, wenn keine Aufträge bearbeitet werden können, z.B. im Fall eines Saphira-Clients-Ausfalls. Wenn die Queue voll ist, werden keine neuen Einträge mehr eingefügt. In einem solchen Fall wird beim Aufruf der Methode *addAssignment* eine Ausnahme vom Typ „QueueOverflowException“ ausgelöst, die signalisieren soll, dass der Versuch, einen neuen Auftrag einzutragen, gescheitert ist.

Die Aufträge werden in die Queue als „Assignment“-Objekte eingefügt. Jedem neuen Auftrag soll eine eindeutige ID verliehen werden. Das wird über die Methode *getNextSequenceNumber()* realisiert, die die Eindeutigkeit der Auftrags-ID garantiert.

Bei der Abfrage des Zustandes eines Auftrages wird nachgeschaut, ob sich der Auftrag noch in der Queue befindet (siehe Sequenzdiagramm 3.7 Seite 35). Das Vergleich der „Assignment“-Objekte erfolgt dann über die Auftrags-ID der jeweiligen Aufträge. Dazu wird die Methode *equals(...)* der Klasse „Assignment“ entsprechend angepasst.

3.4.3.3 Saphira-Kommunikationskomponente

Die Klasse „SaphiraClientImpl“ implementiert das Interface „SaphiraClientIf“ und ist für die Kommunikation mit dem Roboter-Modul zuständig. Sie hat eine boolesche Instanzvariable *ready*, die aussagt, ob der Roboter für eine neue Aufgabe bereit ist. *Ready* kann in zwei Fällen einen negativen Wert aufweisen: wenn sich der Roboter gerade bei der Ausführung einer Aufgabe befindet und wenn keine Verbindung zum Roboter steht. Deswegen wird in der Methode *getState()*, die dem „Auftragsverwaltung“-Modul zur Verfügung steht, zuerst der Status der Verbindung abgefragt. Das geschieht über die Methode *isConnected()*. Wenn keine Verbindung besteht, wird versucht, mit Hilfe der Methode *connect()* eine solche aufzubauen. Wenn der Versuch, die Verbindung herzustellen, scheitert, wird eine Ausnahme vom Typ „RobotConnectException“ ausgelöst. In diesem Fall bekommt die Instanzvariable *ready* den Wert *FALSE*. Die beschriebene Funktionalität ermöglicht, den Zeitpunkt des Anbindens des Roboters an das System zu variieren. So muss

der Roboter nicht gleich beim Starten des Systems angeschaltet werden, sondern dies kann auch später geschehen.

Als Weiters besitzt die Klasse „SaphiraClientImpl“ eine Instanzvariable *robot* vom Typ *int*. Die beinhaltet die Information darüber, ob die Verbindung mit einem Pioneer-Roboter oder mit einem Roboter-Simulator erfolgen soll. Diese Information wird bei der Initialisierung der Klasse aus der *config*-Datei ausgelesen.

Für die eigentliche Verbindung mit der Roboter-DLL steht die Klasse „SaphiraWrapper“ zur Verfügung, die als ihr Stellvertreter fungiert und in der die Instanzvariable *wrapper* der Klasse „SaphiraClientImpl“ initialisiert wird.

Für den aktuellen Auftrag steht die Instanzvariable *job* vom Typ „Job“. Zum Anstoßen der Ausführung einer neuen Aufgabe steht die Methode *deliverArtefact(...)* zur Verfügung, in der die Methode *execute(...)* des „SaphiraWrapper“-Objektes aufgerufen wird. Das „Job“-Objekt wird als Parameter an diese Methode übergeben, was die Verfolgung der Auftragszustands-Änderungen bei der Ausführung im Roboter-Modul ermöglicht. Wenn die Aufgabe erfolgreich ausgeführt wurde, wird die ID dieses Auftrages in den als ArrayList implementierten Container *finishedJobs* eingetragen. Beim Scheitern der Ausführung wird diese ID in den *failedJobs*-Container eingefügt, der ebenfalls als ArrayList realisiert wird. Somit wird ein „Verlieren“ eines Auftrages ausgeschlossen. Die Ermittlung des Zustandes eines Auftrages wird in der Methode *getStateForJob(...)* umgesetzt.

3.4.3.4 Manager

Die Klassen „Manager“ und „ManagerThread“ des Moduls „Auftragsverwaltung“ sind für die Aufnahme und das Weiterleiten neuer Aufträge an die „Saphira-Kommunikationskomponente“ verantwortlich. Sie beide können auf die „AssignmentQueue“ zugreifen. Um die inkonsistenten Zustände der Queue zu vermeiden, werden das Einfügen und das Holen eines Auftrages als synchronisierte Zugriffe auf die Queue realisiert.

