Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Studienarbeit

Oliver Schmidt

Entwicklung und Aufbau eines Kamera-Service basiert auf “.net”-Technologie

Fachbereich Elektrotechnik und Informatik
Department of Electrical Engineering and Computer Science

Oliver Schmidt

Entwicklung und Aufbau eines Kamera-Service basiert auf „.net“-Technologie

Studienarbeit
im Studiengang Technische Informatik

am Fachbereich Elektrotechnik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Kai von Luck

Abgegeben am 03.02.2005

Oliver Schmidt

Thema der Diplomarbeit

Entwicklung und Aufbau eines Kamera-Service basiert auf „.net“-Technologie

Stichworte

Objekterkennung, Client-Server Kommunikation, wireless-LAN, PDA, Firewire-Kameras, „.net“.

Kurzzusammenfassung

In dieser Arbeit wird ein System entwickelt, mit dem mobile Geräte die Positionen von bestimmten Objekten, die sich in einem überwachten Feld bewegen, von einem Server abfragen können.
Die mobilen Geräte kommunizieren über WLAN mit dem Server. Beim Server können Objekte registriert werden, deren Position dann mit Hilfe von Bilderkennungs-Algorithmen aus dem jeweils aktuellem Kamera-Bild berechnet wird. Die Positionen der registrierten Objekte können dann jederzeit abgerufen werden.
Die Realisierung baut auf der „.net“-Technologie auf.

Oliver Schmidt

Title of the paper

Development and Construction of a Camera-Service, based on “.net”-technology.

Keywords

Object-Detection, Client-Server communication, wireless-LAN, PDA, Firewire-Cameras, „.net“.

Abstract

Inside this report the construction of a system is described, which enabled mobile units to request the positions of some objects, which are moving in an area, from a server.
The mobile units use wireless-LAN to communicate with the server. They can register objects on the server. The positions of the registered objects will be calculate from the current camera-image with image-recognition-algorithms. After that the positions can be request form the mobile units.
The implementation based on „.net“-technology.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung und Motivation

Im Bereich der Robotik spielt der Einsatz von optischen Sensoren und die anschließende Auswertung des Bild-Signals eine große Rolle. Vor allem autonome Roboter brauchen diese Informationen, um sich in unserer Welt orientieren zu können, um so ihre jeweilige Aufgabe erfüllen zu können. Autonome Roboter gibt es inzwischen nicht nur in der Industrie sondern teilweise auch schon in Privathaushalten. Dort sind zum Beispiel automatische Staubsauger oder Rasenmäher unterwegs. In diesem Bereich wird es in den nächsten Jahren sicherlich noch viele neue Entwicklungen geben.

Mein Interesse an autonomen Robotern wurde im Projekt „Robot-Fußball“ geweckt und da ich denke, dass dieses eine Zukunftsweisende Technologie ist, hab ich mich entschlossen in diesem Bereich meine Studienarbeit zu machen.

Abbildung 1.1: Robot-Fußball

Während des Projektes haben wir mit sogenannter On-Board-Sensorik gearbeitet. An dem Roboter haben wir verschiedene Sensoren befestigt, mit denen dieser seine Umwelt(Gegner, Ball, Tore und Spielfeldbegrenzung) wahrnehmen und sich dadurch orientieren konnte. Dieses funktionierte zwar relativ gut. Die On-Board-Sensorik hat aber den Nachteil, dass der Roboter den Ball oder die Tore nicht sehen kann, wenn der Sichtkontakt, z.B. durch den Gegner unterbrochen wird. Solche oder ähnliche Problematiken stellen sich auch bei ernsthaften Anwendungen, bei denen sich autonome Roboter in einem Raum mit unvorhersehbaren Hindernissen zu recht finden müssen.

Eine mögliche Lösung, um dem Roboter eine bessere Orientierung zu ermöglichen, könnte darin liegen, ihm eine Möglichkeit zu verschaffen, von einem externen Observer globale Informationen über seine Umwelt erhalten zu können. Auf diese Weise würde ihm eine Art „Global-Map“ zur Verfügung stehen.

Die Robotersteuerung läuft im Robot-Lab der HAW Hamburg üblicherweise auf einem 6.270-Board, welches vom MIT entwickelte wurde. Zur Kommunikation mit einem externen Observer bietet sich der Einsatz von einem PDA an, da diese heutzutage standardmäßig WLAN- und Bluetooth- Schnittstellen integriert haben. Sowohl das Board, als auch der PDA sind klein und leicht genug, um sie in einem Robot integrieren zu können.

Der Einsatz von einem PDA stellt einen weiteren interessanten Aspekt dieser Studienarbeit dar. Kleine mobile Geräte spielen im Bereich der Informatik eine immer größere Rolle. Sowohl im privaten, als auch im kommerziellen Bereich werden diese Geräte dazu verwendet, an jedem Ort auf seine eigenen Daten und auf das Internet zugreifen zu können. Gekoppelt mit einem GPS-Empfänger können sie auch als Navigationssystem eingesetzt werden. Auch in diesem Bereich wird es in den nächsten Jahren sicherlich viele spannende neue Anwendungen geben.

Diese Studienarbeit beschreibt ein mögliches Design und die Realisierung eines Systems, mit dem der Roboter ständig seine eigene Position auf dem Spielfeld abfragen kann. Kapitel 2 gibt einen Überblick über die Rahmenbedingungen und die eingesetzten Technologien. In Kapitel 3 wird die genaue Aufgabenstellung und die Zielsetzungen beschrieben.. Kapitel 4 zeigt den Entwurf des Systems. Das Kapitel 5 beschreibt die Resultate und gibt einen abschließenden Ausblick für eine eventuelle Weiterentwicklung des Systems.

Kapitel 2

Rahmenbedingungen und Technologie

Hier werden die in der Studienarbeit eingesetzten Technologien beschrieben.

2.1 Hardware

Als PDA setzte ich den im Labor vorhandenen „Pocket PC 2003“ der Firma Hewlett Packard ein. Dieser verfügt sowohl über eine Bluetooth, als auch über eine WLAN-Schnittstelle. Die Kommunikation zwischen dem Pocket-PC und dem Server wird über WLAN erfolgen, da WLAN größere Distanzen zwischen Sender und Empfänger zulässt, sowie eine größere Bandbreite hat. Prinzipiell kann man aber auch ohne weiteres Bluetooth verwenden, da die Protokolle auf höherer Ebene die gleichen sind.

Das Standard-Betriebsystem auf dem Pocket PC ist „MS Windows CE 4.2“. Dieses Betriebssystem ist seit der Version 4.1 in der Lage, .net-Code auszuführen.

Abbildung 2.1 : Pocket PC 2003

Als Server nutze ich einen handelsüblichen Desktop-PC aus dem Labor mit dem Betriebssystem „Windows XP professional“.

Es stehen sowohl USB-Kameras als auch Firewire-Kameras zur Verfügung. Die Firewire-Kameras haben einige Vorteile, wie z.B. eine schnellere Bildübertragung und eine bessere Bildqualität. In unserem Labor steht die Firewire-Kamera „Sony DFW- VL 500“ zur Verfügung.

Abbildung 2.2 : Kamera „Sony DFW-VL 500“

2.2 Software

2.2.1 Programmiersprache / Entwicklungsumgebung

Zur Implementierung der Software, werde ich mit dem .net-Framework von Microsoft arbeiten.

