Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Jens Gabrikowski

„User generated content“ am Beispiel einer autonom arbeitenden Webcam

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Jens Gabrikowski

„User generated content“ am Beispiel einer autonom arbeitenden Webcam

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Thomas C. Schmidt
Zweitgutachter : Prof. Dr. rer. nat. Gunter Klemke

Abgegeben am 08. August 2008

Thema

„User generated content“ am Beispiel einer autonom arbeitenden Webcam

Stichworte

Webcam, Webservice, User generated content, RPC, CMS

Kurzzusammenfassung

Ziel dieser Arbeit ist es ein System zu beschreiben und exemplarisch zu implementieren, mit dem es möglich ist, autonom Bildinhalte zu generieren und auf einem Webserver zu veröffentlichen. Das System soll Bilder an Orten ohne kabelgebundenen Internetzugang erfassen und über ein Mobilfunknetz auf einen Webserver übertragen.

Title of the paper

„User generated content“ by an example of an autonomous working webcam

Keywords

Webcam, web service, user generated content, RPC, CMS

Abstract

The goal of this thesis is to describe a system and to implement it by an example, which is able to autonomously generate pictured content and publish it to a web server. The system should capture images at places without wired internet access and transmit them to a web server by using a mobile network.

Danksagung

An erster Stelle geht mein Dank an meinen Betreuer Herrn Prof. Thomas C. Schmidt für seine Unterstützung beim Vorbereiten und Erstellen dieser Arbeit.

In Anbetracht der Tatsache, dass mit dieser Arbeit meine Studienzeit an der HAW-Hamburg endet, möchte ich mich auch bei allen anderen Professoren, Dozenten, Labormitarbeitern und einer Vielzahl von Kommilitonen bedanken, die mir ein interessantes und abwechslungsreiches Studium an der Hochschule bereitet haben.

Mein ganz besonderer Dank jedoch geht an meine ganze Familie, die immer für mich da ist und mir den Rücken frei hält, wenn ich ihre Hilfe brauche.

Meiner Frau Uta danke ich für ihre Unterstützung, ihre Entbehrungen und ihre Geduld während der gesamten Studienzeit und ihren besonderen Einsatz in den letzten Monaten.

Für seine Bereitschaft, sich immer wieder von Mama ablenken zu lassen, wenn er an der Treppe nach oben zum Büro stand und nach seinem Papa zum Spielen gerufen hat, bedanke ich mich bei unserem Sohn Tim.

Oma, ich halte mein Versprechen.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Vorwort

Die Mitglieder des Heli-Club Hamburg e.V. gehen alle ihrem gemeinsamen Hobby, dem Bauen und Fliegen von Modellhubschraubern nach. Für das Betreiben der Modellhubschrauber unterhält der Verein einen Modellflugplatz am Stadtrand von Hamburg, auf dem sich die Mitglieder regelmäßig treffen und ihre Modelle fliegen lassen.

Neben dem Flugplatz betreibt der Verein auch eine Homepage, über die der Vorstand die Mitglieder und andere Interessierte über aktuelle Geschehnisse informieren kann. Auch werden auf der Homepage Beiträge und Fotos veröffentlicht, welche die Mitglieder über ihr Hobby erstellt haben.

Das Hobby macht bei schönem Wetter und in Gemeinschaft mehr Spaß. Daher haben die Mitglieder den Wunsch zu wissen, was so auf dem Flugplatz los ist. Wie ist das Wetter vor Ort? Treffe ich auf andere Mitglieder? Lohnt es sich zum Flugplatz zu fahren? Das sind die Fragen, die sich viele der Mitglieder stellen, bevor sie eine Entscheidung fällen und ihre Autos packen und zum Flugplatz fahren oder nicht.

Aus diesem Bedürfnis ist bei den Mitgliedern die Vision entstanden, ein aktuelles Bild vom Flugplatz auf der Homepage des Vereins veröffentlichen zu können. Ein Eindruck über das Wetter und den Betrieb am Flugplatz würde die Homepage interessanter werden lassen und für viele Mitglieder als Entscheidungshilfe dienen, den Flugplatz aufzusuchen.

Diese Vision soll verwirklicht werden und es soll eine Lösung geschaffen werden, welche regelmäßig Bilder vom Flugplatz auf der Homepage veröffentlicht.

1.2 Über das Thema dieser Arbeit

Unter dem Begriff „User generated content“, auch als UGC abgekürzt, versteht man im Allgemeinen Inhalte, die nicht durch eine Redaktion erzeugt werden, sondern durch die Nutzer bzw. Konsumenten des Mediums.

Der Begriff UGC hat sich im Zusammenhang mit dem Begriff Web 2.0[1] etabliert. Eine der wesentlichen Architekturänderungen am Web 2.0 gegenüber dem klassischen bekannten Web ist der Aufbruch der festen Rollen Inhaltersteller und Inhaltkonsument. Früher wurden die Inhalte durch eine Redaktion gesammelt, verfasst, technisch aufbereitet und dann im Internet publiziert und von Konsument (Internetnutzer) betrachtet. Man kann diese Verbreitung als 1-n Beziehung darstellen. Eine kleine feste Redaktion erstellt Beiträge für eine große Konsumentengruppe. Beim Web 2.0 verschieben sich die Aufgaben. Jeder Konsument ist in der Lage eigene Inhalte (UGC) zu erstellen und zu veröffentlichen. Betrachtet man dieses Verhältnis von Redakteuren zu Konsumenten ergibt sich eine n-n Beziehung.

Jedoch ist lediglich der Begriff UGC mit dem Web 2.0 einhergegangen. Der Sachverhalt nutzergenerierter Inhalte, den er beschreibt, ist eine lange vor dem Internet genutzte Technik. Beispiele hierfür sind z.B. Leserbriefe in Zeitschriften, Einträge in Gästebüchern und Beiträge in offenen Kanälen.

Die derzeit aussagekräftigsten Beispiele für UGC im Internet sind Videoplattformen wie z.B. YouTube, Nachschlagewerke der Wikimedia[2] und Weblogs[3] freier Autoren.

In dieser Arbeit wird vor allem der technische Aspekt betrachtet, der bei der Erzeugung, der Speicherung und der Veröffentlichung von UGC eine Rolle spielt. Der eigentliche Autor des UGC wird eine autonom arbeitende Webcam (siehe Abschnitt 3.1) sein. Der UGC wird im Folgenden als Inhalt bezeichnet.

---

[1] **Web 2.0** fasst Techniken und Elemente der Internetgestaltung zusammen, welche das klassische Modell des Internets von Redaktion und Konsument ablösen. Bekannt wurde der Begriff durch den Artikel "What is Web 2.0" (24) von Tim O’Reilly.

[2] **Wikimedia** ist eine internationale nichtstaatliche Non-Profit-Organisation, die sich der Förderung freien Wissens verschrieben hat. Das mit Abstand bekannteste Wikimedia-Projekt ist die freie Enzyklopädie Wikipedia.

[3] **Weblog,** ein auf einer Internetseite geführtes Tagebuch, welches Einträge zu bestimmten Aspekten oder Themen beinhaltet.

1.3 Vorgaben und Wünsche an die Lösung

Im Rahmen der Vision, aktuelle Bilder des Flugplatzes auf der Homepage zu veröffentlichen, wurden in Gesprächen mit den Mitgliedern und dem Vorstand des Vereins einige Wünsche an das System herausgearbeitet.

Damit die Bilder auf der Homepage überhaupt einen Informationswert haben, dürfen sie nicht zu alt sein. An den Wochenenden sollten somit bis zu zehn Bilder pro Tag aufgenommen und zeitnah auf der Homepage veröffentlicht werden. In der Woche, den Herbst- und Wintermonaten kann das Intervall größer ausfallen und eine geringere Anzahl an Bildern pro Tag veröffentlicht werden.

Einige Vorgaben an die Lösung hängen nicht mit der Vision zusammen, sondern ergeben sich aus den Umweltbedingungen, die man dort am Flugplatz vorfindet. Aufgrund der Abgeschiedenheit zu besiedeltem Gelände ist der Flugplatz weder an dem öffentlichen Stromnetz noch an das Netzwerk irgendeines Kommunikationsanbieter angeschlossen.

Die gesuchte Lösung muss also mit Energie, die z.B. aus einem Solargenerator gewonnen wird, auskommen können.

Die Verbindung zum Internet kann nur funktechnisch hergestellt werden. Beobachtungen und die regelmäßige Nutzung von GSM[4] und GPRS[5] am Flugplatz lassen die Versorgung durch diese Netze dort ausreichend erscheinen.

Der Flugplatz ist im Landschaftsschutzgebiet gelegen. Um den Anblick der Landschaft nicht übermäßig zu strapazieren und potentielle Diebe nicht auf die Installation aufmerksam zu machen, sollte diese möglichst klein und unauffällig am bereits vorhandenen Windsackmast des Flugplatzes angebracht werden. Von dort besteht auch der gewünschte Überblick über die für die Mitglieder interessanten Bereiche des Flugplatzes.

Natürlich verfügt der Verein nur über begrenzte Mittel aus den Mitgliedsbeiträgen. Es ist somit angestrebt eine in der Anschaffung als auch im Betrieb kostengünstige Lösung zu realisieren. Die Montage an dieser

---

[4] **GSM** Global System for Mobile Communications (früher Groupe Spécial Mobile, GSM) ist ein von der CEPT festgelegter Standard für digitale Mobilfunknetze.
[5] **GPRS** General Packet Radio Service ist ein von der ETSI verabschiedeter Standard für paketorientierte Übertragung im Bereich des Mobilfunks.

unzugänglichen Stelle erfordert es, die Installation aus der Ferne zu konfigurieren.

Die Kosten für die Datenübertragung sollen gering gehalten werden. Diese Anforderung impliziert, dass bei volumenabhängigen Tarifen das Datenvolumen gering gehalten werden soll.

Serverseitig ist die Integration in das bestehende CMS[6] Joomla[7] gewünscht. Joomla ist ein in PHP[8] entwickeltes, MySQL[9] gestütztes, Open Source Content Management System. Joomla hat ein API[10], über die es möglich ist, eigene Module und Komponenten einzubinden.

Es ist erwünscht, den aktuellen Inhalt, sowie eine Historie des aktuellen und des vergangenen Tages anzuzeigen. Ältere Inhalte verlieren an Aktualität und Interesse und brauchen somit nicht weiter vorgehalten, archiviert oder angezeigt werden.

1.4 Rechtliche Grundlagen zum Betrieb einer Webcam

Das Größte rechtliche Problem beim Betrieb einer Webcam im öffentlichen oder halböffentlichen Raum sind die evtl. abgebildeten Personen. Jede Person hat nach §22 KunstUrhG das Recht am eigenen Bild und darf nicht uneingeschränkt ohne dessen Einverständnis fotografiert oder anders abgebildet werden.