Die Klasse „Manager“ besitzt eine Instanzvariable *map* vom Typ „HashMap“, in der die in der *config*-Datei definierten Mapping-Daten für die zulässigen Räume enthalten sind. Vor der Annahme eines neuen Auftrages wird in der Methode *addNewAssignment(...)* zuerst überprüft, ob die als Parameter übergebene Raumnummer in der *map* vorhanden ist, für den Auftrag demnach ein zulässiger Zielraum angegeben wurde. Wenn dies nicht der Fall ist, wird eine Ausnahme vom Typ „IllegalParameterExeption“ ausgelöst und die Ausführung der Auftrag-Annahme abgebrochen.

Das Anstoßen der Bearbeitung der in der Queue stehenden Aufträge über-

nimmt die Methode *run()* der Klasse „ManagerThread“, die von der Klasse *java.lang.Thread* abgeleitet ist. Das Ausführen einer neuen Aufgabe wird nur dann initiiert, wenn der Status der „Saphira-Kommunikationskomponente“, welchen „ManagerThread“ über die im vorigen Abschnitt beschriebene Methode *getState()* abfragt, *TRUE* aufweist.

Nach dem Holen eines Auftrages aus der „AssignmentQueue“ wird dieser aus der Queue gelöscht. Seine ID und der Zielraum werden an das „Saphira-Communication“-Objekt in der im vorigen Abschnitt beschriebenen Methode *deliverArtefact(...)* zur weiteren Bearbeitung übergeben.

Beim Beenden seiner Arbeit ruft „ManagerThread“ die Methode *exit()* der Fassade-Klasse „SaphiraCommunication“ auf, die ihrerseits ein ordnungsgemäßes Abschließen der Arbeit der „Saphira-Kommunikationskomponente“ initiiert.

3.4.3.5 Roboter-Web-Service

Der Roboter-Web-Service bietet dem Consumer entsprechend seiner im Design beschriebenen Funktionalität zwei Methoden an:

addNewAssignment(...)
getStateForAssignment(...)

Beim Ausführen der Methode *addNewAssignment(...)* können zwei Ausnahmen auftreten, die schon in den vorigen Abschnitten beschrieben wurden: „IllegalParameterExeption“ und „QueueOverflowException“. Als Reaktion darauf werden in dieser Methode entsprechend zwei andere Ausnahmen ausgelöst: „IllegitimateRoomException“ und „FailedInsertException“. Die beiden sind von der Klasse „org.apache.axis.AxisFault“ abgeleitet, was garantiert, dass beim Auftritt einer dieser Ausnahmen ein besonderer SOAP-Fault (Fehler) generiert wird, der dem Web-Service-Client eine Information über die konkrete Ursache des Fehlers gibt und ihm so erlaubt, fallspezifisch darauf

```
...

  <operation name="addNewAssignment" parameterOrder="paramRoom">
    <input name="addNewAssignmentRequest" message="tns:addNewAssignmentRequest" />
    <output name="addNewAssignmentResponse" message="tns:addNewAssignmentResponse" />
    <fault name="FailedInsertException" message="tns:FailedInsertException" />
    <fault name="IllegitimateRoomException" message="tns:IllegitimateRoomException" />
  </operation>

...
```

Abbildung 3.20: WSDL-Datei: Faults für *addNewAssignment(...)*

zu reagieren. Die spezifischen Faults werden in der WSDL-Datei beschrieben,

die von Axis für den Web Service automatisch generiert wird. Der entsprechende Ausschnitt aus der WSDL-Datei ist in der Listung in Abbildung 3.20 auf Seite 53 zu sehen.

3.4.4 Realisierung der PDA-Anwendung

Die PDA-Anwendung besteht aus zwei Formularen. Das erste Formular dient zum Aufrufen des Web Services, das zweite zum Vornehmen der Provider-Einstellungen. Auf den Abbildungen 3.21 und 3.22 sind die Screenshots der beiden Formulare zu sehen.