Die .net-Technologie s.a. [Galileo Computing] ist ein noch relativ neues Software-Konzept, welches viele Vorteile der Programmiersprache Java auch für Programmierer anderer Programmiersprachen nutzbar machen soll. Wie Java arbeitet auch ein .net-Programm nicht direkt auf der Hardware, sondern nutzt ebenso eine Art virtuelle Maschine. Diese heißt bei .net Common Language Runtime. Die Spezifikation dieser Laufzeitumgebung ist, was für Microsoft eigentlich untypisch ist offen gelegt und kann daher auch auf andere Plattformen portiert werden. Dazu gibt es das Open-Source Projekt „Mono“. Dieses ermöglicht es das .net-Framework auch für andere Betriebssysteme, wie z.B. Linux, Solaris oder Mac OS nutzen zu können s.a. [Diplomarbeit „Lars Mählmann“].

Der Vorteil der .net-Technologie ist, dass es keine Rolle spielt, in welcher Programmiersprache ein Programm geschrieben ist, da man mit unterschiedlichen Programmiersprachen die gleichen Bibliotheken nutzt und der Code letztendlich auch in den gleichen Byte-Code („Intermediate Language“) übersetzt wird. Dieser wird dann unter Verwendung der Bibliotheken auf der CLR ausgeführt.

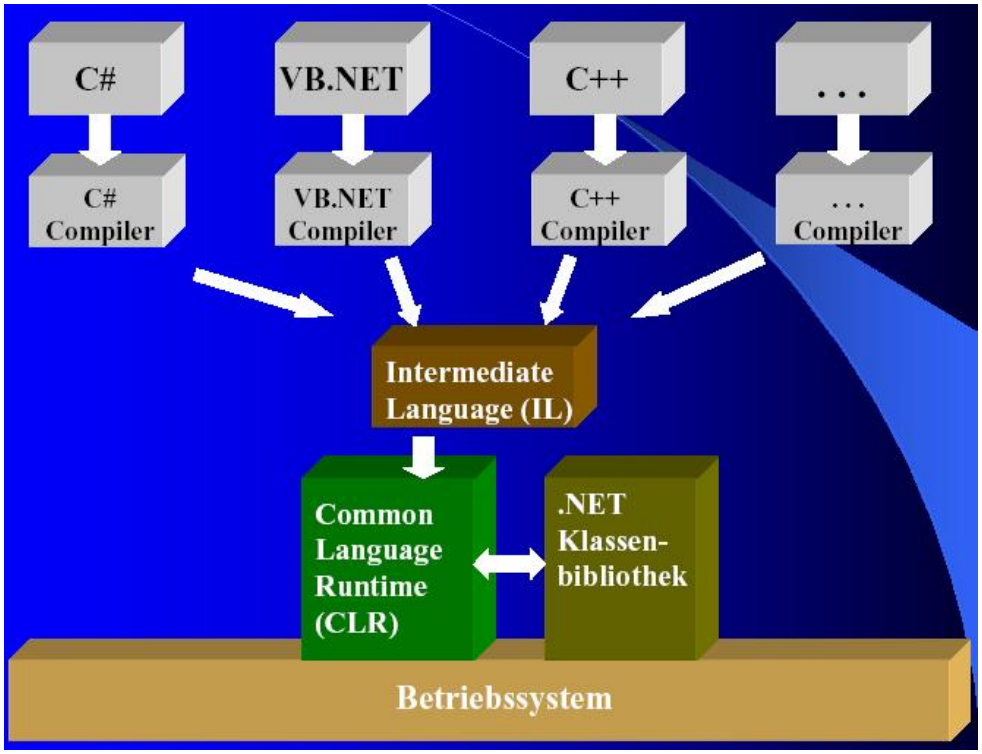

Abbildung 2.3 : Das .net-Konzept

Ich werde mit der Programmiersprache „C#“ arbeiten, da ich bisher hauptsächlich mit Java gearbeitet habe und „C#“ relativ ähnlich ist. Es gibt sogar die von Java bekannte Garbage-Collection.

Für die Software, die später auf dem Pocket-PC ausgeführt werden soll, gibt es das „.net-Compact-Framework“. Dieses stellt eine etwas abgespeckte Version des .net-Frameworks dar, die wesentlichen Bestandteile sind jedoch vorhanden.
Als Entwicklungsumgebung bietet sich das „MS Visual Studio.net“, da dieses extra für .net entwickelt wurde und auch über einen relativ guten GUI-Builder verfügt.

2.2.2 Kamera-Grabber / Bildvearbeitung

Für den Zugriff auf die von der Kamera aufgenommenen Bilder und deren Auswertung nutzte ich das „Camera-Framework“, welches Ilija Revout in seiner Diplomarbeit [Diplomarbeit „Ilija Revout“] entwickelt hat. Dieses Framework umfasst sowohl einen Kamera-Grabber, als auch eine Bildverarbeitungssoftware. Diese ermöglicht es verschiedene Bildfilter zu definieren und diese sogar zu kombinieren. Die Filter werden im XML-Format gespeichert. Als Bildformat erwartet das Framework ein 24 Bit RGB-Format.

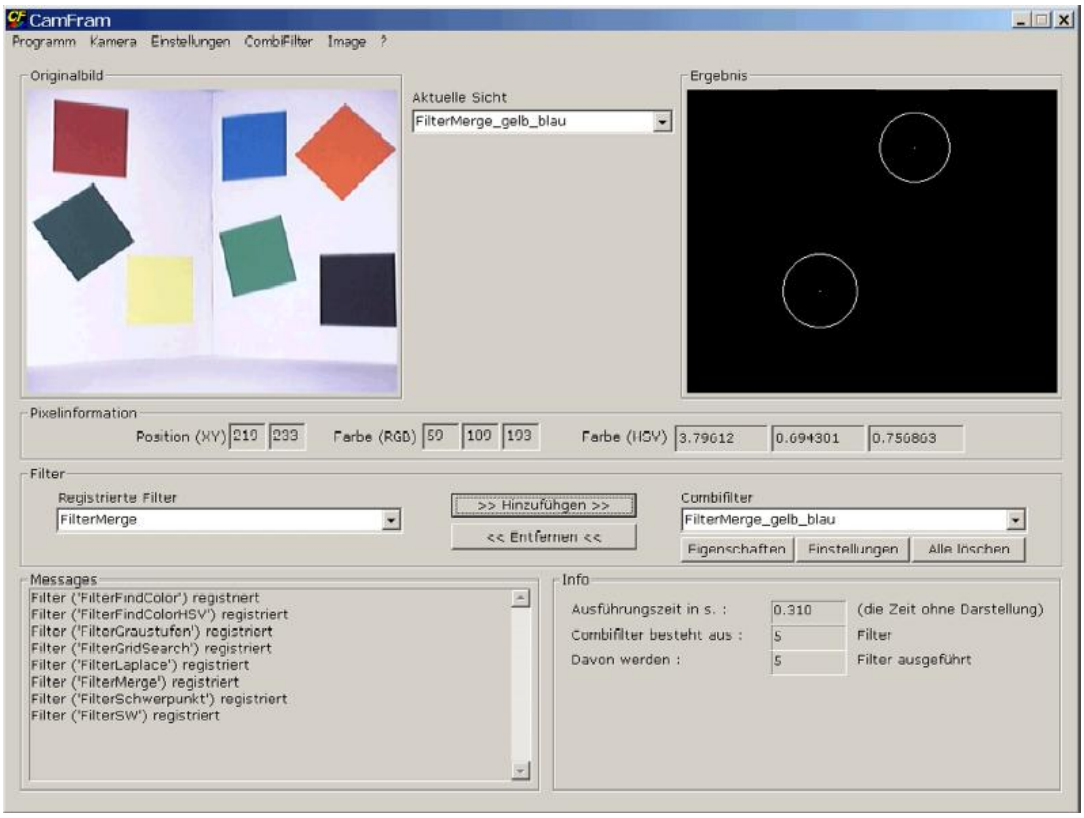

Abbildung 2.4 : Oberfläche des Kamera-Framework
(Ein Beispiel für einen kombinierten Schwerpunkt-Filter)

Kapitel 3

Aufgabe

3.1 Szenario

Das Szenario für diese Aufgabe ist relativ einfach. An einem „Fußball-Roboter“ wird ein PDA befestigt, der sowohl mit der Robotersteuerung, als auch mit einem Kameraserver kommunizieren kann.