Unter bestimmten Umständen wird das Betreiben einer Webcam an solchen Orten jedoch möglich, wenn gewisse Randbedingungen eingehalten werden. Nach § 23 Abs. 1 Nr. 2 KunstUrhG (1) Personen als Beiwerk: "Werden Personen als Beiwerk neben einer Landschaft oder anderen Örtlichkeiten abgebildet, ist eine Bildnisveröffentlichung ebenfalls ohne ihre Einwilligung zulässig. Die abgebildete Person darf jedoch nicht der eigentliche Zweck der Aufnahme sein, vielmehr darf sie lediglich als Staffage im Bild sein."

Der Sinn und eigentliche Zweck der Aufnahme muss somit die Landschaft oder der öffentliche Platz sein und auch so dargestellt werden. Die mit abgebildeten Menschen dürfen in dieser Darstellung nur Beiwerk sein. Das heißt für die Darstellung des Flugplatzes, bei der es im Wesentlichen um die

---

[6] **CMS ist ein** Content Management System (siehe Abschnitt 3.7).
[7] **Joomla** ist ein OpenSource CMS (33).
[8] **PHP** steht für eine serverseitige Skriptsprache (30).
[9] **MySQL** ist ein relationales Datenbanksystem (36).
[10] **API (**application programming interface) steht für eine Programmierschnittstelle, welche die Anbindung von zusätzlicher Software an ein bestehendes Softwaresystem ermöglicht.

Wetterverhältnisse und den aktuellen Besucheransturm angeht, besteht keine rechtliche Einschränkung.

1.5 Ziel dieser Arbeit

Ziel dieser Arbeit soll es sein ein System zu erstellen, mit dem am Flugplatz des Vereins autonom Bilder gemacht, ins Internet übertragen und dort durch das CMS auf der Homepage des Vereins angezeigt werden können.

Besonderes Augenmerk soll dabei der Autonomieaspekt erhalten. Das System soll autonom von ortsgebundenen Energieversorgung arbeiten können. Ebenso soll es von kabelgebundener Internetanbindung autonom sein. Ein weiterer Autonomieaspekt ist das selbstständige Arbeiten der Lösung, die über Wochen hinweg keine Benutzereingriffe erfordert. Diese Autonomieaspekte sollen bei der Auswahl der zu verwendenden Technologien im Vordergrund stehen.

Nicht Ziel dieser Arbeit sind die elektrotechnische Umsetzung der Hardware für die Installation am Flugplatz sowie die Energiegewinnung für diese Installation mit einem Solargenerator. Es wird jedoch in Abschnitt 3.3 eine grobe Abschätzung über die Dimensionierung eines Soloargenerators getroffen.

2 Szenario eines Systems zur autonomen Erstellung von Inhalten

2.1 Systemübersicht

Abbildung 2.1 zeigt eine Systemdarstellung, welche das zu lösende Problem darstellt. Aus der Abbildung gehen die einzelnen Systemkomponenten und ihr Zusammenspiel hervor.

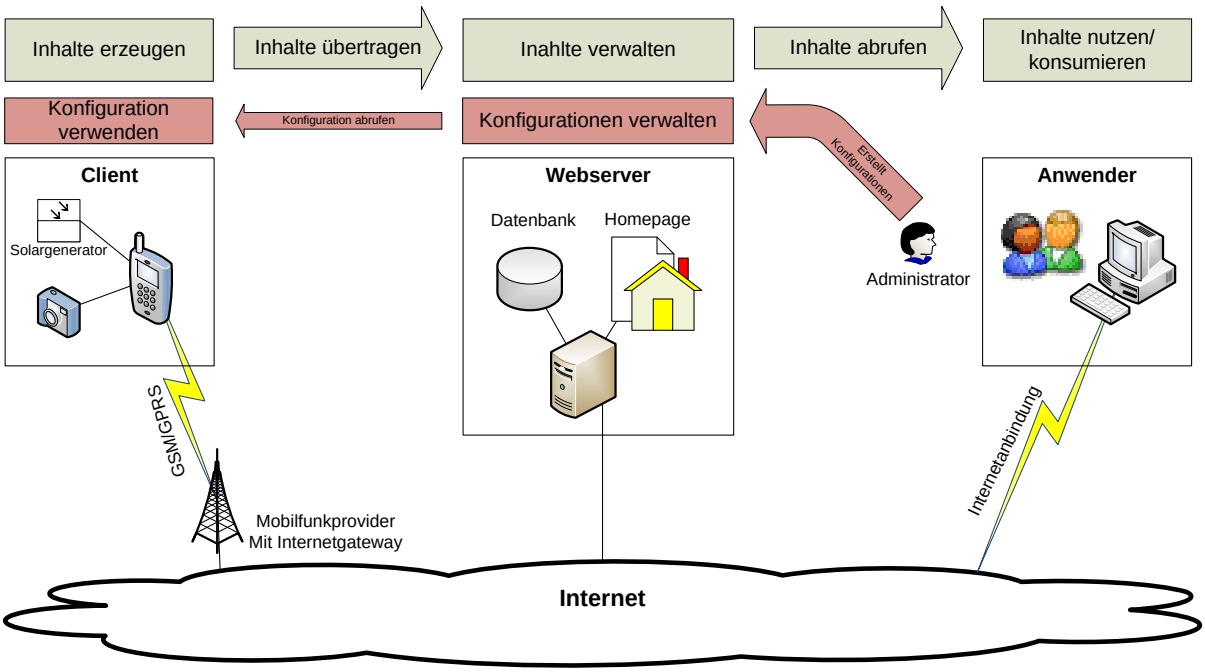

Abbildung 2.1 Systemübersicht

Der Client ist die autonom arbeitende Installation, welche abhängig von der Konfiguration zu bestimmten Zeiten den Inhalt erzeugt. Der Client ist in der Lage Bilder zu machen und andere Daten zu erfassen. Diese erzeugten Inhalte werden dann automatisch vom Client zum Webserver übertragen. In diesem Zusammenhang kann der Client eine neue Konfiguration vom Webserver abrufen.

Der Webserver empfängt den Inhalt vom Client. Er verwaltet den Inhalt im CMS, welches ihn in der Datenbank speichert. Der Webserver hält den Inhalt zum Abrufen über das CMS auf der Homepage bereit. Des Weiteren hält der Webserver die vom Administrator erstellten Konfigurationen für den Client zum Abrufen bereit.

Der Anwender nutzt die Homepage und ruft darüber den Inhalt vom Webserver zum Betrachten ab.

Der Administrator kann Konfigurationen für den Client erstellen. Diese Konfigurationen werden vom Webserver verwaltet und dem Client bereitgestellt.

2.2 Anforderungen an das Gesamtsystem, an Einzelsysteme und Schnittstellen

Aus der Systemübersicht und den Gesichtspunkten aus Abschnitt 0 „

Vorgaben und Wünsche an die Lösung“ lässt sich der gesamte Problemkomplex in kleinere Probleme, Sachverhalte und Anforderungen an das System zerlegen.

2.2.1 Betriebsrobustheit des Clients

Der Client wird an einer schlecht zugänglichen Stelle installiert. Ein Zugriff auf die Hardware oder eine Bedienung der Software ist nur sehr eingeschränkt möglich. Hieraus ergeben sich die aufgeführten Anforderungen an den Client und die Clientsoftware.

2.2.1.1 Autonomer Betrieb

Die Anwendung soll benutzerunabhängig und automatisch ihren Zweck erfüllen. Mögliche auftretende Störungen dürfen die Anwendung nicht in einen Zustand versetzen, in dem sie die Erfüllung ihres Zweckes nicht ohne einen Benutzereingriff wieder aufnehmen kann.

2.2.1.2 Fernkonfigurierbar

Obwohl die Anwendung autonom arbeiten soll, kann es vorkommen, dass ein Ändern der Konfiguration vorgenommen werden soll. Dafür soll das Ändern der Konfiguration aus der Ferne möglich sein.

2.2.1.3 Robustheit gegen Übertragungsstörungen

Kommt es beim Übertragen der Daten vom Client zum Webserver zu Übertragungsstörungen, sollen diese vom System erkannt werden. Es sollen dabei keine Daten verloren gehen.

2.2.2 Energiearmer Betrieb des Clients

Für die Clienthardware steht nur die von einem Solargenerator gelieferte Energiemenge zur Verfügung. Der mittlere akkumulierte Energiebedarf der Clienthardware muss also deutlich unter der zur Verfügung gestellten Energiemenge des Generators liegen.

2.2.3 Erweiterbarkeit

Das beschriebene und zu erstellende System stellt vorerst die erste Iteration einer Entwicklung dar. Weitere Iterationen und Erweiterungen des Systems mit weiteren Sensoren wie z.B. Temperatur, Wind oder Regensensoren sollen möglich sein.

2.2.4 Kleine kompakte, unauffällige Bauform des Clients

Die Clienthardware und der Solargenerator sollen an einem Fahnenmast angebracht werden. Diese Installation soll aus ästhetischen Gründen unauffällig sein und das Landschaftsbild nicht übermäßig verändern.

2.2.5 Kostengünstiger Betrieb des Systems

Um die begrenzten Mittel des Vereins zu schonen, soll der Betrieb des Systems bei den entstehenden Betriebskosten wie Energieversorgung und Funknetzwerkproviderkosten kostengünstig arbeiten.

2.2.6 Anforderungen an die Schnittstelle zwischen Client und Webserver

Da der Client zu bestimmten Uhrzeiten Inhalt erstellt und überträgt, soll die Webserverschnittstelle für den Client jederzeit bereitstehen. Der Client muss nach der Übertragung eine Rückmeldung über den Erfolg oder Misserfolg der Übertragung erhalten um entscheiden zu können, die Daten vorerst lokal zu speichern und später erneut zu übertragen oder die Daten nach erfolgreicher Übertragung lokal zu löschen.

2.2.7 Einzelanwender Kontext

Ein Webserver stellt immer nur die Inhalte eines Clients dar. Es ist nicht vorgesehen Gruppen von Clients zu bilden und in einem gemeinsamen Kontext zu betreiben.

2.2.8 Datenschutz

Bei den erstellten Inhalten handelt es sich nicht um personenbezogene oder anderweitige schützenswerte Daten, von denen nicht jeder Einsicht erlangen darf. Es soll nur sichergestellt werden, dass die auf dem Webserver angezeigten Inhalte von dem richtigen Client stammen und es sich nicht um Fremdeinspielungen handelt.

2.2.9 Integration in die bestehende Infrastruktur

Da der Verein bereits eine Homepage betreibt, ist es notwendig, dass sich die Lösung in die bestehende Infrastruktur der Homepage des Vereins integrieren lässt.

2.2.10 Anforderungen an den Webserver

Die bereits vorhandene Installation auf dem Webserver, insbesondere das dort betriebene CMS Joomla, muss in soweit erweiterbar sein, dass die Schnittstellen zum Client und zum Anwender realisiert werden können. Das Speichern der generierten Inhalte muss möglich sein. Ein Bereich zum Anzeigen des aktuellen Inhaltes sowie ein Bereich zur Anzeige aller Inhalte des aktuellen und vergangenen Tages müssen auf dem Webserver realisierbar sein.