Abbildung 3.21: PDA-Anwendung: Programmstart

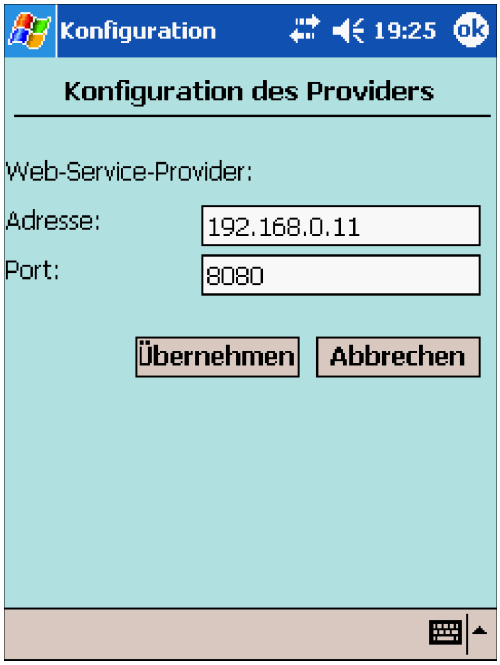

Abbildung 3.22: PDA-Anwendung: Einstellungen

Das Design des Web-Service-Formulars ist so konzipiert, dass es dem Benutzer bei der Bedienung des Programms hilft (siehe Abschnitt 3.3.4). So ist beim Starten des Programms nur der Button zur Aufgabe einer neuen Bestellung sichtbar, da andere Steuerelemente, wie z.B. zur Abfrage des Ausführungszustandes der Bestellung, zu diesem Zeitpunkt überflüssig sind (siehe Abbildung 3.21). Wenn dieser Button betätigt wurde, erscheinen im Formular weitere, in diesem Zusammenhang notwendige, Steuerelemente. Die Interaktionen des Formulars im Verlauf der Anwendung sind auf den Screenshots 3.23 und 3.24 Seite 55 abgebildet.

Abbildung 3.23: PDA-Anwendung: Neuer Auftrag

Abbildung 3.24: PDA-Anwendung: Ausführung

Für die Provider-Einstellungen und die beim Web-Service-Aufruf erhaltene Auftrags-ID wird die Klasse „Modell“ benutzt, in der diese Daten in entsprechenden Instanzvariablen gehalten werden. Die Provider-Daten werden beim Starten des Programms aus der Konfigurationsdatei *provider_einstellungen.xml* gelesen. Wie schon erwähnt wurde, können diese Einstellungen im Laufe des Programms geändert werden. In einem solchen Fall wird die neue Konfiguration in dieser Datei abgespeichert, damit beim nächsten Anwendungsstart die aktuellen Einstellungen übernommen werden können. Das Lesen und das Schreiben in die XML-Datei geschieht über die Methoden *readXml()* und *writeXml()* der Klasse „Xml_Manipulator“, in der diese Funktionalität mit Hilfe der Klassen der .NET-Bibliothek „System.XML“ implementiert wurde.

Bei der Abfrage des Auftragszustandes bekommt die PDA-Anwendung einen Integer-Wert als Antwort. Um dieses Ergebnis besser auswerten und interpretieren zu können, werden mögliche Zustände in der abstrakten Klasse „Form_Konfig“ als Konstanten deklariert.

Kritische Stellen in dieser Anwendung sind die Aufrufe der Web-Service-Methoden, die zu Ausnahmen führen können, z.B. bei einem erfolglosen Versuch, mit dem Server in Verbindung zu treten. Im Programm werden des-

wegen Exceptions-Behandlungen realisiert, die beim Auftritt eines Fehlers weiterhin einen ordnungsgemäßen Ablauf des Programms garantieren und dem Benutzer den Fehler mitteilen.

3.4.5 Realisierung des Roboter-Moduls

Wie schon im Abschnitt 3.2.3 erklärt wurde, wird für das Roboter-Modul ein externes Programm verwendet. Dieses Programm ist in der Sprache Colbert in Form von *activities* (engl.) geschrieben (siehe dazu Abschnitt 2.4.4). Die in Colbert geschriebenen Aktivitäten können aus einem C-Programm aufgerufen werden. Dazu stellt Saphira entsprechende Funktionen zur Verfügung.

Das Roboter-Modul ist also eine in C geschriebene DLL, die das Verbindungsglied zwischen dem Server-Modul „SaphiraKommunikationskomponente“ und der Funktionalität der Saphira-Software und des in Colbert geschriebenen externen Programms darstellt. Diese DLL enthält die Implementierung der nativen Methoden der im Abschnitt 3.4.3.3 schon beschriebenen Java-Klasse „SaphiraWrapper“:

init(...)
connectToRobot(...)
disconnectFromRobot()
isConnected()
execute(...)

In der Methode *init(...)* wird die Saphira-Laufumgebung gestartet und der Versuch unternommen, eine Verbindung zum Roboter aufzubauen. Die Information darüber, ob die Verbindung mit einem Simulator oder einem Pioneer-Roboter stattfinden soll, wird dieser Methode als Parameter übergeben.