Der auf dem Roboter befestigte PDA stellt zu einem beliebigen Zeitpunkt eine Anfrage an den Server, wo sich zu diesem Zeitpunkt ein bestimmtes Objekt auf dem Spielfeld befindet. Das Objekt wird im einfachsten Fall durch seine Farbe beschrieben. Dazu müssten alle interessanten Objekte(z.B. Roboter, Ball, Tore) auf dem Spielfeld durch entsprechende Farbmarkierungen versehen werden. Der Server ermittelt mit Hilfe des Kamera-Frameworks aus dem aktuellen Kamerabild die Position des gesuchten Objektes und sendet diese zurück an den PDA.

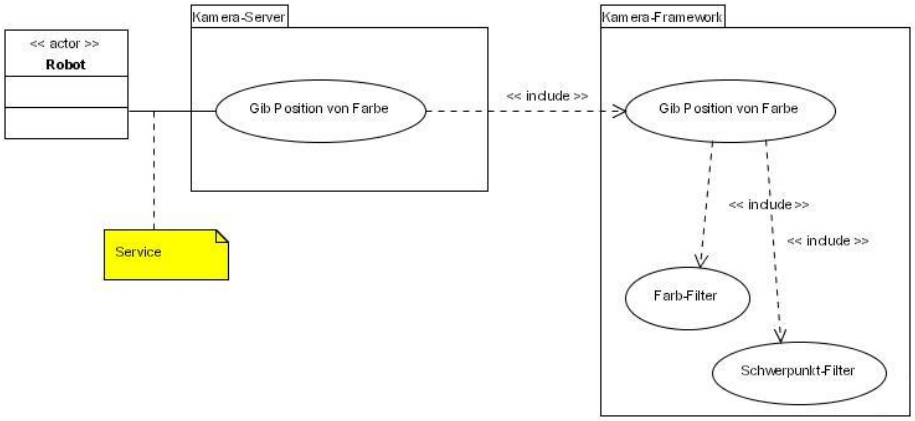

Abbildung 3.1 : Anwendungsfall für den Kamera-Server

Damit die Kamera einen optimalen Blickwinkel erhält, wird diese mittig über dem Tisch aufgehängt. So kann das Kamera-Framework stets alle auf dem Spielfeld befindlichen Objekte lokalisieren.

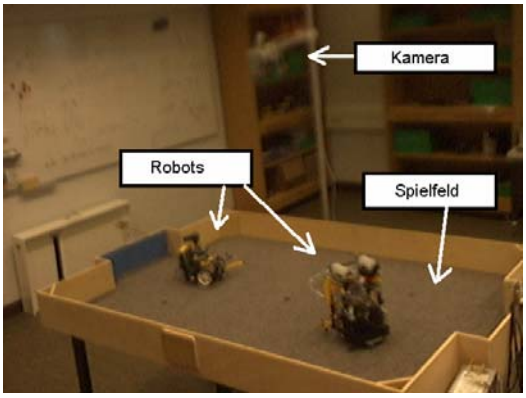

Abbildung 3.2 : Kamera-Position

Auf diese Weise bekommt der Robot eine globale Sicht. Die Vorteile dieser Perspektive wird durch folgenden Abbildungen verdeutlich. Die globale Perspektive wirkt wesentlich übersichtlicher, als die lokale. Diese zusätzlichen Informationen ermöglichen der Robot-Steuerung, eine wesentlich bessere Orientierung und dadurch mehr taktische Möglichkeiten, wie z.B. eine Torverteidigung.

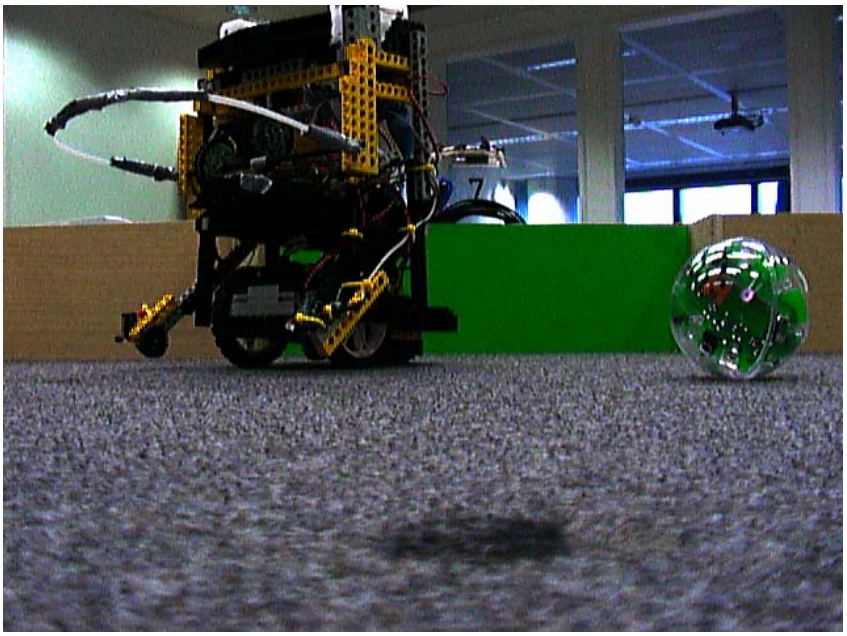

Abbildung 3.3 : On-Board-Perspektive

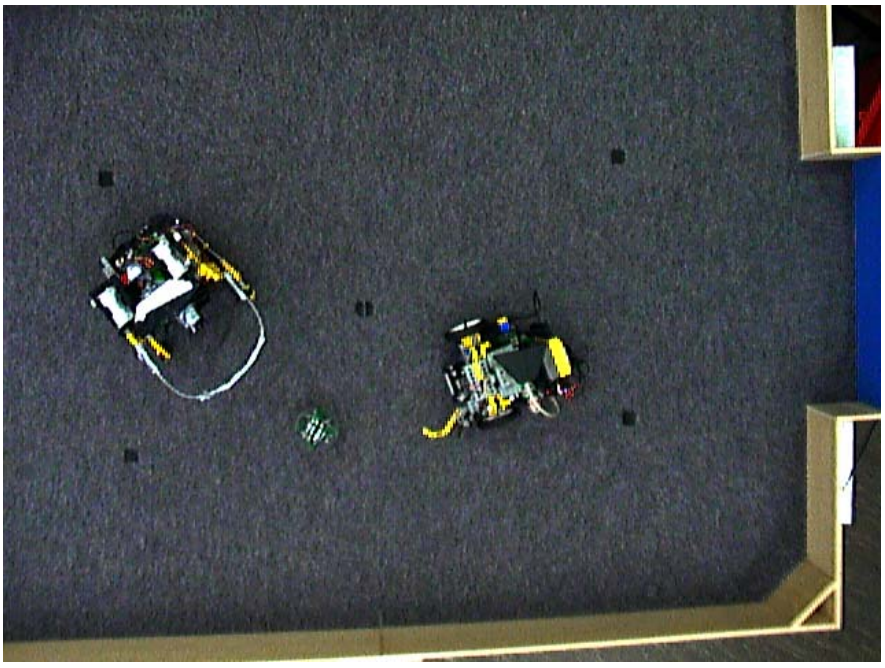

Abbildung 3.4 : Globale Perspektive

3.2 Anforderungen und Zielsetzungen

Damit das System in der Praxis auch Einsatzfähig ist, müssen folgende Anforderungen erfüllt werden.