3 Technologieauswahl

Die Technologieauswahl ist genau genommen eine Machbarkeitsstudie mit der herausgefunden werden soll, ob sich für die Teilsysteme adäquate Lösungen und Abbildungen in der Realität finden lassen. Im Rahmen der Studie werden verschiedener Tests durchgeführt, mit denen die Eigenschaften der Technologien beurteilt werden sollen.

3.1 Kameras

Für die Clienthardware wird eine Kamera benötigt. Kameras, die in der Lage sind Daten zu liefern, welche elektronisch weiter verarbeitet werden können, gibt es mittlerweile in allen Ausprägungen. Die geläufigsten Kameratypen sollen hier unter den Aspekten Energiebedarf, Anbindung an Fremdsysteme, Größe und Fernbedienbarkeit erläutert werden. Für den geplanten Einsatz der Kamera zur Erfassung einer groben Übersicht über ein Gelände können der Qualitätsaspekt und die Auflösung der Kamera außer Acht gelassen werden, da alle untersuchten Kameras ausreichend hoch aufgelöste Bilder mit wenigstens 320*240 Bildpunkten liefern.

3.1.1 Digitalkameras

Digitalkameras gibt es in vielen Klassen was Qualitätsansprüche, Größe oder Preis angeht. Auch sind hier Produkte zu finden, die Langzeit- oder Intervallaufnahmen anfertigen. Eine Anbindung ans Internet in der hier geforderten Form ist jedoch unüblich. Fast alle Digitalkameras verfügen über eine USB oder FireWire Schnittstelle. Jedoch ist es immer notwendig, einen Computer mit einer entsprechenden Schnittstelle zu betreiben, um die Bilder weiter zu bearbeiten und ins Internet zu übertragen. Im Allgemeinen wird der Energiebedarf durch einen integrierten Akku gedeckt. Möglichkeiten zum Ein- bzw. Ausschalten fehlen jedoch. Fernauslösen ist bei einigen Modellen über Infrarot oder potentialfreien Kontakt möglich. Die Größe einer kleinen Digitalkamera ist vergleichbar mit der Größe einer Zigarettenschachtel.

Abbildung 3.1 Handelsübliche Digitalkamera

3.1.2 Webcams

Webcam bezeichnet heutzutage im Allgemeinen eine PC-Kamera, die in der Regel über USB oder FireWire an einen PC angeschlossen wird und mit Hilfe einer Software Bilder entweder ins Internet oder zu einem Kommunikationspartner überträgt. Die Energieversorgung findet über die PC Schnittstelle statt. Die Steuerung und das Energieverhalten dieser Kameras sind über die Software zu steuern. Die Größe von solchen Webcams geht hinunter bis zur Größe einer Streichholzschachtel.

Die Geburtsstunde der ersten Webcam „*Trojan Room Coffee Pot Camera“* (2) liegt weit zurück im Jahr 1991. An der University of Cambridge wurde eine Kamera mit einer Framegrabberkarte[11] verbunden um den Füllstand der Kaffeemaschine mit einer Clientsoftware abrufen zu können. Die Parallelen zu diesem Projekt sind groß. Abgesehen von den Autonomieaspekten hatten auch hier die Anwender das Bedürfnis einen für sie weit entfernten Sachverhalt über ein Computernetzwerk einfach sichtbar zu machen.

Abbildung 3.2 Philips CCD Kamera als Bestandteil der ersten Webcam der Welt.

Abbildung 3.3 Standard USB Webcam

[11] **Framegrabberkarte** ist eine Hardwareschnittstelle, welche digitale Einzelbilder oder Videosequenzen von Videosignalen am Eingang erstellt.

3.1.3 IP Kameras

IP Kameras sind die Weiterentwicklung der klassischen Webcam. Die IP Kameras besitzen ein Mikroprozessor und haben eine Netzwerkschnittstelle wie z.B. LAN, WLAN oder ISDN, über die die Verbindung zum Netzwerk hergestellt wird.
Anders als die Webcams ist die Software zur Weiterverarbeitung der Bilder schon in das Betriebssystem der Kamera integriert und die Kamera ist in der Lage, die Bilder ins Netzwerk zu senden oder diese über einen integrierten Webserver zum Abruf bereitzuhalten.

Um den Webserver der IP Kameras kontinuierlich zu betreiben, ist hier ein Energiesparen durch Deaktivieren des Gerätes nur schwer möglich.

Da diese Kameragattung ohne extra Hardware direkt an die Netzwerkinfrastruktur angeschlossen werden kann, eignet sie sich für Überwachungs- und Beobachtungszwecke. Aus diesem Grund sind von dieser Kameragattung auch Modelle in wettergeschützten Gehäusen für die Außenmontage zu bekommen.

Die Größe dieser Kameras liegt im Bereich von der Größe einer Zigarettenschachtel bis zur Größe eines Schuhkartons.

Abbildung 3.4 AXIS 211 Outdoor IP Camera

3.1.4 Mobiltelefone

Die meisten Mobiltelefone sind heutzutage mit einer kleinen Kamera ausgestattet. Diese Kameras sind in der Regel für einfache Schnappschüsse, kleine Kurzvideos und Portraits für Kontaktlisten ausgelegt und gedacht.

Für die gängigen Mobiltelefonbetriebssysteme sind von den Softwareherstellern SDKs[12] zu bekommen, mit denen der Zugriff auf die Hardware und die Steuerung der Kamera möglich sind. Die Tatsache, dass ein Mobiltelefon während des Betriebes auf Energie aus einem Akku angewiesen ist, hat zur Folge, dass alle Hardware- und Softwarekomponenten auf sparsamen Umgang mit der zur Verfügung stehenden Energie ausgelegt sind.

Die Größe der Mobiltelefone ist immer weiter auf ein Maß geschrumpft, welches es anatomisch gerade noch möglich macht, damit zu telefonieren und es zu bedienen. Auch hier kann die Größe einer Zigarettenschachtel als Mittelmaß angesetzt werden.

Abbildung 3.5 HTC P4350Smartphone mit integrierter 2 Megapixel Webcam

Alle untersuchten Aspekte wie Energiebedarf, Fernbedienbarkeit und Kontrolle über die Kamera, Fernwartbarkeit und Größe sprechen für die Verwendung eines Mobiltelefons.

[12] **SDK** steht für Software Development Kit: Bibliotheken und Werkzeuge wie Compiler zur Erstellung eigener Software für das entsprechende Zielsystem.

Das Mobiltelefon ist in den letzten Jahren mehr und mehr zu einer Multimediaplattform mit Internetanbindung herangewachsen und bietet so die optimale Umgebung für die angestrebte Lösung.

Mit einem Mobiltelefon bekommt man Kamera, drahtlose Internetkommunikation und geringen Energiebedarf in einem kleinen Gerät, in dem die Schnittstellen der einzelnen Komponenten bereits aufeinander abgestimmt sind.

Somit ist ein Mobiltelefon mit der passenden Ausstattung die geeignete Clienthardware, die den Anforderungen aus Abschnitt 2.2 genügt.

3.2 Clientbetriebssystem

Mit der Auswahl eines Mobiltelefons als Clienthardware ist die Auswahl des Clients noch nicht abgeschlossen. Mobiltelefone sind heutzutage Hardwareplattformen, auf denen nicht mehr nur die monolithische Mobiltelefonanwendung läuft, sondern Betriebssysteme, welche die Nutzung weitere Software ermöglichen. Bei der Auswahl des Mobiltelefons ist noch das darauf installierte Betriebssystem hinsichtlich folgender Aspekte zu beurteilen:

Zum Übertragen der Daten ins Internet ist es notwendig, dass das Betriebssystem alle notwendigen Funktionen zum Aufbau und Betrieb der Internetverbindung implementiert hat. Zum Übertragen von Daten über diese Verbindung und im Internet müssen wenigstens die Protokolle IP[13] und TCP[14] implementiert sein.

Um auf dem Betriebssystem die zu erstellende Clientsoftware laufen zu lassen, ist es unumgänglich, dass für das Betriebssystem ein SDK verfügbar oder das Betriebssystem in anderer Form erweiterbar ist.

Das Betriebssystem muss einen Zugriff auf die Clienthardware unterstützen oder kapseln und als Schnittstelle bereitstellen um z.B. das Energiemanagement der Hardware zu steuern.

Das erfasste Bild und weitere Daten wie Datum und Uhrzeit der Aufnahme sollen dann zum Webserver übertagen werden. Die Software soll neben Datum und Uhrzeit in der Lage sein zukünftig auch noch weitere Name und Wertpaare zu übertragen.

---

[13] **IP** steht für Internet Protocol (34) und ist die vom Übertragungsmedium unabhängige Vermittlungsschicht.
[14] **TCP** bedeutet Transmission Control Protocol (26) und ist ein verbindungsorientiertes, paketvermitteltes Transportprotokoll zwischen zwei Endpunkten in Computernetzwerken.

Das aufgenommene Bild und die Daten sollen bei Übertragungsfehlern beim nächsten Zeitpunkt, an dem ein Bild erfasst wird, erneut mit übertragen werden. Aus diesem Grund müssen diese durch die Clientsoftware zwischengespeichert werden können.

Das Energieverhalten der Clienthardware soll auch durch die Clientsoftware gesteuert werden. Die Clientsoftware soll alle nicht benötigten Komponenten der Clienthardware deaktivieren oder in Energiesparzustände versetzen und erst bei Bedarf wieder reaktivieren.

Aus einem Artikel der Financial Times Deutschland vom 26.06.2008 (3) geht hervor, dass fast 70% der Mobiltelefone mit dem Betriebssystem Symbian OS betrieben werden. Windows Mobile erreicht einen Marktanteil von 9%. Den Rest teilen sich die Betriebssysteme Android, Mac OS X, Blackberry OS(RIM) und sonstige.

Aufgrund dieser Verteilung werden hier nur die derzeit populärsten Betriebssysteme Symbian OS und Windows Mobile untersucht.

3.2.1 Symbian OS

Nach der Studie des Datenblatts zu Symbian OS V9.5 (4) und Recherchen im Symbian Entwicklernetzwerk (5) geht hervor, dass ein SDK, eine C++[15] API und Plugins für die Eclipse IDE[16] zur Entwicklung eigener Software vorhanden sind. Das Betriebssystem stellt Funktionen zum Zugriff auf eine interne Echtzeituhr zur Verfügung. Das ermöglicht das Erstellen einer Anwendung, die zu bestimmten Zeiten agiert. Es sind alle Protokolle für die Internetverbindung im Betriebssystemkern vorhanden. Auch eine Java VM[17] ist vorhanden, was die Entwicklung der Clientsoftware als Java Anwendung ermöglicht.