In der Methode *execute(...)* erfolgt der eigentliche Aufruf des Colbert-Programms. Dieser Methode soll ein Objekt von der Java-Klasse „Job“ als Parameter übergeben werden. Dies wird über den JNI Cast-Mechanismus möglich. Die Methode erhält ein JNI-Objekt als Parameter. Dieses Objekt wird dann auf das konkrete erwartete Objekt abgebildet, wie seine Properties auch, deren Werte dann geändert werden können. Beim „Job“-Objekt in der Methode *execute(...)* ist das die Property *state*, die ihren Wert ändern soll. Zurzeit haben die Zustandsänderungen des „Job“-Objektes eine grobe Granularität, da im Colbert-Programm keine Auskünfte über den Fortschritt der Ausgabe-Ausführung vorgesehen sind. Unmittelbar vor der eigentlichen Roboter-Ausführung bekommt diese Property den Wert *IN_EXECUTE*, nach der Ausführung den Wert *FINISHED*.

Kapitel 4

Fazit und Ausblick

4.1 Ergebnisse

Die am Anfang dieser Arbeit gesetzten Hauptziele wurden erreicht: Es wurde erfolgreich eine Architektur für ein verteiltes Roboter-Service-System entwickelt, das folgende drei Komponenten aufweist:

- PDA-Anwendung
- Server-Anwendung
- Serviceroboter

Die an das System gestellten Anforderungen wurden in dieser Arbeit erfüllt. Das Roboter-Service-System besitzt folgende Eigenschaften:

- Das System bietet seinen Nutzern die komfortable Möglichkeit, Dienste eines Serviceroboters mittels Web Services in Anspruch zu nehmen.
- Es ermöglicht so die Nutzung des Serviceroboters von beliebigen Client-Geräten aus.
- Es übernimmt die Verwaltung von Service-Anfragen, also die Annahme von Aufträgen, ihre Weiterleitung an den Roboter und die Initiierung der Auftragsbearbeitung.
- Es ermöglicht die Multiuser-Fähigkeit des Serviceroboters.

Die Arbeit hat gezeigt, dass sich der Einsatz von Web-Service-Technologien sehr gut für die Entwicklung von lose gekoppelten Systemen eignet, in denen keine Echtzeit-Übermittlung erforderlich ist. Im konkreten Fall gilt das auch

für die Verwendung von Web Services als Client-Kommunikationsmöglichkeit zu Serviceroboter-Systemen.

Web Services haben sich als leicht zu implementieren erwiesen. Obwohl für den Web-Service-Provider und den Consumer verschiedene Entwicklungsplattformen benutzt wurden, sind keine Schwierigkeiten bei der Realisierung der in dieser Arbeit an beide gestellten Anforderungen aufgetreten. Beide Plattformen – Apache Axis und Microsoft .NET – bieten sehr praktische Werkzeuge für die Entwicklung von Web Services und von ihren Consumer. Allerdings sind Unstimmigkeiten zwischen Axis und Java 1.5 aufgetreten, welche sich als Fehler bei dem Einsatz von Axis-Deployment-Mechanismus gezeigt haben. Aus diesem Grund wurde auf die Verwendung von Java 1.5 in dieser Arbeit verzichtet. Bei der Entwicklung des Web-Service-Consumers hat auch das .NET-Framework ein Mangel gezeigt. Hier gibt es nur einen allgemeinen Mechanismus zur Behandlung von Fehlern (Faults), die beim Web-Service-Aufruf auftreten können. Sie alle werden in eine Ausnahme vom Typ *SoapException* konvertiert, was die Bearbeitung von benutzerdefinierten Server-seitigen Ausnahmen erschwert. Die spezifische, vom Ausnahme-Typ abhängige Behandlung ist zwar möglich, ist allerdings unkomfortabel, da die Information über den Ausnahme-Typ nur in Form von *String* zur Verfügung steht.

Die .NET-Entwicklungsumgebung wurde ebenfalls erfolgreich für die Realisierung der PDA-Anwendung eingesetzt, insbesondere sind hier die für die Entwicklung von GUI zur Verfügung stehenden Werkzeuge zu erwähnen.

Bei der Realisierung des Roboter-Moduls sind einige Schwierigkeiten aufgetreten, welche die Anbindung und den Aufruf von Colbert-Funktionen von C aus betreffen, die aber letztendlich doch bewältigt wurden. Diese Schwierigkeiten sind unter anderem auf die mangelhafte Dokumentation von Saphira- und Colbert-Umgebung zurückzuführen.

Insgesamt hat diese Arbeit gezeigt, dass die Nutzung von Serviceroboter-Diensten über Web Services realistisch und vielversprechend ist. Diese Richtung der Entwicklung von lose gekoppelten verteilten Service-Systemen mit dem Einsatz von mobilen Robotern besitzt ein bedeutendes Potenzial, das einen interessanten Stoff für weitere Arbeiten bietet.