- → Da bei einem Fußball-Spiel mindestens zwei Roboter zum Einsatz kommen sollte der Server natürlich in der Lage sein, Anfragen von mehreren Clients zu bearbeiten.

- → Beim „Roboter-Fußball“ verändern sich die Positionen der zu beobachtenden Objekte relativ schnell. Daher sollte eine Anfrage in möglichst kürzer Zeit beantwortet werden.

- → Im Roboter-Labor laufen ständig weitere Projekte, die sich mit den Themen PDA, Robotersteuerung und Bildverarbeitung beschäftigen. Daher sollten die einzelnen Komponenten so gestaltet werden, dass diese auch für andere Anwendungen genutzt und weiterentwickelt werden können.

Kapitel 4

Entwurf und Realisierung

Dieses Kapitel beschreibt das Design des Systems. Zunächst wird ein Überblick über die einzelnen Komponenten gegeben. Anschließend wird das Design der Komponenten und deren Kommunikation untereinander beschrieben. Abschließend wird beschrieben, was realisiert wurde.

4.1 Gesamtdesgin

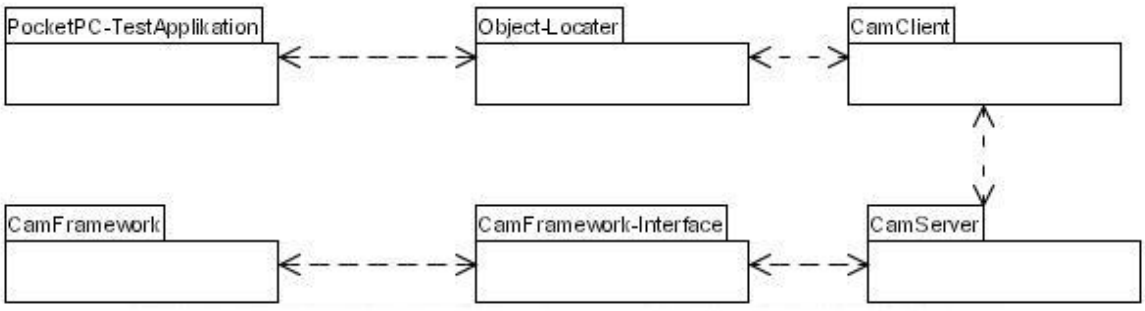

Abbildung 4.1: Die einzelnen Komponenten

Die Aufgabe habe ich in mehrere Teilprogramme unterteilt. Dieses ermöglicht einen einfachen Austausch einzelner Komponenten, bzw. die Nutzung einzelner Komponenten für andere Anwendungen.

Die Aufgaben der einzelnen Teilprogramme werden im folgenden kurz Beschrieben.

PocketPC-TestApplikation
Dieses ist eine Applikation, welche auf dem Pocket-PC ausgeführt wird. Sie soll dazu dienen die Funktionalität des Systems testen zu können. Der Anwender kann hier einen Server angeben und ein oder mehrere Farb-Objekte definieren können, die dann beobachtet werden. In einem späteren Einsatz würde dieses Programm durch die Robotersteuerung ersetzt werden. Diese Applikation simuliert also die Robotersteuerung.

Object-Locater
Hier soll die jeweilige Anwendung Objekte definieren können. Der Objekt-Lokater sendet seine definierten Objekte an den Server, damit beim Kamera-Framework die entsprechenden Filter-Einstellungen vorgenommen werden können. Bei Bedarf kann die Anwendung dann die entsprechenden Objekt-Positionen anfordern.

CamClient
Das Programm „CamClient“ ist eine DLL die auf dem Pocket-PC ausgeführt werden kann. Diese soll eine WLAN-Verbindung mit dem Server herstellen und an diesen Anfragen stellen können.

CamServer
Das Programm „CamServer“ ist eine Applikation, die auf einem PC ausgeführt wird. Es soll auf einem Port auf Anfragen eines Clients warten. Wenn ein Client eine Anfrage stellt, soll diese interpretiert, ausgeführt und beantwortet werden.

CamFramework-Interface
Dieses ist ein System, welches vom CamServer benutzt werden soll, um auf das CamFramework zugreifen zu können. Der Server kann durch dieses Interface gezielt die Anfragen stellen, die er hat, ohne die Implementierung des CamFrameworks zu kennen.

Das Prinzip des gesamten Systems funktioniert folgendermaßen. Eine beliebige Anwendung registriert ein zu beobachtendes Objekt beim „ObjectLocater“. Dieser sendet das Objekt mit Hilfe der Komponente „CamClient“ an den Server. Dieser veranlasst, dass die Filtereinstellungen des „CamFrameworks“ so angepasst werden, dass das entsprechende Objekt nun ständig beobachtet wird.
Die Komponente „CamFrameInterface“ fordert zyklisch die Positionen aller zu beobachtenden Objekte an und speichert diese. Fordert nun ein Client die Position von einem zuvor registrierten Objektes an, wird einfach der zwischengespeicherte Wert zurückgesendet.
Durch das Registrieren und das Zwischenspeichern der Positions-Daten kann die Abfrage wesentlich schneller beantwortet werden, als wenn man bei jeder Anfrage die Filtereinstellungen anpassen würde.