3.2.2 Windows Mobile

Im Microsoft Developer Network (6) finden sich alle Informationen über das Windows Mobile 6.0 SDK (7). Dort gibt es übersichtliche und bis runter zum Funktionsaufruf detailierte Informationen über die Funktionen des Betriebssystems. Softwareentwicklung mit dem SDK ist in C++ ATL[18] oder

---

[15] **C++** ist eine ISO standardisierte objektorientierte höhere Programmiersprache.
[16] **Eclipse IDE** steht für Eclipse Integrated development environment und ist ein Anwendungsprogramm zum Entwickeln von Software.
[17] **Java VM** ist eine Anwendung, die eine virtuelle Maschine zum Ausführen von Java Programmen zur Verfügung stellt.
[18] Bei der Active Template Library (**ATL**) handelt es sich um einen Satz vorlagenbasierter C++-Klassen zur Anwendungserstellung.

MFC[19], C#[20] oder Visual Basic[21] möglich. Das SDK integriert sich nahtlos in die Microsoft Entwicklungsumgebung Visual Studio Standard Edition. Aus der Windows Mobile Dokumentation des MSDN Embedded Developer Center (8) geht auch hervor, dass alle Funktionen zur Verwendung einer Echtzeituhr zur Verfügung stehen. Alle notwendigen Protokolle für die Internetkommunikation sind vorhanden. Eine Java VM für Windows Mobile wird von der Esmertec AG (9) angeboten, welche bereits auf zahlreichen Windows Mobile Geräten installiert ist, ermöglicht auch unter diesem Betriebssystem die Verwendung einer Java Anwendung.

3.2.3 Auswahl des Betriebssystems

Beide Betriebssysteme erfüllen alle Anforderungen zur Lösung des Problems. Eine Entscheidung ist daher frei zu treffen. Meine Entscheidung fällt auf das Windows Mobile Betriebssystem. Die große Auswahl an Entwicklungssprachen und die zu erwartende Analogie zu der, mir bereits aus anderen Projekten bekannten Windows Entwicklung, führt zu dieser Entscheidung. Der im Verhältnis geringe Marktanteil wird dabei außer Acht gelassen. Die entstehende Lösung wird kein Massenartikel und ist eine Individuallösung. Auch die mögliche Gefahr eines Außenseiters, dass die Produktlinie in naher Zukunft schon eingestellt werden könnte, ist zu vernachlässigen, da sie wohl immer noch die Lebenszeit des erstellten Systems überdauern wird.

3.3 Solargenerator

Um immer genug Energie für den Betrieb des Clients zur Verfügung zu haben ist es notwendig, den Solargenerator ausreichend zu dimensionieren. Ein Beobachten der prozentualen Akkuanzeige der Hardware des Prototypen hat einen Bedarf von ca. 10% der Akkuladung/Tag ergeben. Bei einem Akku mit einer Kapazität von $C = 1130mAh$ ist das ein Entladung Q_{tag} von:

$$Q_{tag} = C * 10\% = 1130mAh * 10\% = 113mAh$$

Bei einer Nennspannung des Akkus von $U_{nenn} = 3{,}7V$ ergibt sich am Tag ein Energiebedarf E_{tag} von:

---

[19] **MFC** Microsoft Foundation Classes ist eine objektorientierte Klassenbibliothek zur Entwicklung von Anwendungen mit grafischer Benutzerschnittstelle. Anwendungen mit grafischen Benutzeroberflächen für Windows.

[20] **C#** ist eine von Microsoft für das .NET Framework entwickelte objektorientierte höhere Programmiersprache

[21] **Visual Basic** ist eine proprietäre objektorientierte Programmiersprache von Microsoft.

$$E_{tag} = P * t = U_{nenn} * I * t = U_{nenn} * Q_{tag} = 3{,}7V * 113mAh = 418.1mWh \approx 150Ws$$

Dieser Wert ist nur ein grober Richtwert. Er kann aufgrund von Temperatur des Akkus, benötigter Sendeleistung des GSM Modul und schwankender Übertragungszeiten der Daten variieren. Bei der Dimensionierung des Solargenerators ist das zu berücksichtigen und in Langzeittests zu überprüfen. Angenommen wird vorerst ein Faktor $k = 3$. Weiter muss noch der Wirkungsgrad $\eta = 0{,}8$ beim Laden eines Lithium Ionen Akkus berücksichtigt werden.

Daraus ergibt sich die Energie von

$$E_{\frac{Gen}{tag}} = \frac{E_{tag} * k}{\eta} = \frac{150Ws * 3}{0{,}8} = 562{,}5Ws$$

Diese Energie muss der Solargenerator auch an Tagen mit geringer Sonneneinstrahlung liefern können.

Geht man pessimistisch an die Dimensionierung heran, kann man nach Angaben von Solarstrom.de (10) in einer norddeutschen Region in den Monaten März bis Oktober mit einer Sonnenscheindauer von über 100 h/Monaten = 3,3h/Tag rechnen.

Ein Solargenerator muss also in diesen 3,3h einen Strom von

$$I \geq \frac{E}{U * t} = \frac{562{,}5Ws}{3{,}7V * 3{,}3h} = 46\frac{Ws}{Vh} = 46\frac{VAs}{3600Vs} = 12{,}8mA$$

liefern.

Ein möglicher Generator könnte das *SOLARMODUL F. 6V AKKUS* der Firma Schott Solar (11) sein. Dieses Modul liefert bei direkter Sonneneinstrahlung einen Nennstrom von 85mA und hat dabei eine Nennspannung von 8,4V. Die Spannung sollte dem Gerät zum Laden über einen Spannungsregler bereitgestellt werden. Durch die mit 8,4V über doppelt so hohe Spannung soll erreicht werden, dass die Spannung auch bei schwacher Einstrahlung ausreicht um den Akku zu laden.

Prinzipiell gehe ich an dieser Stelle davon aus, dass es mit handelsüblichen Solarmodulen möglich ist die Energieversorgung der Clienthardware permanent sicher zu stellen. Ob die hier getroffenen Annahmen zur Dimensionierung ausreichend sind, wird erst der Langzeittest in der Praxis zeigen.

3.4 Framework

Die Entwicklung von Anwendungen für Windows Mobile ist in verschiedenen Programmiersprachen und Frameworks möglich. Prinzipiell sollte die Realisierung der Clientsoftware in allen Sprachen, die auf das Windows Mobile SDK aufsetzen, möglich sein. Jedoch kann bei geschickter Wahl der richtigen Sprache und des richtigen Frameworks eine Menge Programmieraufwand durch die Verwendung fertiger Bibliotheken vermieden werden.

Die Hauptaufgaben der Clientsoftware, die zu implementieren sind, werden die Steuerung der Hardware, die Benutzerschnittstelle und die Schnittstelle zum Webserver sein. Unter diesen Aspekten wird die Auswahl des Frameworks getroffen.

3.4.1 Win32 native / C++

Für den Zugriff auf die Hardware und die Nutzung der Windows Mobile API Funktionen eignet sich C++ hervorragend. Das Erstellen von Benutzerschnittstellen ist in C++ jedoch aufwändig und umständlich. Im Buch zur Win32API von Charles Petzold (12) ist nachvollziehbar, dass ein einfaches „Hello World“ Fenster schnell in 50 Zeilen Quellcode mündet. Da reines C++ verwendet wird und es für fast jedes softwaretechnische Problem eine C++ Referenzimplementierung gibt, ist die Schnittstelle zum Webserver, egal mit welcher Technologie sie realisiert wird, zu erstellen und zu bedienen.

3.4.2 MFC / C++

MFC ist eine C++ Klassenbibliothek. Die Klassen der MFC Bibliothek implementieren das Adapter Muster (13) zur Win32API und kapseln diese. Mit diesen Klassen ist es wesentlich einfacher Benutzerschnittstellen zu erstellen als es direkt mit der Win32API möglich ist. Der Quellcode wird übersichtlicher, es ist leichter möglich die Ansicht von den Daten und der Anwendungslogik zu trennen. Die Anwendung mit dem „Hello World“ Fenster ist schon mit ca. 10 Zeilen Quellcode möglich. Für den Zugriff auf die Hardware und die Bedienung der Schnittstelle zum Webserver gelten die gleichen Aussagen wie unter 3.4.1.

3.4.3 C# / .Net

Bevor Microsoft im Jahr 2000 das .Net Framework vorstellte, verwendete man zur Windows Programmierung die Windows API, die MFC Klassen oder Frameworks von anderen Anbietern. Alle dieses Techniken und Frameworks hatten das gemeinsame Problem, dass sie über die Jahre gewachsen waren. Sie hatten neue Anforderungen an moderne Anwendungssoftware zu erfüllen,

für die sie ursprünglich gar nicht gemacht waren. Ein Großteil der Erweiterungen bestand darin, Dinge möglich zu machen, die mit der Technologie eigentlich nicht möglich waren. Mit dem .Net Framework hat Microsoft eine konsequent objektorientierte Klassenbibliothek geschaffen, die aktuelle Probleme löst, erweiterbar ist und dadurch zukunftssicher sein soll. Parallel dazu wurde die Programmiersprache C# entwickelt, welche optimal an die Möglichkeiten des .Net Frameworks angepasst wurde. Untersucht man das .Net Framework und C# hinsichtlich der für die Clientanwendung geforderten Aspekte, so stellt man fest, dass sich derzeit kein Framework besser eignet um auf Windows Mobile Benutzerschnittstellen zu realisieren. Der Zugriff auf die Hardware wird für viele Operationen schon durch das .Net Framework gekapselt. Für besondere Anforderungen ist der direkte Zugriff auf die Windows Mobile API möglich. Für die Kommunikation mit Fremdsystemen sind bereits viele Techniken im .Net Framework enthalten oder stehen als externe Bibliotheken zur Verfügung.

Für die Verwendung des .Net Frameworks unter dem Betriebssystem Windows Mobile gibt es die im funktionsumfang abgespeckte Version .Net Compact Framework.

3.4.4 Java

Java eignet sich wohl gleichermaßen wie MFC oder .NET für die Erstellung der Benutzerschnittstelle. Da Java den Anspruch erhebt Plattform unabhängig zu sein, wird von den Compilern nur ein einheitlicher Bytecode erzeugt, der dann in einer virtuellen Maschine auf dem Zielsystem laufen kann. Mit der gewollten Plattformunabhängigkeit auf der einen Seite muss man mit dem kleinsten gemeinsamen Nenner, was die Anforderungen und Möglichkeiten der virtuellen Maschine angeht, auf der anderen Seite leben. Somit ist es mit Java problematisch, die Grenzen der virtuellen Maschine zu überschreiten und Betriebssystem- oder Gerätekomponenten anzusprechen. Ein Steuern der Clienthardware über die Grenzen der standardmäßig vorgesehenen Operationen wäre nur über selbst zu erstellende Bibliotheken für die Virtuell Maschine möglich. Aus diesem Grund ist Java nicht das passende Framework für die Clientanwendung.