4.2 Ausblick

Obwohl die an das Roboter-Service-System gestellten Anforderungen in dieser Arbeit erfüllt wurden, gibt es einige Ideen, die aus Zeitgründen nicht umgesetzt wurden, die aber Konzepte für die Weiterentwicklung des Systems und seiner Funktionalität liefern könnten. Hier sind einige von ihnen:

- Persistenz

  Dazu gehört sowohl die Persistenz der am Server ankommenden Aufträge für den Fall eines Server-Ausfalls als auch die Persistenz der Auftrags-ID in der PDA-Anwendung, die der PDA-Client beim erfolgreichen Aufruf des Roboter-Web-Services erhält.

- Transaktionen

  Zur Zeit wird der Auftrag bei seiner fehlerhaften Ausführung in einer Liste mit „gescheiterten“ Aufträgen registriert, damit der PDA-Client die Möglichkeit hat, das Fehlschlagen seines Auftrages zu erfahren. Aus der Auftragsqueue wird er unwiderruflich gelöscht. Um dennoch ein zufriedenstellendes Ergebnis zu erhalten, muss der Client den gleichen Auftrag noch einmal abgeben. Es wäre empfehlenswert, einen Transaktionsmechanismus zu entwickeln, der die fehlgeschlagenen Aufträge zurück in die Queue einfügen würde, um später nochmals zu versuchen, sie auszuführen. Die Anzahl der Versuche könnte in einer Konfigurationsdatei definiert werden. Die Möglichkeit zur Verfolgung des Bearbeitungsfortschrittes sollte dabei trotzdem garantiert werden.

- Events

  Vom Server wird derzeitig ein einfacher Mechanismus für periodisches Nachschauen in der Auftragsqueue und das Abholen von neuen Aufträgen benutzt. Alternativ könnten dafür Events verwendet werden, die aber in dieser Arbeit aus Zeitgründen nicht eingesetzt wurden, da das keinen Einfluss auf das Konzept des entwickelten Roboter-Service-Systems hätte.

- Sicherheit

  In dieser Arbeit wurde die Sicherheit der Kommunikation zwischen dem Server und dem Web-Service-Client außer Acht gelassen. Mögliche Erweiterungen der System-Eigenschaften wären die Authentifikations- und Autorisations-Mechanismen für den Client und ein sicherer Datenaustausch über Web Services.

Ein weiteres Beispiel der erweiterten Funktionalität des Roboter-Service-Systems kann das Reduzieren der System-Abhängigkeit vom Roboter sein. Bei einem Ausfall des Roboters könnte das System trotzdem gewährleisten,

dass die von Clients (in dieser Arbeit von PDAs) kommenden Aufträge immer zu einem zufriedenstellenden Ergebnis ausgeführt werden. Wenn der Serviceroboter in der Hotel- oder Ferienclub-Branche eingesetzt wird, könnten die Aufträge an das „menschliche“ Bedienungspersonal umgeleitet werden, das dort üblicherweise vorhanden ist. Das kann z.B. in Form einer Nachricht (einer SMS) an die zutreffende Person geschehen. In diesem Fall sollte die Verwaltung des Servicepersonals und der Verteilung der Aufgaben auch vom System übernommen werden. Noch nicht bearbeitete Aufträge könnten auch an eine Anwendung mit einem Web-Interface umgeleitet werden, wo sie dann gesammelt und für das Bedienungspersonal als offene Aufgaben zur Verfügung gestellt werden. Die Personen aus der Bedienung könnten über diese Anwendung erfahren, ob noch Dienste zu erledigen sind, und würden dann die Möglichkeit haben, sich neue Aufgaben „zuzuschreiben“.

Eine interessante Erweiterung des Roboter-Service-Systems stellt die Benutzung von RFID-Technik (Radio Frequency Identification (engl.)) dar. Zurzeit wird von dem Nutzer des Systems vorgegeben, wohin der Roboter fahren soll. Wenn ein Nutzer also einen Kaffee bestellt, ist er an den Raum gebunden, den er bei der Bestellung-Aufgabe angegeben hat, bis der Roboter mit der Bestellung zu ihm kommt. Die Ausstattung der PDAs mit RFID-Transpondern und der Einsatz-Umgebung des Systems mit RFID-Lesegeräten würde neue Möglichkeiten für die Flexibilität des Services eröffnen. Der Benutzer könnte sich in der Umgebung frei bewegen. Das System würde ihn jederzeit lokalisieren und dem Roboter die neue Position des Benutzers mitteilen können. Dazu müssen allerdings die Grenzen der RFID-Technologie und der Roboter-Funktionalitätsmöglichkeiten genau untersucht werden. An der HAW Hamburg wurden schon Arbeiten mit RFIDs durchgeführt. Als Beispiel kann hier die Diplomarbeit von Martin Stein [ST05] aufgeführt werden. Die Verknüpfung des Roboter-Service-Systems mit dieser Technologie bietet Stoff für weitere Entwicklungen.