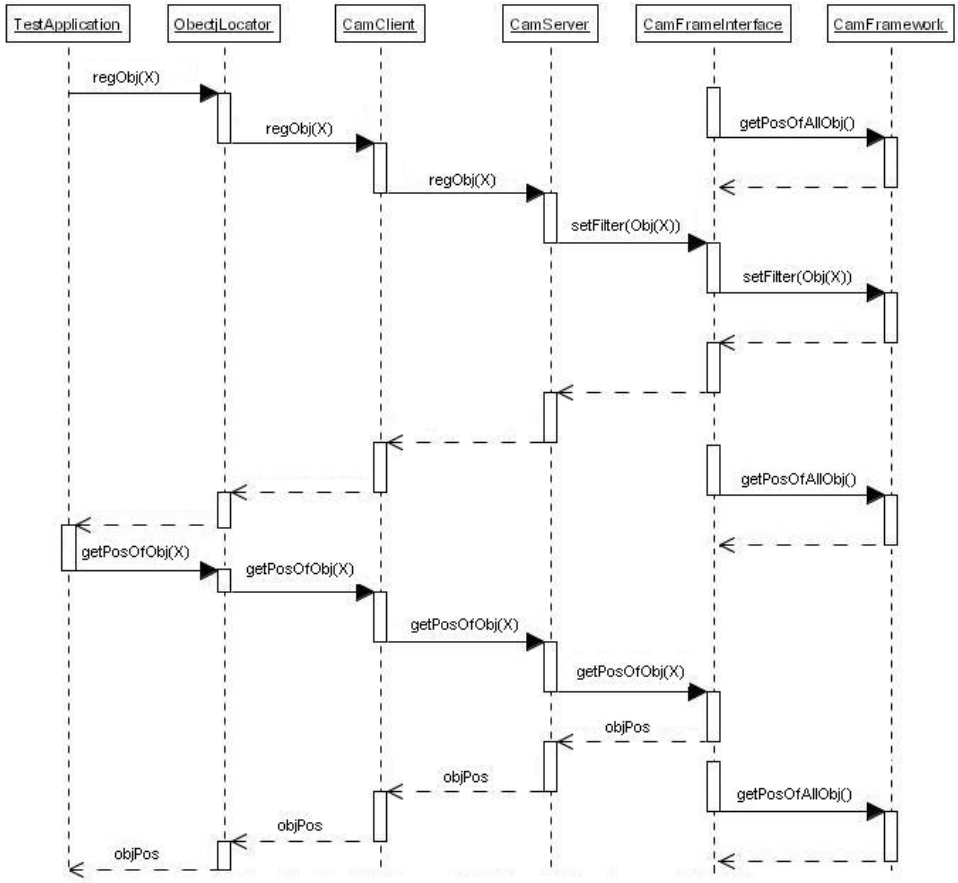

Abbildung 4.2 : Eine Objekt-Registrierung mit anschließender Positionsabfrage

In den folgenden Abschnitten werden die einzelnen Komponenten erläutert.

4.2 Komponente „CamClient“

Diese Komponente setzt auf dem „Compact-Framework“ auf und soll auf dem PDA von der Komponente „ObjectLocater“ dazu verwendet werden, bestimmte Anfragen an einen Server stellen zu können.

Die Komponente „CamClient“ besteht aus zwei Klassen. Eine allgemeine TCP-Client-Klasse, die es ermöglicht, eine Verbindung zu einem Server aufzubauen. Außerdem soll sie einen beliebigen Anforderungs-String an den Server senden und anschließend einen Antwort-String zurückerhalten können. Diese Klasse ist somit Unabhängig vom Verwendungszweck und kann daher auch für andere Anwendungen verwendet werden.
Die Kommunikation erfolgt über eine TCP-Socket-Verbindung.
Die zweite Klasse dieser Komponente dient als Schnittstelle für die Komponente „ObjectLocater“, die diese Komponente benutzt. Diese bietet eine anwendungsspezifische Funktionalität, indem sie die Objekt-Registrierungen und Positions-Anforderungen in Strings verpackt und die Antworten des Servers wieder in Anwendungsspezifische Daten übersetzt. Dieses bezeichnet man auch als „marshalling“. Möchte man die Komponente um zusätzliche Anforderungsmöglichkeiten erweitern, muß man nur der Klasse CamClient weitere „marshall“-Methoden hinzufügen. Außerdem müssen auf der Server-Seite die entsprechenden Erweiterungen vorgenommen werden.

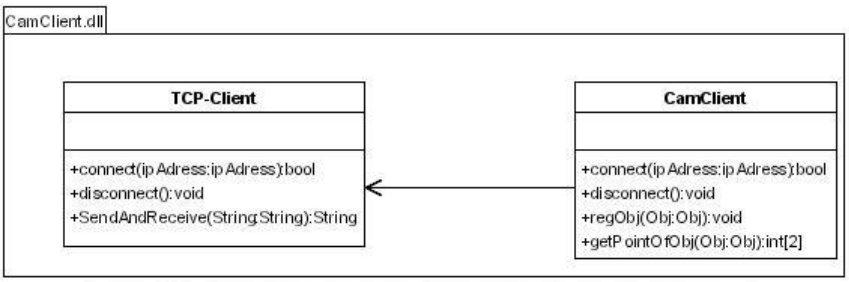

Abbildung 4.3 : Klassendiagramm der Komponente „CamClient“

4.3 Komponente „Object-Locater“

Diese Komponente besteht aus zwei Klassen. Die Klasse „Application-Interface“ stellt Schnittstellen für eine Applikation bereit, die dieses System benutzen möchte. Sie nimmt Registrierungen von zu beobachtenden Objekten, sowie Anforderungen über deren Position entgegen. Die registrierten Objekte werden in der Klasse „RegObjects“ gespeichert. Wird der Klasse ein neues Objekt hinzugefügt, registriert die Klasse dieses Objekt sofort beim Server.

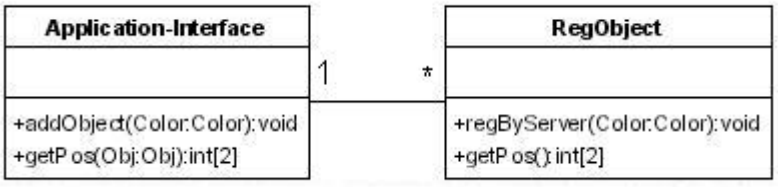

Abbildung 4.4: Klassendiagramm der Komponente „Object-Locater“

4.4 Test-Applikation

Die Test-Applikation soll beispielhaft eine Anwendung simulieren, die vom Server die Positionsdaten zu bestimmten Objekten abfragen möchte.
Dem Anwender wird ein kleines User-Interface bereitgestellt, mit dem die IP-Adresse des Servers, sowie die zu beobachtenden Objekte eingestellt werden können. Diese werden dann bei der Komponente „ObjectLocater“ registriert und deren Position kann dann jederzeit angefordert werden.

4.5 Komponente „CamServer“

Diese Komponente soll Anfragen eines Clients interpretieren, ausführen und beantworten können. Da gleichzeitig Anfragen von mehreren Clients kommen könnten, sollte die Komponente in der Lage sein, durch eine geeignete Multithreading-Architektur mit mehreren Clients gleichzeitig zu kommunizieren. Dazu habe ich die Klasse „TCP-Server“ entworfen. Diese läuft in einem eigenen Thread und hat nur die Aufgabe Anfragen eines Clients entgegen zu nehmen. Sobald eine Anfrage eines Clients vorliegt, startet die Klasse einen weiteren Thread und übergibt die Kommunikation mit dem Client an die Klasse „Communicate“, um anschließend erneut auf eine Client-Anfrage warten zu können.

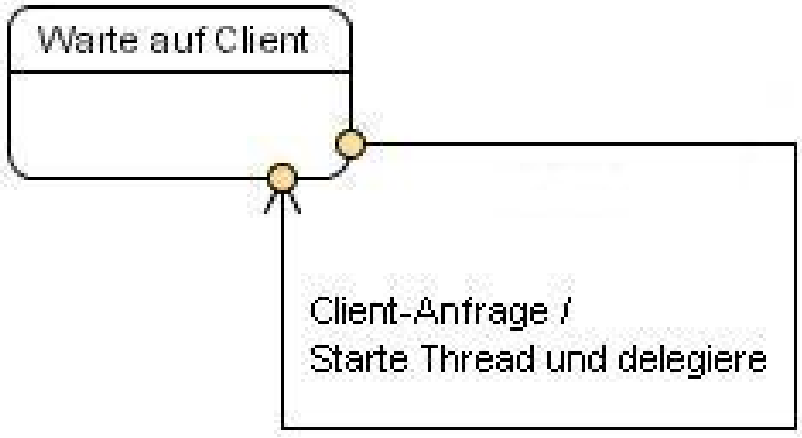

Abbildung 4.5 : TCP-Server wartet auf Clients

In der Klasse „Communicate“ wird dann der Anforderungsstring des Clients empfangen und an die Klasse „Interpreter“ weitergegeben, die dann die Bedeutung der Anforderung interpretieren, bearbeiten und beantworten soll. Dazu benutzt diese Klasse die Komponente „CamFrameInterface“, die den Zugriff auf das Kamera-Framework von Ilija Revout ermöglichen soll. Die Antwort auf die Anfrage wird dann an die Klasse „Communicate“ zurückgegeben. Diese sendet dann die Antwort zurück an den Client.
Wenn man die Komponente um weitere Anforderungsmöglichkeiten erweitern möchte muß man auch hier, wie auf der Client-Seite nur weitere „marschall-Methoden“ in der Klasse „Interpreter“ implementieren.