Aus der Analyse der Anwendungsframeworks geht hervor, dass die native Nutzung der Windows API mit C++ hinsichtlich der Implementierung der Benutzerschnittstelle umständlich und nicht mehr zeitgemäß ist. Die Fähigkeiten von Java sind den Anforderungen zum Erstellen der Benutzerschnittstelle zwar gewachsen, jedoch sind die Grenzen der virtuellen

Maschine zu eng, um gezielt und ungehindert auf die Clienthardware zuzugreifen. Die MFC Bibliothek und das .Net Framework eignen sich gleichermaßen für die Entwicklung der Benutzerschnittstelle, den Zugriff auf die Clienthardware und die Kommunikation mit dem Webserver. Die zukunftsorientierte und aktuelle Klassenbibliothek des .Net Frameworks ziehe ich für die Erstellung der Clientanwendung der MFC Klassenbibliothek vor.

3.5 Webserverplattform

Die Netcraft Ltd. (14) hat für Juni 2008 die folgende Verteilung der Marktanteil der Webserver im Internet ermittelt.

Developer	May 2008	Percent	June 2008	Percent	Change
Apache	83,746,837	49.73%	84,647,780	49.12%	-0.61
Microsoft	58,991,106	35.03%	60,995,528	35.39%	0.36
Google	10,127,956	6.01%	10,468,720	6.07%	0.06
lighttpd	1,523,148	0.90%	1,532,952	0.89%	-0.01
Sun	545,651	0.32%	550,723	0.32%	-0.00

Tabelle 1 Marktanteile der verschiedenen Webserver

Auch wenn die Serverplattform durch die Randbedingung bereits vorgegeben ist, sollen die beiden Marktführer hier einmal kurz diskutiert werden.

3.5.1 Apache, PHP und MySQL

Die vorgegebene Serverplattform ist eine Kombination aus einem Apache Webserver, dem PHP Interpreter und einer MySQL Datenbank. Dies ist eine auch bei vielen Internetprovidern gängige Installation.

Prinzipiell sind mit dieser Kombination alle Anforderungen an den Webserver erfüllt. Es ist möglich in PHP die Schnittstellen für die Clientanwendung zu erstellen. Es ist möglich die Inhalte in der MySQL Datenbank zu speichern und die Inhalte über den Apache Webserver dem Anwender zur Verfügung zu stellen. Die endgültige Diskussion ist damit aber nicht abgeschlossen, da weder die Schnittstellen noch die Datenspeicherung autark auf dem Webserver laufen sollen, sondern Bestandteil des bereits für die Homepage verwendeten CMS

werden sollen. Somit wird die Diskussion um die Webserverplattform abschließend im Abschnitt 3.7 behandelt.

3.5.2 Microsoft Internet Information Server, ASP.NET, MS SQL

Eine weitere nur der Vollständigkeit halber hier erwähnte gängige Webserverplattform ist eine Kombination aus Produkten der Microsoft Familie: Microsoft Internet Information Server als Webserver, ASP.Net[22] als Skriptsprache und Microsoft SQL Server als Datenbankserver. Auch diese Kombination würde alle Anforderungen an den Webserver erfüllen. Ein an dieser Kombination interessanter Aspekt wäre die Tatsache viele bereits fertige Schnittstellen wie z.B. XML-Webservices zwischen der mit dem .Net Framework erstellten Clientanwendung und dem Server seitigem APS.Net Framework vorzufinden und verwenden zu können. Leider ist diese Kombination bei Internetdienstleistern weniger geläufig als die im Abschnitt 3.5.1 genannte.

3.6 Model-View-Controller Paradigma

An dieser Stelle wird auf das Model-View-Controller (MVC) Paradigma eingegangen. Obwohl das MVC-Paradigma erst im Kapitel 4 „Design zur Realisierung“ eine größere Rolle spielt, wird bereits an dieser Stelle darauf eingegangen, da das MVC-Paradigma hilft zu verstehen wie CMS aufgebaut sind.

„MVC steht für Model-View-Controller und ist ein Architekturmuster, das aus der Programmiersprache Smalltalk stammt. Der Grundgedanke der MVC-Architektur ist, dass die eigentlichen Daten von ihrer Präsentation getrennt werden. So kann ein Datum (Model) mehrere Views besitzen.“ (15)

[22] **ASP.Net** steht für Active Server Pages .Net und ist eine Technologie, bei der mit Hilfe von Anwendungen, die auf dem .NET Framework basieren, Webseiten auf dem Server erstellt werden.

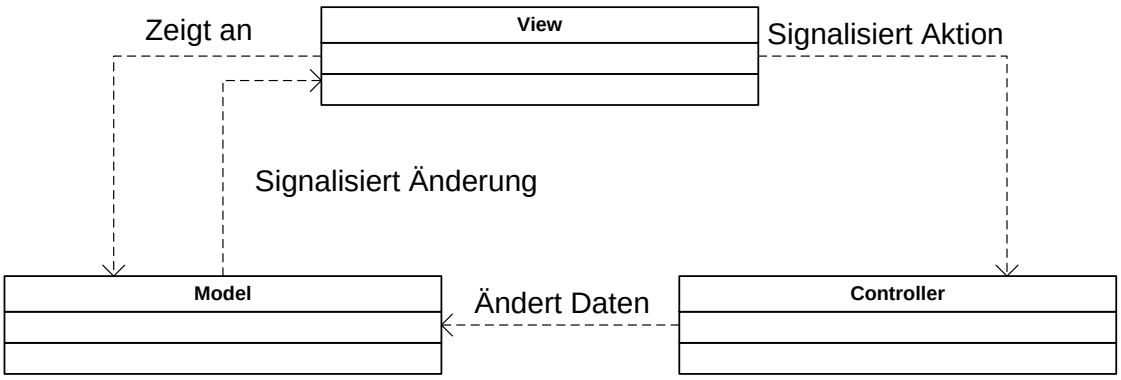

Abbildung 3.6 Zusammenhang zwischen Model, View und Controller

Das MVC Architekturmuster verwendet für die Repräsentation der drei Bestandteile Model, View und Controller drei Objekte.

Das Model-Objekt kapselt und speichert die eigentlichen Daten. Es braucht keine Referenzen zum View- und Controller-Objekt. Jedoch kann eine Schnittstelle definiert sein, mit der das Model-Objekt die View- oder Controller-Objekte darüber informieren kann, dass sich die Daten geändert haben.

Ein View-Objekt hat eine Referenz auf ein Model-Objekt. Das View-Objekt ist für die Präsentation der Daten aus dem Model-Objekt verantwortlich. Es kann dazu auf die Daten des Model-Objektes lesend zugreifen. Häufig gibt es für ein Model-Objekt mehrere View-Objekte, welche die Daten in verschiedenen Darstellungen präsentieren können. Ein View-Objekt kann auch eine Referenz auf ein Controller-Objekt haben. Über diese Referenz können dem Controller-Objekt z.B. Benutzerinteraktionen signalisiert werden, welche dann bestimmte Aktionen im Controller-Objekt auslösen können.

Das Controller-Objekt kapselt die Anwendungslogik und den schreibenden Zugriff auf das Model-Objekt. Aktionen die vom View-Objekt oder anderen Teilen der Anwendung beim Controller-Objekt signalisiert werden, werden vom Controller-Objekt verarbeitet und über eine Referenz auf das Model-Objekt in den Daten gespeichert.

3.7 Content Management Systeme

„Ein Content Management System (CMS) ist eine Software, die eine gemeinschaftliche Arbeit an verschiedenen Inhalten ermöglicht und organisiert. Unter dem Begriff Inhalt ist dabei eine in sich geschlossene Einheit zu

verstehen. Das kann ein Text oder ein Bild sein, aber auch andere Medientypen (Audio, Video etc.) sind denkbar.“ (16)

Auch in CMS findet das in Abschnitt 3.6 vorgestellte MVC-Paradigma Anwendung. Die redaktionellen Inhalte werden in der Regel nur mit einfachen Formatierungen versehen, welche keinen direkten Einfluss auf das Layout haben. Die so gespeicherten Inhalte verhalten sich analog zum Model des MVC-Paradigma. Die Darstellung und Aufbereitung der Inhalte nach bestimmten Layout-Vorgaben übernimmt dann das CMS. Diese Aufgabe des CMS entspricht dem View des MVC-Paradigma. Auch hier werden die Inhalte häufig durch verschiedene Views aufbereitet um sie z.B. zum Drucken anders zu formatieren als für die Bildschirmausgabe. Gibt es auch noch die Möglichkeit die Inhalte über einen View zu verändern, dann übernimmt die Anwendungslogik des CMS an der Stelle auch noch die Rolle des Controllers des MVC-Paradigmas.

Um das beschriebene System zu realisieren ist eine Erweiterbarkeit des CMS notwendig. Zudem sind Methoden erforderlich, die den erzeugten Inhalt in das CMS importieren (Schnittstelle Client /Webserver). Der übertragene und importierte Inhalt muss im CMS gespeichert werden können. Und letztendlich muss der Inhalt dem Anwender in aufbereiteter Form wieder zur Ansicht zur Verfügung gestellt werden.

Da der Verein bereits über eine Internetseite mit einem CMS verfügt, ist an dieser Stelle zu untersuchen, ob das dort verwendete CMS „Joomla“ alle notwendigen Eigenschaften für die Realisierung des Systems aufweist.

3.7.1 Joomla

Joomla verfügt über eine PHP API, welche die Erweiterung des CMSs durch eigene Software komfortabel unterstützt. Joomla nutzt zur Speicherung der Inhalte das relationale Datenbanksystem MySQL. Über die Joomla API oder direkt über PHP kann auf dieses Datenbanksystem zugegriffen und der Inhalt gespeichert, verwaltet und abgerufen werden.

3.7.1.1 Modul

Ein Modul ist eine Erweiterung des CMS Joomla auf Basis dessen APIs. Module werden auf der Homepage in Menus und Navigationsflächen eingeblendet. Ein Modul übernimmt auf jeden Fall immer die Rolle eines Views nach dem MVC-Paradigma, da es wenigstens sich selbst zur Anzeige bringt. Häufig erhalten Module aber mehr Fähigkeiten, als nur sich selbst statisch darzustellen. Dann stellen die Module z.B. die Daten eines Models nach dem

MVC-Paradigma dar oder nehmen auch noch Aufgaben des Controllers wahr und modifizieren das Model nach Benutzerinteraktionen.

Wann ein Modul auf der Homepage zur Anzeige kommt, entscheidet das CMS und ruft dann den Programmcode des Moduls auf, damit es die eigene Darstellung erzeugt. Es können auf jeder Homepageseite beliebig viele Module angezeigt werden. Das Modul hat über die API Zugriff auf Zustände, Eigenschaften, Methoden und die Datenbank des CMSs. Es ist nicht möglich ein Modul direkt aufzurufen, es kann nur im Rahmen der ganzen Seite angezeigt werden.

3.7.1.2 Komponente

In Hinblick auf die Rollen nach dem MVC-Paradigma ist eine Komponente mit einem Modul vergleichbar. Eine Komponente hat ein oder mehrere Views, kann aber auch Model und Controller enthalten.