Literaturverzeichnis

[HA04] Klaus Hauptfleisch, Markt für mobile Geräte im Aufwind, URL: http://www.tecchannel.de/hardware/1461, 19.08.2004, Zugriffsdatum: 23.03.2005

[AIBO] Roboter-Hund AIBO von Sony, URL: http://www.aibo-europe.com/index.asp, Zugriffsdatum: 28.06.2005

[ARTH] Der mobile Roboter „Arthur“, Universität Tübingen, Ein mobiler Roboter mit elektronischer Nase und biomimetischem Sonar, URL: http://www-ra.informatik.uni-tuebingen.de/forschung/outdoor/welcome.html, Zugriffsdatum: 27.06.2005

[GO03] G. Görz, C.-R. Rollinger, J. Schneeberger (Hrsg.), Handbuch der Künstlichen Intelligenz, 4. Auflage, 2003, ISBN: 3-486-27212-8

[MA05] Alexandros Matsikis, Bildgestütztes Teach-In eines mobilen Manipulators in einer virtuellen Umgebung, eine Dissertation an der Rheinisch-Westfälischen Technischen Hochschule Aachen, 2005, URL: http://sylvester.bth.rwth-aachen.de/dissertationen/2005/016/05_016.pdf, Zugriffsdatum: 28.06.2005

[BU02] Prof. Dr. Thomas Burkhardt, Margret Gross-Hardt, Jan Murray, Seminararbeit im Rahmen von „Softwareagenten: Theorie und Praxis“ Thema 2: „Agentenarchitekturen“, 2002, URL: http://www.uni-koblenz.de/~klaasd/Downloads/Agentenarchitekturen.pdf, Zugriffsdatum: 28.06.2005

[SP05] Siegfried Spolwig, Karel D. Robot - der Delphi-Karel, 2004-2005, URL: http://www.oszhdl.be.schule.de/gymnasium/faecher/informatik/delphi_karel/index.htm, Zugriffsdatum: 27.06.2005

[TS01] Tschernobyl nach der Abschaltung, 22. Mai 2001 URL: http://archives.arte-tv.com/hebdo/archimed/20010522/dtext/sujet4.html, Zugriffsdatum: 28.06.2005

[NASA] NASA Mars Exploration Program, URL: http://marsprogram.jpl.nasa.gov/, Zugriffsdatum: 29.06.2005

[RC05] RoboCup German Open 2005, URL: http://www.ais.fraunhofer.de/GO/2004

[KN03] Alois Knoll, Roboter für Menschen - Zielvorstellungen und Ansätze für autonome smarte Serviceroboter, in: F. Mattern (Hg.) Total vernetzt - Szenarien einer informatisierten Welt, 2003 URL: http://sunknoll1.informatik.tu-muenchen.de/~knoll/publikationen/roboter-fuer-menschen.pdf, Zugriffsdatum: 29.06.2005

[IPA] Care-O-bot, Fraunhofer-Institut für Produktionstechnik und Automatisierung, URL: http://www.care-o-bot.de/, Zugriffsdatum: 30.06.2005

[TB] Tourbot, URL: http://www.ics.forth.gr/tourbot, Zugriffsdatum: 30.06.2005

[IASG] Intelligent Autonomous Systems Group, Institut für Informatik, Universität Bonn, URL: http://www.informatik.uni-bonn.de/~rhino/, Zugriffsdatum: 30.06.2005

[BL99] Florian Boldt, Hauke Löhler, Realisierung eines autonomen Büroboten Roboters an der FH-Hamburg, Studienarbeit, 1999

[HA05] Franz J. Hauck, Architekturen für verteilte Internetdienste, Universität Ulm, 2005, http://www-vs.informatik.uni-ulm.de/teach/ss05/avid/, Zugriffsdatum: 03.07.2005

[BR03] Prof. Dr. Hans vor der Brück, Betriebssysteme, Fachhochschule Augsburg, Fachbereich Informatik, 2003, URL: http://bs4u.informatik.fh-augsburg.de/skript-bs/bs, Zugriffsdatum: 04.07.2005

[DEGE] Arne Degenring, Konzeption und Implementierung zur Integration eines SAS Servers in SOAP-basierte Web Service Umgebungen, Diplomarbeit, URL: http://www.degenring.de/webservices/diplomarbeit.html, Zugriffsdatum: 03.07.2005

[TAN1] Andrew S. Tanenbaum, Moderne Betriebssysteme, 2. überarbeitete Auflage, deutsche Ausgabe, 2003, ISBN: 3-8273-7019-1

[TAN2] Andrew S. Tanenbaum, Maarten van Steen, Verteilte Systeme: Grundlagen und Paradigmen, 2003, ISBN: 3-8273-7057-4