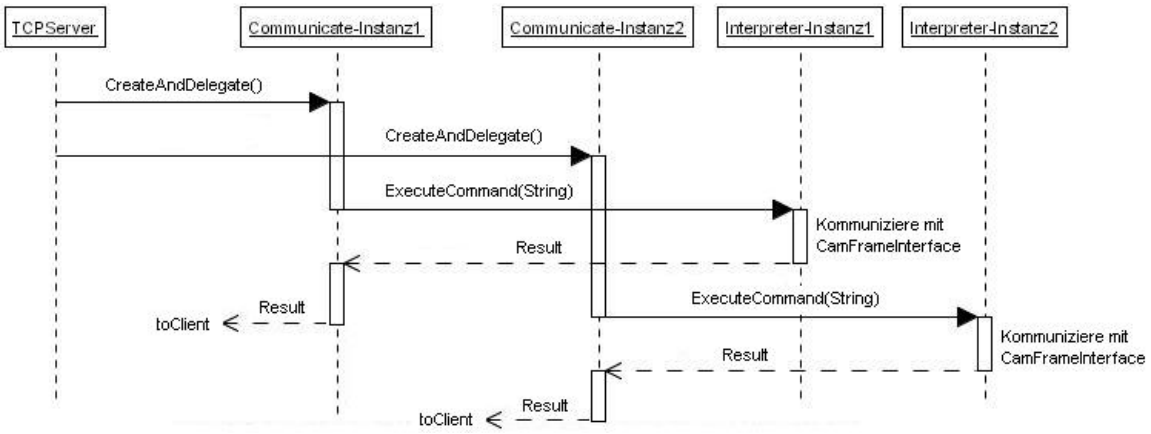

Abbildung 4.6 : Sequenzdiagramm CamServer

Außerdem enthält diese Komponente noch eine Klasse „Mainform“, die zur Kontrolle und zu Testzwecken jede Client-Anfrage auflistet.

4.6 Komponente „CamFrameInterface“

Diese Komponente soll einen einfachen und möglichst schnellen Zugriff auf das Kamera-Framework ermöglichen. Es müssen Filtereinstellungen gesetzt werden können und die Filter-Ergebnisse abgefragt werden können. Das Abfragen der Filterergebnisse sollte zyklisch in einem eigenen Thread erfolgen, damit die aktuell ermittelten Positionsdaten jederzeit abgefragt werden können.

Der Zugriff auf das Kamera-Framework stellt ein Problem dar. Das Kamera-Framework wurde nicht mit dem .net-Framework entwickelt, sondern besteht aus MFC-Libarys. Da beide Technologien von der Firma Microsoft entwickelt wurden und die kompilierten Libaries in beiden Fällen die Datei-Endung „.dll“ haben, könnte man zwar im ersten Moment denken, dass die Integration kein Problem sei. Dieses ist aber leider nicht der Fall, denn beide Technologien haben einen großen prinzipiellen Unterschied. Während MFC-Code direkt auf der Hardware ausgeführt wird, arbeitet .net-Code auf einer Virtullen Maschiene. Daher ist es auch nicht ohne weiteres Möglich, MFC-Libaries in .net-Projekte zu integrieren. Um nun aber trotzdem aus einem .net-Projekt, den „alten“ MFC-Code verwenden zu können gibt es verschiedene Methoden.

Zu den Programmiersprachen, in denen man .net-Code implementieren kann gehört auch c++, welches sich von der Syntax her kaum vom MFC-c++ unterscheidet. Eine Portierung ist aber Aufgrund der unterschiedlichen Bibliotheken bei einem so großem Projekt, wie dem Kamera-Framework eine sehr aufwendige Lösung [Microsoft_1].

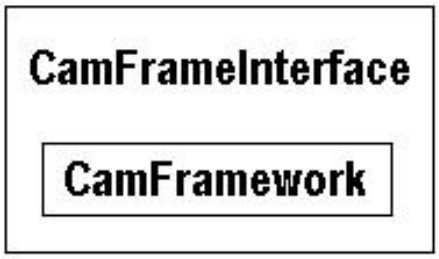

Abbildung 4.7: Lösung „Integration“

Es gibt eine Möglichkeit MFC-Code zu direkt aus einem .net-Projekt zu benutzen. Dafür stellt das .net-Framework die Bibliothek „System.Runtime.InteropServices“ bereit. Dabei sind jedoch nur statische Funktionsaufrufe möglich, so dass wenn man Objekte von Klassen erschaffen möchte, sogenannte „Wrapper-Klassen“ entwickeln muß. Das bedeutet, wenn man ein Objekt x erschaffen möchte, muß man eine Funktion schreiben muß, die dieses tut und den Zeiger speichert, so dass eine weitere Funktion eine Methode der Klasse ausführen kann. Diese Möglichkeit auch noch eine weitere Schwierigkeit. Die Parameter-Übergabe ist nur die einfachsten Datentypen, wie „Integer“, „String“, usw. unproblematisch. Viele andere können nur dann übergeben werden, wenn man eine relativ komplizierte Typ-Konvertierung vornimmt s.a. [Microsoft_2] .

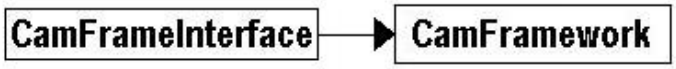

Abbildung 4.8: Lösung „Wrapper-Klasse“

Eine dritte Möglichkeit besteht darin, das .net- und das MFC-Programm als jeweils eigenständigen Prozess ausführen zu lassen und dann über eine Interprozess-Kommunikation miteinander kommunizieren zu lassen. Dabei gibt es auch verschiedene Möglichkeiten. Im einfachsten Fall kann der Austausch über Dateien erfolgen. Dieses ist allerdings relativ langsam und belastet den Rechner. Eleganter wäre eine Kommunikation über „Shared Memory“.

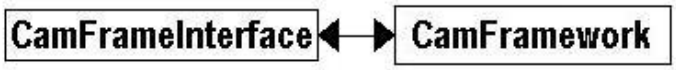

Abbildung 4.9: Interprozesskommonikation

4.7 Realisierung

Aufgrund eines Redesigns und mangelnder Zeit wurde der zuvor beschriebene Entwurf nicht in allen Punkten umgesetzt. Es wurde jedoch ein Prototyp entwickelt, der die grundsätzliche Funktion des Systems demonstrieren kann. Es wurde eine Test-Applikation entwickelt, die dem Benutzer auf dem PDA eine kleine Benutzer-Oberfläche zur Verfügung stellt, mit der er Farb-Objekte definieren kann. Deren Position wird dann zyklisch mit Hilfe der Komponente „CamClient“ beim Server abgefragt. Die zurückgelieferten Positionsdaten werden auf der Oberfläche angezeigt.