Auch wann eine Komponente zur Anzeige kommt, wird vom CMS entschieden. Anders als die Module wird immer nur eine aktive Komponente vom CMS aufgerufen und im zentralen Bereich der Homepageseite zur Anzeige gebracht.

Üblicherweise verwendet man die Module zur Anzeige bestimmter Sachverhalte und für triviale Benutzerinteraktionen. Komplexe Inhalte oder Benutzerinteraktionen werden über Komponenten realisiert und stehen dann im Mittelpunkt der Darstellung.

3.7.1.3 Direkter Zugriff

Gemeinsam mit einem Modul oder einer Komponente können eigene Zugriffsmöglichkeiten geschaffen werden, welche direkt aus einem Browser oder von einem anderen Client aufgerufen werden können. Diese Zugriffsmöglichkeiten können die API des CMS mit einbinden und haben den gleichen Zugriff auf das CMS wie Module und Komponenten. Anders ist, dass der Aufruf direkt geschieht und nicht das CMS darüber entscheidet, was wann zur Anzeige kommt. Von dieser Möglichkeit macht man z.B. bei der Bereitstellung eines Webservices gebrauch, bei dem nicht die Homepage angezeigt werden, sondern nur die Kommunikation für den Zugriff abgewickelt werden soll.

3.7.1.4 Backend

Das CMS verfügt über einen Backend Bereich. Das Backend ist der Bereich eines CMSs, den nicht der Internetnutzer verwendet, wenn er die Homepage besucht, sondern von Administratoren und Redakteuren, die die Homepage verwalten und die Inhalte erstellen und pflegen. Für Module und Komponenten

können Eigenschaften definiert werden, welche dann im Backend konfiguriert werden können. Auch für Module und Komponenten übernimmt das Backend alle drei Rollen des MVC-Paradigmas. Es werden Daten zur Anzeige gebracht und können durch Benutzerinteraktion verändert werden.

Abschließend kann man feststellend behaupten, dass das CMS Joomla den Anforderungen zur Realisierung des beschriebenen Systems in den Punkten Anzeigen und Speichern der Inhalte sowie Realisierung der Schnittstellen gewachsen ist. Gemeinsam mit dieser Auswahl bestätigt sich auch die Auswahl der Webserverplattform aus Abschnitt 3.5.

3.8 Datentransport über die Schnittstelle zwischen Client und Webserver

Die Inhalte, welche von der Clientanwendung erstellt werden, müssen über das Internet zum Webserver übertragen werden. Während der Webserver in über eine Standleitung ohne zusätzliche Volumenkosten für den Datenverkehr an das Internet angebunden ist, baut die Clientanwendung eine Datenverbindung zum Internet auf, bei der die Kostenberechnung anhand des übertragenen Datenvolumens stattfindet. Somit sollten die zu übertragenden Datenmengen so gering wie möglich ausfallen.

Zu übertragen sind Inhalte in Form von Binärdaten, Text und skalaren Werten.

Die folgenden Übertragungsmöglichkeiten werden unter den Aspekten des zu übertragenden Datenvolumens und der Möglichkeit die Quelle der Daten sicher zu identifizieren untersucht.

3.8.1 FTP

Das File Transfer Protocol (FTP) (17) wird zum Übertragen von Dateien zwischen zwei Computern verwendet. Das Übertragen der Inhalte über FTP erfordert einen konfigurierten FTP-Zugang auf dem Webserver. Dieses kann bei fast jedem Webserver und Provider als gegeben angenommen werden. Der Client könnte seine Inhalte jederzeit beim FTP-Server des Webservers abliefern. Der FTP-Server würde die Authentifizierung des Clients vornehmen. Für das Übertragen von Binärdaten ist FTP ein geeigneter Kandidat. Die Daten werden ohne Umcodieren direkt in Originalgröße übertragen, was das Datenvolumen gering hält. Zum Übertragen der Texte und skalaren Werte müssen diese in Dateien verpackt werden und dann übertragen werden. Das übertragen der Dateien allein nimmt diese aber noch nicht in die Verwaltung des CMSs mit auf. Der Webserver müsste regelmäßig überprüfen, ob neue

Dateien eingegangen sind. Das hat zur Folge, dass nach der Übertragung einer Datei zwar bekannt ist, ob diese zum Webserver übertragen und dort gespeichert werden konnte, jedoch nicht ob der Server diese Daten auch erfolgreich verarbeiten konnte. Der Server könnte im Gegenzug nach der Verarbeitung das Ergebnis als Antwort in einer Datei auf dem FTP-Server bereitstellen, jedoch müsste der Client dieses Ergebnis dann asynchron abholen und auswerten. Das Herunterladen der vom Webserver in Dateiform bereitgestellten Konfigurationen für den Client wäre problemlos möglich.

3.8.2 SMTP[23] / POP3[24] (Email)

Der Client könnte seine Inhalte auch als Email über SMTP versenden. Dafür könnten die Daten im Text der Email oder als Dateianhänge versendet werden. Die Texte können direkt übertragen werden. Die skalaren Werte könnte man als Text kodiert übertragen. Da SMTP ursprünglich nur für den Datenaustausch von 7-Bit-ASCII Zeichen vorgesehen war, werden die Binärdaten zur Übertragung als Emailanhang Base64[25] kodiert. Die Kodierung vergrößert den Datenstrom um 1/3.

Der Webserver könnte eine Emailadresse und ein Emailkonto bereithalten, zu der bzw. dem der Inhalt geschickt werden könnte. Es wäre möglich den Webserver so zu konfigurieren, dass auf dem Emailkonto nur Emails von authentifizierten Clients entgegengenommen werden. Zum Bereitstellen der Konfigurationen für den Client würde der Webserver für den Client ebenfalls ein Emailkonto bereithalten, welches der Client dann per POP3 nach Authentifizierung abfragen könnte und die bereitgestellten Konfigurationen empfangen würde.

3.8.3 HTTP[26] POST

Mit dem HTTP Protokoll ist es möglich Daten von Webservern abzurufen. Mit einer POST Anforderung können einzelne Parameter, aber auch größere Datenmengen, und Binärdaten übertragen werden. Für diese Übertragung

---

[23] **SMTP** steht für Simple Mail Transfer Protocol und ist ein Standard Protokoll zum Übertragen von Mails über Computernetze. RFC821 (28)
[24] **POP3** bedeutet Post Office Protocol 3 und ist ein Standard Protokoll mit dem Clients Mails beim Server abholen. RFC1939 (29)
[25] **Base64** ist eine Kodierung zum Übersetzen von 8 Bit Daten in eine Zeichenfolge, die aus 65 verschiedenen Codepage unabhängigen Zeichen besteht. Hierbei werden 3 Nutzbytes in 4 Zeichenfolgebytes kodiert. RFC4648 (27)
[26] **HTTP** steht für Hypertext Transfer Protocol und ist ein Protokoll auf Anwendungsebene zum Übertragen von Daten über Netzwerke. RFC2616 (35)

werden die Parameter und Binärdaten URL-kodiert[27]. Dies ist ein Verfahren, welches zur Kodierung der Daten nur Zeichen verwendet, die nach RFC3986 erlaubte Zeichen zur Kodierung einer URL sind. Nicht erlaubte Zeichen werden durch ein %-Zeichen gefolgt vom dreistelligen ASCII-Code des Zeichens kodiert. Im ungünstigsten Fall, wenn jedes Zeichen kodiert werden muss, vergrößert sich der Datenstrom um den Faktor vier. Das http-Protokoll unterstützt Mechanismen wie „Basic Authentication“ (18) und „Digest Access Authentication“ (18) zur Authentifizierung des Clients gegenüber dem Webserver. Da die POST-Anforderung direkt vom Webserver und dem CMS bearbeitet wird, kann auch eine qualifizierte Antwort über die erfolgreiche Verarbeitung der übertragenen Inhalte an den Client gesendet werden.

3.8.4 XML-RPC

XML-RPC (19) ist ein Standard, der einen Mechanismus beschreibt, mit dem man in verteilten Systemen Methoden oder Funktionen auf entfernten Maschinen aufrufen kann. XML-RPC verwendet für den Aufruf der Methoden und Funktionen das HTTP-Protokoll. Es können die Datentypen Integer, Double, Boolean, String, DateTime (ISO8601), Base64 sowie Strukturen und Arrays dieser Typen als Parameter und Funktionsergebnisse verwendet werden. Die zu übertragenden Parameter und die Funktionsergebnisse werden dabei in XML dargestellt. Durch das Darstellen der Parameter in XML kann sich der Datenstrom bei kurzen Parameterwerten bis zu einem Faktor von ca. 50 vergrößern. Binärdaten werden vor der Übertragung Base64 kodiert. Dadurch wird der Datenstrom genau um 1/3 vergrößert. Dafür verändert sich der Datenstrom bei der anschließenden URL-Kodierung der HTTP-Post Anforderung nicht mehr, denn Base64 verwendet nur Zeichen, die in URL-Kodierung nicht kodiert werden müssen. Authentifizierung wird wie bei HTTP gleichermaßen unterstützt. XML-RPC ist neben PHP und C# bereits Bestandteil vieler Programmiersprachen und daher gut geeignet Daten zwischen unterschiedlichen Systemen auszutauschen. Da es sich um einen Methodenaufruf beim Webserver handelt, erhält der Client als Antwort ein konkretes Ergebnis über den Erfolg seiner Anfrage und der Verarbeitung der Daten.

3.8.5 .NET XML-Webservice

Analog zu XML-RPC ist im Microsoft .Net Framework ein Mechanismus zum Aufrufen entfernter Methoden über HTTP und XML bereitgestellt. Hierfür ist

---

[27] **URL-Kodierung** oder Prozentkodierung ist ein Mechanismus nach RFC3986 (25) zur Kodierung von Zeichen, die nach RFC3986 (25) nicht in URLs enthalten sein dürfen.

jedoch serverseitig die Verwendung des .Net Frameworks über eine eigene Anwendung oder ASP.Net erforderlich, welche der Webserver auf dem das System laufen soll nicht unterstützt.

3.8.6 Zusammenfassung und Auswahl des Datentransportes

Übersicht über die Eigenschaften der untersuchten Übertragungsverfahren für die Schnittstelle zwischen Client und Webserver.