[CO02] George Coulouris, Jean Dollimore, Tim Kindberg, Verteilte Systeme: Konzepte und Design, 3. Auflage, 2002, ISBN: 3-8273-7022-1

[MONO] Mono, Open-Source-Projekt zur Implementierung von .NET, URL: http://www.mono-project.com, Zugriffsdatum: 05.07.2005

[HE05] Mono: .Net für Linux & Co., Heise-Online, 09.02.2005, URL: http://www.heise.de/newsticker/result.xhtml?url=/newsticker/meldung/56196&words=Mono, Zugriffsdatum: 05.07.2005

[RMI] Java Remote Method Invocation (Java RMI), Sun Developer Network, URL: http://java.sun.com/products/jdk/rmi/, Zugriffsdatum: 05.07.2005

[RMI-IIOP] Java RMI over IIOP, Java 2 SDK, Standard Edition Documentation, URL: http://java.sun.com/j2se/1.4.2/docs/guide/rmi-iiop, Zugriffsdatum: 05.08.2005

[HP] Hewlett Packard, URL: http://www.hp.com, Zugriffsdatum: 13.04.2005

[WO02] Jörg Franz Wollert, Das Bluetooth Handbuch, 2002, ISBN: 3-7723-5323-1

[IEEE] Institute of Electrical and Electronics Engineers, URL: http://www.ieee.org, Zugriffsdatum: 13.04.2005

[KA03] Dr. Franz-Joachim Kauffels, Lokale Netze, Band 1, 15. Auflage, 2003, ISBN: 3-8266-0994-8

[NE01] Edgar Nett, Michael Mock, Martin Gergeleit, Das drahtlose Ethernet, 2001, ISBN: 3-8273-1741-X

[KR04] Dirk Krafzig, Karl Banke, Dirk Slama, Enterprise SOA: Service-Oriented Architecture Best Practices, 2004, ISBN: 0-13-146575-9

[XML] Extensible Markup Language, URL: http://www.w3.org/XML, Zugriffsdatum: 21.04.2005

[HA04] Tobias Hauser, Ulrich M. Löwer, Web Services: Die Standards, 2004, ISBN: 3-89842-393-X

[DW] UserLand Software Inc., URL: http://davenet.scripting.com, Zugriffsdatum: 06.07.2005

[CH03] David A. Chappell, Tyler Jewell, Java Web Services, deutsche Ausgabe, 2003, ISBN: 3-89721-284-6

[WA04] Dapeng Wang (Hrsg.), Thomas Bayer, Thilo Frotscher, Marc Teufel, Java Web Services mit Apache Axis, 2004, ISBN: 3-935042-57-4

[BE01] Urban Bettag, Web-Services, 2001, URL: http://www.gi-ev.de/informatik/lexikon/inf-lex-web-services.shtml, Zugriffsdatum: 27.06.2005

[GOO] Google Web APIs für Web-Service-basierte Nutzung des Google-Suchsystems, URL: http://www.gi-ev.de/informatik/lexikon/inf-lex-web-services.shtml, Zugriffsdatum: 03.07.2005

[AM] ActivMedia, URL: http://www.activmedia.com, Zugriffsdatum: 26.04.2005

[AIC] Artificial Intelligence Center, URL: http://www.ai.sri.com, Zugriffsdatum: 27.04.2005

[PM99] Pioneer 2 Mobile Robot - Operation Manual, ActivMedia Robotics, 1999

[SA99] Kurt G. Konolige, Pioneer Mobile Robots – Saphira Software Manual 6.1, Artificial Intelligence Center SRI International, 1999

[CO99] Kurt G. Konolige, COLBERT: A Language for Reactive Control in Saphira, Artificial Intelligence Center SRI International, 1999

[KO96] Kurt G. Konolige, Karen Myers, The Saphira Architecture for Autonomous Mobile Robots, Artificial Intelligence Center, 14. November 1996

[GA04] Erich Gamma, Richard Helm, Ralph E. Johnson, John Vlissides, Entwurfsmuster, 2004, ISBN: 3-8273-2199-9

[BA99] Heide Balzert, Lehrbuch der Objektmodellierung: Analyse und Entwurf, 1999, ISBN: 3-8274-0285-9

[WE98] Ivo Wessel, GUI-Design: Richtlinien zur Gestaltung ergonomischer Windows-Applikationen, 1998, ISBN: 3-446-19389-8

[MSDN] MSDN Library: .NET-Entwicklung, URL: http://www.microsoft.com/germany/msdn/library/net/default.mspx, Zugriffsdatum: 26.05.2005

[ST] Alexander Seizeller, E-Mail: alex-seizeller@yahoo.de; Kien Nghia Tran, E-Mail: ktran11@yahoo.de