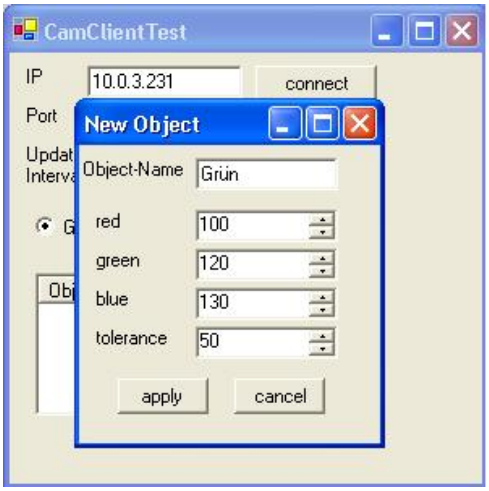

Abbildung 4.10 Dialog „Neues Farb-Objekt definieren“

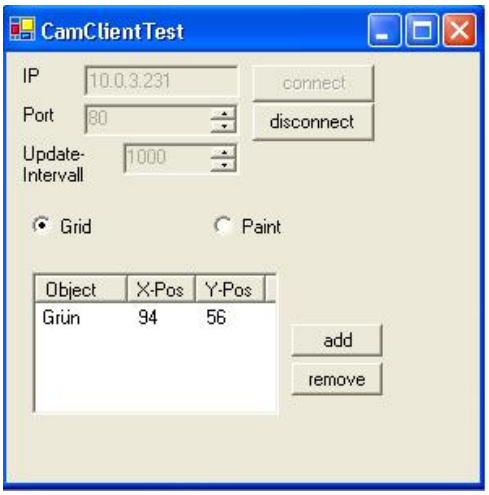

Abbildung 4.11 Dialog „Anzeige der Positionsdaten“

Die Komponente „CamClient“ ist vom Prinzip her, so wie Entwurf beschrieben realisiert und kann eine Anforderung in einen String verpacken, an den Server senden und die Antwort auswerten.

Die Komponente „ObjectDetector“ ist im bisher noch nicht realisiert und somit ist auch das Prinzip der Objekt-Registrierung beim Server noch nicht umgesetzt.

Der Server ist soweit implementiert, dass er auf anfragende Clients wartet und die Kommunikation mit einem solchen in einen zusätzlichen Thread auslagert, um für weitere Clients ansprechbar zu sein. So wie es im Entwurf beschrieben wurde. Die Positionsanforderungen werden an die Komponente „CamFramInterface“ weitergeleitet.

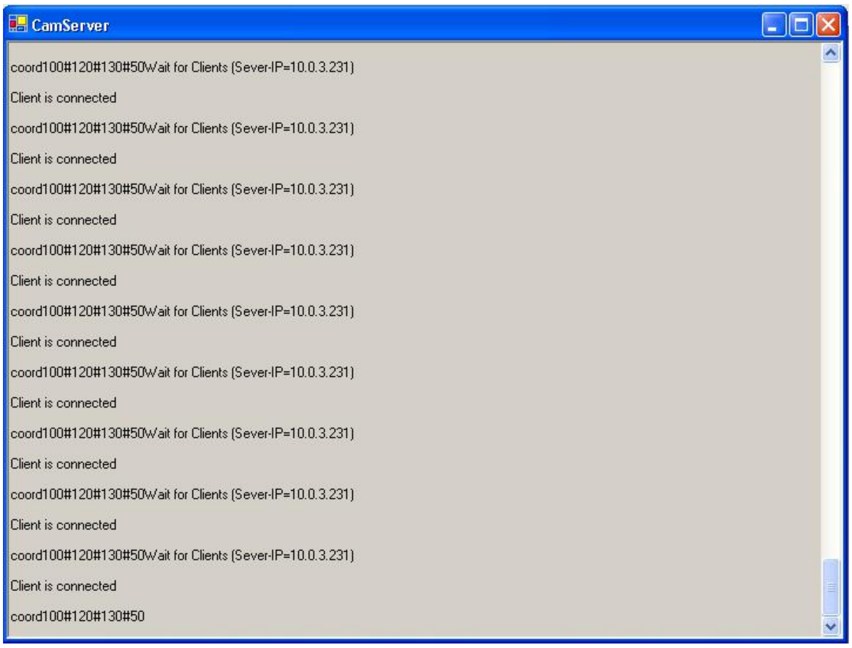

Abbildung 4.12 Kontroll-Fenster des Servers

Die Komponente „CamFramInterface“ ist bisher so realisiert, dass sie das Camera-Framework als einen separaten Prozeß startet. Die Einstellung des Filters erfolgt dadurch, dass die xml-Filterdatei des Camera-Frameworks neu erstellt wird. Das Camera-Framework fragt die Filterdatei ab und schreibt die jeweils ermittelten Positionsdaten in eine weitere Datei. Diese wird dann vom „CamFramInterface“ gelesen und über die Server-Komponente an den anfragenden Client gesendet.

Mit diesem Prototypen werden noch nicht alle in Kapitel 3 aufgestellten Anforderungen zufriedenstellend erfüllt. Die Grundfunktionalität, die darin besteht, dass ein mobiler Client die Positionsdaten eines Objektes von einem Server abfragen kann, ist aber bereits vorhanden. Wie eine zukünftige Weiterentwicklung aussehen kann, wird im folgenden Kapitel beschrieben.

Kapitel 5

Resümee

Dieses Kapitel beschreibt die Ergebnisse dieser Studienarbeit und gibt einen Überblick über den aktuellen Entwicklungsstand. Abschließend werde ich einen Ausblick auf eine zukünftige Weiterentwicklung geben.

5.1 Ergebnisse

In dieser Studienarbeit ist ein System entworfen worden, welches es ermöglicht, dass man von einem PDA aus die Positionen von bestimmten Objekten beobachten lassen kann.

Dabei ist es durch die Multithreading-Architektur im Server und den Einsatz entsprechender Kombinationsfilter im „Camera-Framework“ möglich, dass mehrere Clients gleichzeitig dass System nutzen können.

Eine weitere Anforderung an das System war, dass die Anfragen in möglichst kurzer Zeit beantwortet werden können. Dieses ist durch die Registrierung der zu beobachtenden Objekte und der Pufferung der Messergebnisse auf der Server-Seite erreicht worden. Dabei besteht allerdings die Gefahr, dass die Aktualisierung der Objekt-Positionen bei sehr vielen registrierten Objekten so lange dauert, dass die zurückgelieferten Positions-Daten etwas veraltet sein könnten. Bei dem konkreten Einsatzgebiet „Robot-Fußball“ dürfte das jedoch nicht der Fall sein, da hier die Anzahl der Objekte überschaubar ist.

Die Anforderung an die Erweiterbarkeit des Systems und Wiederverwendungsfähigkeit einzelner Komponenten wird durch eine klare Trennung der Aufgabenbereiche gewährleistet.

5.2 Entwicklungsstand

Wie in Abschnitt 4.7 bereits beschrieben, erfüllt der bisher entwickelte Prototyp noch nicht alle gestellten Anfoderungen. So ist das Konzept der Objekt-Registrierung noch nicht im Prototypen implementiert. Es ist aber gelungen die Grundfunktionalität des Systems zu demonstrieren. So ist es bereits möglich, dass ein Anwender auf dem PDA mit einer kleinen Benutzeroberfläche ein Objekt definieren kann, dessen Position dann zyklisch beim Server abgefragt und nach einer akzeptablen Verzögerungszeit auf der Oberfläche angezeigt wird.