Technologie	Erreichbarkeit Betrieb als Server	Authentifizierung des Clients	Erfolgsmeldung nach Verarbeitung	Volumen Vergrößerung	Binär/Text/Werte
FTP	Ja	Ja	Nein	Nein	Ja/Ja/Ja
SMTP/POP3	Ja	Ja	Nein	ca. 33%	Ja/Ja/Ja
HTTP POST	Ja	Ja	Ja	Bis zu 400% im schlechtesten Fall	Ja/Ja/Ja
XML-RPC	Ja	Ja	Ja	Bei großen Binärmengen ca. 33%	Ja/Ja/Ja

Tabelle 2 Eigenschaften der Datentransporte für die Schnittstelle Client/Webserver

Die Verfahren FTP und SMTP/POP3 eignen sich aufgrund der fehlenden unmittelbaren Antwort über den Erfolg der Verarbeitung der Daten nicht so gut wie HTTP und XML-RPC. Nachteilig an HTTP ist der bis zu viermal größere Datenstrom der Binärdaten. Bei XML-RPC wird das durch vorheriges kodieren vermieden. Hier bleibt nachteilig der Überhang durch die XML Darstellung der Parameter anzumerken. Da aber in dem geplanten System wesentlich mehr Binärdaten statt Werte in Parametern übertragen werden, fällt die Wahl hier auf XML-RPC.

Wollte man das Datenvolumen weiter reduzieren, wäre noch ein hybrides Verfahren, welches sich aus einer Kombination mehrerer Verfahren zusammensetzt, denkbar. Ein Übertragen der Binärdaten via FTP, bei dem der Datenstrom in Originalgröße übertragen wird, gefolgt von einem HTTP Post Aufruf mit den Funktionsparametern, welcher auf dem Webserver die Verarbeitung der Daten auslöst und die Antwort über die erfolgreiche Verarbeitung liefert.

3.9 Anzeige beim Anwender

Der Anwender hat keinen aktiven Einfluss auf das System. Dennoch spielt er eine wichtige Rolle. Das System existiert ja nur für den Zweck die Inhalte für den Anwender zu erfassen und sie ihm aufbereitet zur Verfügung zu stellen.

Das Nutzen der Daten muss für den Anwender einfach und unkompliziert sein. Es müssen sich die Inhalte darstellen lassen.

3.9.1 HTML

Die Aufbereitung und Anzeige der Inhalte als HTML Seiten ist serverseitig unproblematisch. Auf Anwenderseite ist die Grundvoraussetzung ein Internetbrowser. Dieser kann als vorhanden angenommen werden, da der Anwender ja auch den anderen Inhalt der Homepage damit betrachtet. Somit ist keine weitere Installation beim Anwender notwendig und er kann den ihm vertrauten Browser verwenden. Diese Lösung ist weitestgehend Plattformunabhängig, da alle gängigen Desktopbetriebssysteme über aktuelle Browser verfügen.

3.9.2 Widget

Ein Widget ist ein kleines Hilfsprogramm, welches in eine komplexere Umgebung eingebettet ist. Mit so einem Widget könnte man den aktuellen Inhalt anzeigen. Der Vorteil, es gibt viele Widgets für die verschiedenen Widgetumgebungen, die in der Lage sind Bilder einer Webcam anzuzeigen. Nachteil für den Anwender ist die notwendige Installation eines solchen Widgets. Die Plattformunabhängigkeit geht dadurch verloren.

In erster Linie wird in der angestrebten Lösung die Bereitstellung der Inhalte über HTML verfolgt. Dieses Vorgehen erscheint am flexibelsten zu sein und beim Anwender auf die größte Akzeptanz zu stoßen.

4 Design zur Realisierung

Die Teilsysteme, welche in der „Systemübersicht“ in Abschnitt 2.1 dargestellt sind, sollen nun unter Berücksichtigung der Anforderungen und der ausgewählten Technologien entwickelt werden.

4.1 Clientanwendung

Die Clientanwendung übernimmt alle für den autonomen Betrieb des Clients notwendigen Aufgaben. Neben der primär wahrnehmbaren Aufgabe regelmäßig Inhalt zu erzeugen und diesen zu übertragen, ist es Aufgabe der Anwendung den Energiebedarf des Gerätes so gering wie möglich zu halten und bei zu geringer vorhandener Energie die primäre Aufgabe einzuschränken um eine sichere Aufrechterhaltung der restlichen Mechanismen zu gewährleisten.

Für die Architektur der Anwendung wird ein Model mit zwei Threads vorgesehen. Der erste Thread, der gestartet wird, ist der Anwendungsthread. Dieser Thread hat die Hauptaufgabe die Benutzerschnittstelle darzustellen und die Benutzereingaben zu verarbeiten. Dieses Verhalten kann als passiv bezeichnet werden, da dieser Thread selbstständig keine Aktionen auslöst, sondern immer nur auf Ereignisse reagiert. Beim Starten des Anwendungsthreads, was ja auch ein Ereignis ist, auf das der Thread reagiert, ist dieser noch dafür verantwortlich den Workerthread zu starten. Dieser Workerthread implementiert ein aktives Verhalten und löst eigenständig die Aktionen zur Erfüllung der Aufgabe der Anwendung aus.

Neben den beiden Threads gibt es weitere Klassen zur Kontrolle der Hardware, Konfiguration zum Erfassen der Bilder und zur Kommunikation mit dem Webserver. Diese Klassen werden in den nächsten Abschnitten näher beschrieben.

4.1.1 Hardwaresteuerung und Energieverwaltung

Um der Anforderung eines energiearmen Betriebes gerecht zu werden und damit einen kleiner dimensionierten Solargenerator zu ermöglichen, ist es sinnvoll, wenn die Anwendung die Energie des Gerätes steuert.

Zwei Möglichkeiten zum Energiesparen stehen hier im Vordergrund. Wenn die Anwendung keine aktiven Aufgaben erfüllt und nur auf den nächsten Zeitpunkt zum Erfassen einer Aufnahme wartet, ist es sinnvoll das Gerät in den „suspend“ Modus zu versetzen. In diesem Zustand ist der Energiebedarf des Gerätes stark reduziert. Es werden alle Anwendungen angehalten und nur auf

eintretende Ereignisse wie z.B. das Ablaufen eingestellter Timer reagiert. Eine weitere Möglichkeit viel Energie zu sparen steckt in Kontrolle des GSM-Moduls. Das GSM-Modul verbraucht im Betrieb durch die ständige Empfangsbereitschaft und gelegentliche Kommunikation mit dem GSM-Netz Energie. Bei der angestrebten Aufnahme und Übertragung von zehn Bildern pro Tag wird das GSM-Modul nur ca. 2% des Tages gebraucht. Hier liegt es also nahe das GSM-Modul erst vor der Übertragung zu aktivieren und dann nach der Übertragung wieder zu deaktivieren.

Die grundlegenden Mechanismen zum Steuern des Gerätezustandes zwischen den Zuständen „on“ und „suspend“ werden von der Windows Mobile API bereits zur Verfügung gestellt und können einfach aufgerufen werden.

Zum Steuern der GSM Hardware und Verbindung muss man auf die Funktionen der Windows TAPI[28] zugreifen. Es gibt von Alex Feinman eine Warpperbibliothek (20), welche die Aufrufe der TAPI komfortabel in C# kapselt.

Die Mechanismen zum Steuern des Gerätezustandes der GSM Hardware, GSM und GPRS Verbindung werden in einer Klasse `HWController` gekapselt. Diese Klasse entspricht dem Entwurfsmuster Singleton nach (13). Das Singleton Entwurfsmuster bietet den Vorteil, dass es von einer Klasse genau ein Exemplar gibt, auf das über eine statische Methode von jedem Teil der Anwendung zugegriffen werden kann ohne die Referenz auf die Instanz zu kennen.

Der Workerthread kann über diesen `HWController` die Hardware mit einfachen synchronen und nicht blockierenden Aufrufen steuern und abfragen.

4.1.2 Konfiguration

Um die Anwendung flexibel zu halten sind einige Parameter in einer Konfiguration zusammengefasst.

Es soll konfiguriert werden können, ob die Anwendung die Steuerung der Geräteenergie des GSM-Moduls und der GPRS-Verbindung übernehmen soll. Unter einigen Umständen könnte es notwendig sein, dass die Anwendung einzelne Module nicht steuert. Dieses könnte z.B. der Fall sein, wenn Module gemeinsam mit einer anderen Anwendung auf dem Gerät genutzt werden. Zu diesem Zweck stellt die Konfiguration die Parameter ControlPower, ControlGSM und ControlGPRS bereit.

---

[28] **TAPI** steht für Telephony Application Programming Interface und ist eine von Microsoft und Intel eingeführte Softwareschnittstelle für Telefonieanwendungen.

Der Pfad zum Webserver, die Webservicemethoden und das Passwort für den Webservice sollen auch in Parametern eingestellt werden können. Damit wird es möglich das Gerät an Änderungen der Infrastruktur anzupassen.

Ein ganz wesentlicher Punkt der Konfiguration ist der Zeitplan. Der Zeitplan entscheidet wie viele Bilder pro Stunde aufgenommen werden sollen. Dieser Zeitplan orientiert sich an Wochentagen und wiederholt sich so alle sieben Tage. Für jeden Tag und jede Stunde kann ein Wert von 0 bis 60 eingestellt werden. Dabei steht 0 für kein Bild erfassen und die Werte 1-60 jeweils für den Abstand zwischen weiteren Aufnahmen in dieser Stunde.

Die Konfiguration wird in einer XML-Datei auf dem Gerät gespeichert. Für die Verwendung innerhalb der Anwendung wird die Konfiguration durch eine Klasse `Settings` repräsentiert. Diese Klasse entspricht ebenfalls dem Entwurfsmuster Singleton nach (13). Beim ersten Zugriff auf die Klasse werden die Einstellungen aus der XML-Datei in die Felder der Klasse eingelesen und stehen der Anwendung zur Benutzung bereit. Über eine Methode Update kann die Klasse dazu veranlasst werden die geänderten Einstellungen wieder in eine XML-Datei zu schreiben.

Die Speicherung als XML-Datei wurde gewählt um ein einheitliches Austauschverfahren zwischen Webeserver und Clientanwendung zu etablieren. Der Webserver kann dem Client Konfigurationen in XML bereitstellen, welche der Client dann übernehmen und speichern kann. Dieses ermöglicht eine Veränderung der Konfiguration aus der Ferne und nicht nur von der Benutzerschnittstelle direkt am Gerät.

4.1.3 Bild aufnehmen

Eine eigentlich trivial klingende Anforderung an die Anwendung ist das Aufnehmen der Bilder. Leider sind die Möglichkeiten die hier durch das .Net Compact Framework geboten werden sehr spartanisch und nicht auf Automation ausgelegt.

Das .Net Compact Framework stellt einen `CameraCaptureDialog` zum Aufnehmen von Bildern zur Verfügung. Bevor der Dialog zur Anzeige gebracht wird, kann man einige Einstellungen wie z.B. Aufnahmetyp Bild/Video, Auflösung, Zielpfad und Dateiname einstellen. Wenn der Dialog dann modal aufgerufen wird, wird vom .Net Compact Framework der Dialog zum Aufnehmen des Bildes angezeigt. Dieser Dialog ist aber nicht Bestandteil des .NET Compact Frameworks, sondern wird vom Gerätehersteller bzw. Kamerahersteller zur Verfügung gestellt. Somit unterscheiden sich die Dialoge

in ihrem Erscheinungsbild und ihrer Funktion von Gerät zu Gerät. Bei dem für die Beispielimplementierung verwendeten Gerät ist das Einstellen der Auflösung und des Dateinamens wirkungslos. Der Dialog erscheint immer mit den Einstellungen für ein Bild von 640*480 Bildpunkten. Die Bilder werden immer unter einem Dateinamen nach dem Bildungsmuster IMAGE_****.jpg gespeichert, wobei **** eine fortlaufende Zahlenfolge ist. Ein weiterer Nachteil ist die Tatsache, dass der Dialog auf die Interaktion mit einem Benutzer angewiesen ist, der nach dem Ausrichten der Kamera den Auslöser betätigt um ein Bild zu erfassen.