[GOR] Gesellschaft für Operations Research (GOR) e.V., URL: https://gor.uni-paderborn.de, Zugriffsdatum: 06.07.2005

[BELE] Bertelsmann Lexikon Online, URL: http://www.wissen.de/xt/default.do?MENUNAME=Suche&SEARCHTYPE=topic&query=Operations+Research, Zugriffsdatum: 06.07.2005

[JMS] Java Message Service, URL: http://java.sun.com/products/jms/index.jsp, Zugriffsdatum: 01.08.2005

[EC04] Claudia Eckert, IT-Sicherheit: Konzepte, Verfahren, Protokolle, 2004, ISBN: 3-486-20000-3

[SC00] Hans-Jürgen Scheibl, Visual C++ 6.0 für Einsteiger und Fortgeschrittene, 2000, ISBN: 3-446-19548-3

[LO98] Dirk Louis, C/C++-Kompendium, 1998, ISBN: 3-8272-5386-1

[UL04] Christian Ullenboom, Java ist auch eine Insel, 4. Auflage, 2004, ISBN: 3-89842-526-6, URL: http://www.galileocomputing.de/openbook/javainsel4/index.htm

[AU99] Calvin Austin, Monica Pawlan, Advanced Programming for the Java 2 Platform, Sun Developer Network: Tutorials & Code Camps, November 1999, URL: http://java.sun.com/developer/onlineTraining/Programming/JDCBook/index.html

[SC02] Arne Schäpers, Rudolf Huttary, Dieter Bremes, C# Kompendium, 2002, ISBN: 3-8272-6015-9

[DR03] Peter Drayton, Ben Albahary, Ted Neward, C# in a Nutshell, Deutsche Ausgabe, 2003, ISBN: 3-89721-299-4

[JDOM] JDOM-Projekt: eine komplette, Java-basierte Lösung zum Zugreifen, Manipulieren und Ausgeben von XML-Daten aus dem Java-Code, URL: http://www.jdom.org, Zugriffsdatum: 26.07.2005

[ST05] Martin Stein, Entwicklung eines auf RFID basierenden mobilen Objekt-Tracking-Systems, Diplomarbeit, HAW Hamburg, 2005

Anhang A

Inhalt der CD

Zu dieser Arbeit wird eine CD beigefügt. Die Verzeichnis-Struktur dieser CD ist in der Abbildung A.1 anschaulich dargestellt.

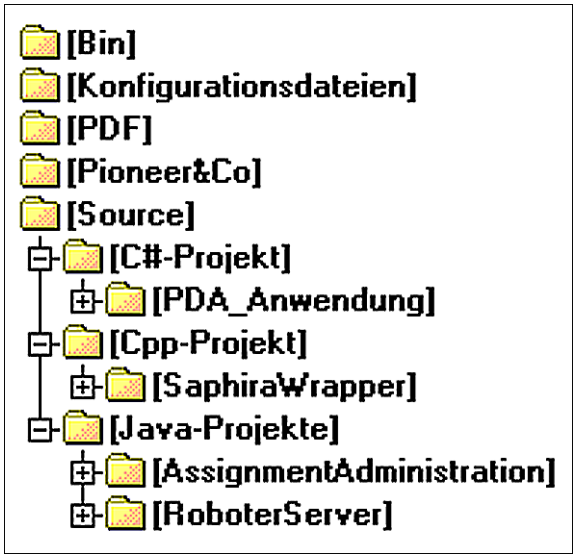

Abbildung A.1: Inhalt der CD

Die CD enthält folgende Informationen:

- Das Verzeichnis **Bin** enthält eine WAR-Datei mit der kompletten Server-Anwendung für Tomcat Web-Container, die kompilierte PDA-Anwendung und die „SaphiraWrapper“-DLL für den Roboter.
- Im Verzeichnis **Konfigurationsdateien** befinden sich die XML-Konfigurationsdateien für die PDA-Anwendung und für den Server.

- Das Verzeichnis **PDF** enthält diese Arbeit im PDF-Format.
- Im Verzeichnis **Pioneer&Co** stehen die für den Pioneer-Roboter benötigten Dateien: die Map- und die World-Datei mit der Einsatzumgebung des Roboters, das Colbert-Programm *goToDoor.ACT* von Alexander Seizeller und Kien Nghia Tran und das Test-Colbert-Programm *patrol.ACT*.
- Das Verzeichnis **Source** beinhaltet den Quellcode der Projekten „PDA-Anwendung“, “SaphiraWrapper“ (das Roboter-Modul), „Assignment-Administrattion“ (das „Auftragsverwaltung“-Modul und die Saphira-Kommunikationskomponente) und das Tomcat-Projekt „RoboterServer“ mit dem Roboter-Web-Service.

Versicherung über die Selbständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den	
Ort, Datum	Unterschrift