Die parallele Nutzung des Systems durch mehrere Clients ist mit dem jetzigen Prototypen noch nicht möglich. Die Multithreading-Architektur im Server ist zwar bereits implementiert, es fehlen jedoch noch die Objekt-Registrierungen im Server und die damit zusammenhängenden Filtereinstellungen, die gleichzeitig nach den Schwerpunkten mehrerer Farbwerte filtern können. Bisher werden die Filtereinstellungen so vorgenommen, dass jeweils nur nach dem Schwerpunkt einer Farbe gesucht wird.

Außerdem ist der Zugriff auf das „Camera-Framework“ noch nicht zufriedenstellend realisiert. Der Zugriff erfolgt beim jetzigen Stand über ein parallelen Prozess, mit dem über Dateien kommuniziert wird.

5.3 Ausblick

Damit das System in der Praxis einwandfrei funktioniert, müssen im Camera-Framework, bzw. beim Zugriff auf dieses noch einige Erweiterungen vorgenommen werden.

Das „Camera-Framework“ ist zwar grundsätzlich so Modular entwickelt, dass es egal ist mit was für ein Kameratyp angeschlossen ist. Da das Framework intern mit dem RGB24-Format arbeitet, muss das Framework für jedes andere Bildformat um eine Grabber-Klasse erweitert werden, die das jeweilige Bildformat in das RGB24-Format umrechnet. Die Kamera „Sony-DFW 500“ liefert nur ein Format, welches zu RGB24 kompatibel ist und zwar das Y444-Format. Dieses aber leider nur in niedriger Auflösung. Dieses ist bei dieser konkreten Anwendung nicht so problematisch, da der Robot-Fußball-Tisch nicht sehr groß ist und es aufgrund der kleinen Zeitverzögerung bei einem bewegten Spiel sowieso keine exakten Messergebnisse geben kann. Aber trotzdem wäre eine größere Auflösung natürlich wünschenswert.

Ein weiteres technisches Problem besteht mit der Kamera-Konfiguration der „Sony-DFW 500“. Wie bei jeder Kamera kann man hier Einstellungen der Helligkeit, des Kontrastes, usw. vornehmen. Das „Camera-Framework“ stellt zwar im Benutzer-Interface einen Dialog zur Verfügung, mit dem diese Einstellungen vorgenommen werden können, es gibt jedoch keine Möglichkeit diese Einstellungen zu speichern, bzw. zu laden. Sobald man die Kamera neu anschließt, müssen diese Einstellungen manuell konfiguriert werden. Dieses ist erstens umständlich und zweitens besteht die Gefahr, dass die in einer Anwendung eingestellten Farbwerte nicht mehr stimmen. Diese Gefahr besteht zwar Aufgrund von wechselnden Lichtverhältnissen immer, ist aber trotzdem nicht gut. Daher sollte das „Camera-Framework“ um eine solche Möglichkeit erweitert werden.

Wie bereits im Abschnitt „Entwicklungstand“ beschrieben, ist der Zugriff auf das „Kamera-Framework“ noch nicht zufriedenstellend gelöst. In Kaptiel 4.6 habe ich bereits die Problematik, sowie drei Lösungen beschrieben. Die Lösung den Code zu Konvertieren habe ich Aufgrund des Aufwandes bereits ausgeschlossen. Die Lösung den Code über die „Wrapper-Klassen“ zu integrieren erscheint mir die eleganteste und schnellste Lösung zu sein. Meine Versuche eine solche Lösung zu

realisieren scheiterten jedoch daran, dass das Camera-Framework ein Handle von einem Fenster benötigt. Alle meine Versuche dieses in einer MFC-DLL zu erstellen führten zu kryptischen Fehlermeldungen, was auch an meinen mangelnden Kenntnissen der Programmiersprache c++ liegen kann. Aber auch eine intensive Recherche führten auf diesem Wege nicht zum Erfolg. Die Methode mit den zwei eigenständigen Prozessen funktioniert zwar, es sollte jedoch die Art der Interprozesskommunikation verbessert werden.
Eine spannende Weiterentwicklung des Systems könnte darin liegen, den Kamera-Service zu einem Webservice auszubauen. Dabei spielen einige Sicherheits-Aspekte eine Rolle.
Man könnte den Zugriff nur auf autorisierte Benutzer beschränken. So könnte man z.B. bestimmten Benutzern oder Benutzer-Gruppen die Überwachung bestimmter Objekte erlauben, während andere Objekte nicht überwacht werden dürfen. Eine solche Zugriffskontrolle spielt in vielen künftigen Web-Anwendungen eine Rolle, denn ein Dienst der öffentlich im Internet zur Verfügung steht, sollte bei den meisten Anwendungen nicht für alle Benutzern zugänglich sein oder zumindest nicht im vollem Umfang.
Für eine solche Zugangs und Zugriffskontrolle würde sich z.B. das Keberos-Protokoll gut eigenen. Bei diesem Protokoll muß sich ein Benutzer zunächst bei einem Keberos-Server(KDC) anmelden und identifizieren. Wenn dieses erfolgreich geschehen ist stellt dieser Server dem Benutzer ein Sitzungsticket aus, mit dem er für bestimmte Aktionen autorisiert wird. Besonders vorteilhaft ist dieses Verfahren bei verteilten Anwendungen, da sich ein Benutzer nur einmal anmelden muß und mit dem Sitzungsticket Zugang zu mehreren Servern bekommen kann. s.a [Martin Hübner]

Darin besteht auch eine Erweiterungsmöglichkeit. Wenn Benutzer regelmäßig auf einen bestimmten Service zugreifen, wäre es von Vorteil, wenn es Möglichkeit gibt Benutzerkonten anzulegen und zu verwalten. In dem speziellen Fall Kamera-Service könnten zum Beispiel spezielle Farb-Objekte, die ein Benutzer registriert hat gespeichert werden, so dass diese nicht bei jeder Sitzung neu registriert werden müssten.

Literaturverzeichnis

1 Galileo Computing : Kühnel, Andreas; Visual c#. 1.Auflage 2003 @Galileo Press Gmbh, Bonn 2003 -ISBN 3-89842-286-0

2 Diplomarbeit „Lars Mählmann“ : Mählmann, Lars; Mono. Noch nicht veröffentlicht.

3 Diplomarbeit „Ilija Revout“ : Revout, Ilija; Design und Realisierung eines Frameworks für Bildverarbeitung. Online. 2003. URL http://users.informatik.haw-hamburg.de/~kvl/revout/diplomarbeit.pdf. Zugriffsdatum 03.02.2005

4 Microsoft_1: Marquardt, Bernd; Anwendungsmigration. 2003. online. URL http://www.microsoft.com/germany/msdn/library/net/MSDNChatMigration.mspx. Zugriffsdatum 03.02.2005

5 Microsoft_2 : Consuming Unmanaged DLL-Functions; online. URL http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconconsumingunmanageddllfunctions.asp Zugriffsdatum 03.02.2005

6 Martin Hübner : Hübner, Martin; Skript IT-Sicherheit Kaptiel 5.3 Zugriffskontrolle (Autorisation). online URL https://swl.informatik.haw-hamburg.de/home/pub/prof/huebner/IT_Sicherheit/ITS_WS04_Kap5.pdf Zugriffsdatum 03.02.2005