Eine andere Möglichkeit ein Bild von der Kamera aufzunehmen ist die Verwendung von DirectShow. Bei dieser Methode wird ein Graph von verschiedenen Filtern aneinander gereiht und ausgeführt. Die Stärken von DirectShow liegen in der Verarbeitung von Videostreams. Für die Aufnahme eines Bildes ist es erst mal notwendig sich einen Graph zu erstellen. Nun kann man sich ein Handle auf den Treiber für die Kamera holen und einen Capturefilter für die Kamera einrichten und am Anfang in den Graph einfügen. Für die Ausgabe des Bildes in eine Datei wird ein Diskwriterfilter erzeugt und an den Graph angehängt. Wenn man nun den Graph ausführt, dann wird das Bild aufgenommen, in eine Datei gespeichert und der Graph wieder beendet.

Bei Studien dieser Methode mit dem für die Beispielimplementierung verwendeten Gerät stellt der Treiber für die Kamera leider nur eine feste Auflösung von 160*120 Bildpunkten zur Verfügung.

Lösung drei, SDK von HTC

Um die einzige Lösung alles perfekt zu erschlagen kurz zu nennen, sei auf die Kamera API des Kameraherstellers hingewiesen. Damit wäre es möglich alle Einstellungen vorzunehmen und das Aufnehmen des Bildes selber auszulösen. Diese ist leider nicht frei erhältlich, sondern nur entgeltlich für Gerätehersteller gegen Unterzeichnung einer nicht Veröffentlichungsvereinbarung erhältlich.

Abwägen der Lösungen:

Die Vorteile der API des Kameraherstellers, alles einstellen zu können, rücken schnell in den Hintergrund, wenn man die Nachteile betrachtet. Die Plattformunabhängigkeit ginge gänzlich verloren. Die Verwendung der API des Kameraherstellers ist auch aus Kostengründen nicht sinnvoll für dieses Projekt. Somit ist zwischen den beiden erstgenannten Lösungen abzuwägen.

Die Vorteile der Verwendung des `CameraCaptureDialogs` aus dem .Net Compact Framework sind der einfache Aufruf und die wesentlich bessere

Bildqualität gegenüber der DirectShow Lösung. Nachteil ist die erwartete Interaktion mit einem Anwender, der den Auslöser betätigt.

Diesen Nachteil hat die Lösung mit DirectShow nicht, da hier der fertig erstellte Graph ausgeführt werden kann und ein Bild aufnimmt. Jedoch sind die umständliche Verwendung der C++ API, die aufwendige Initialisierung des Graphen und die geringe Auflösung des Treibers beim Gerät für die Beispielimplementierung eindeutig nachteilig.

Ein Ausbau der Nutzung des `CameraCaptureDialog` aus dem .Net Compact Framework bringt hier die Lösung. Die Notwendigkeit den Auslöser zu betätigen kann simuliert werden. Zu diesem Zweck konfiguriert man den Dialog und bringt ihn zur Anzeige. Timer gesteuert wird nun wenig später, hier drei Sekunden, regelmäßig eine Nachricht an den Dialog gesendet, welche ihn darüber informiert, dass der Benutzer angeblich den Auslöser betätigt hat. Der Dialog nimmt daraufhin das Bild auf und speichert es in eine Datei. Von dieser Stelle an übernimmt wieder die Anwendung die Kontrolle und das erfasste Bild kann weiter verarbeitet werden.

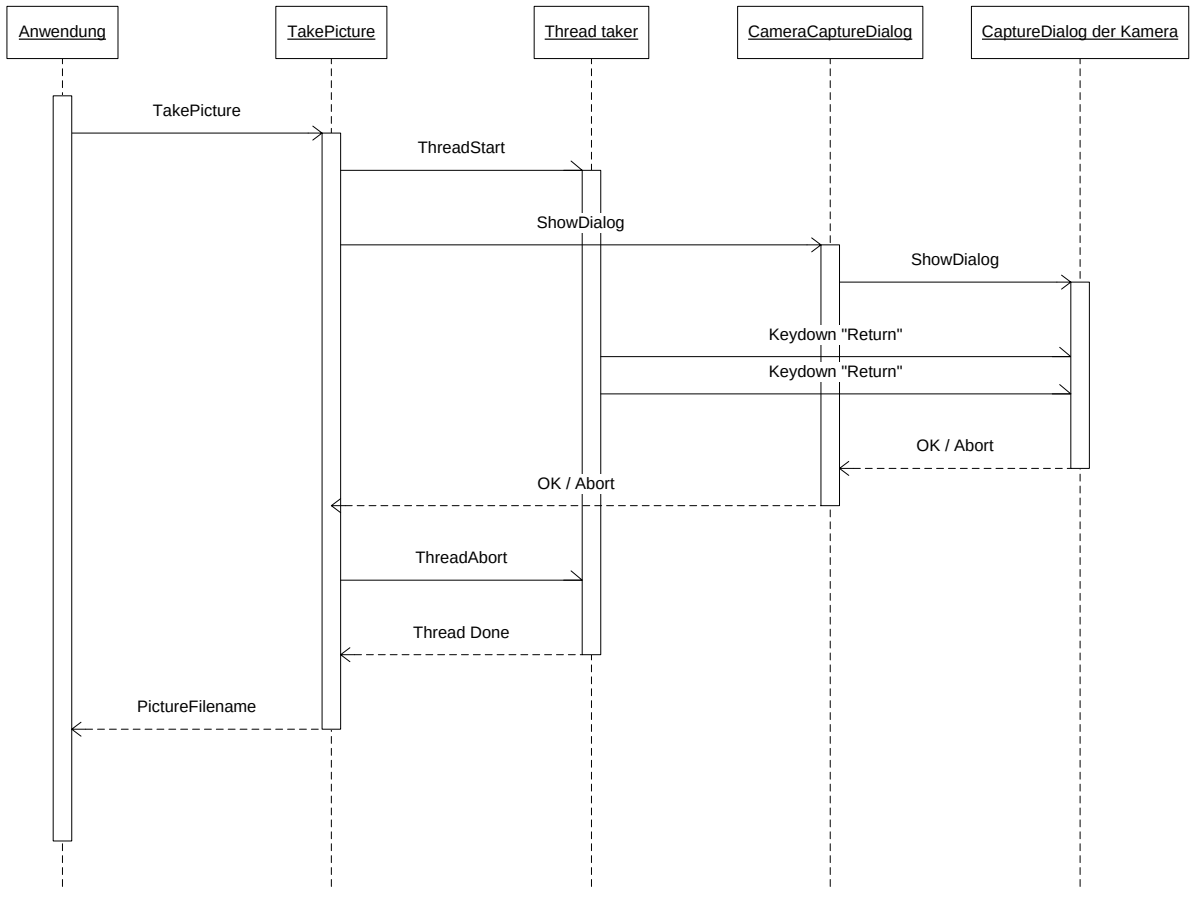

Abbildung 4.1 Sequenzdiagramm der Aufnahme eines Bildes

4.1.4 Schnittstelle und XML-RPC

Für die Übertragung der erfassten Daten sind die in Abschnitt 3.8 ausgewählten Technologien und Protokolle denkbar. Damit es möglich wird den Übertragungsweg für die Anwendung transparent zu kapseln und dennoch austauschbar oder auswählbar zu machen, bietet sich die Verwendung des Strategie-Verhaltensmusters an. Bei der Beispielimplementierung kommt eine an das Strategie-Verhaltensmuster aus (13) angelehntes Klassenmodell zur Anwendung.

Die Vorteile dieser Kapselung liegen darin, dass das anwendende Objekt immer nur mit einer Referenz auf die abstrakte Basisklasse der Strategie arbeitet. Die unterschiedlichen Verhalten werden in den von der Strategie abgeleiteten Klassen implementiert und haben alle mindestens die einheitliche Schnittstelle der Basisklasse.

Die Abbildung zeigt die Struktur, die sich für das Strategie-Verhaltensmuster ergibt.

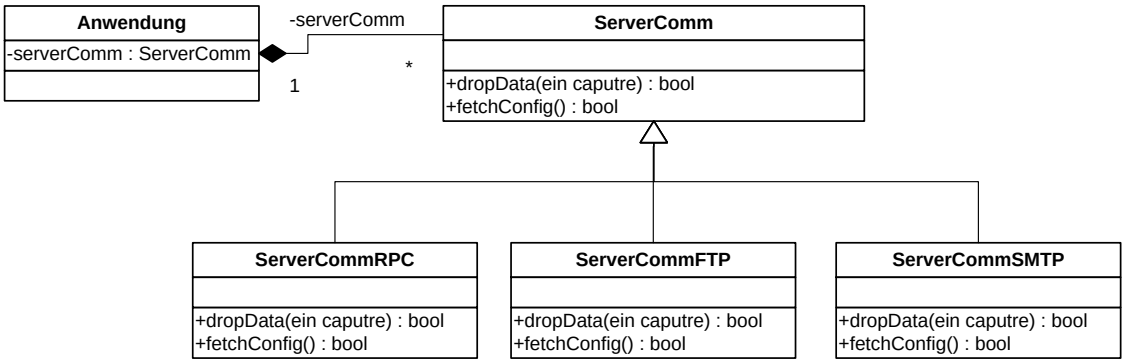

Abbildung 4.2 Struktur des Strategie-Verhaltensmusters für die Schnittstelle zum Webserver

Für das Design wird hier die abstrakte Klasse `ServerComm` und die davon abgeleitete konkrete Klasse `ServerCommRPC` diskutiert.

XML-RPC wird nicht vom .Net Compact Framework zur Verfügung gestellt. Es gibt aber eine externe XML-RPC Bibliothek (21) von Charles Cook, die sich auch unter dem .Net Compact Framework verwenden lässt.

Mit XML-RPC können Methoden auf entfernten Stationen eines verteilten Systems aufgerufen werden. Um dieses für den Aufrufenden transparent zu

gestalten wird auf der lokalen Seite der XML-RPC Schnittstelle ein Proxyobjekt erzeugt, dessen Methoden die gleiche Signatur der Parameter und Antwortwerte aufweisen, wie auch die Methoden auf der entfernten Station.

Für die Methoden des Webservices ergibt sich folgende Darstellung wie sie von der Anwendung über den Proxy im Webserver aufgerufen werden.

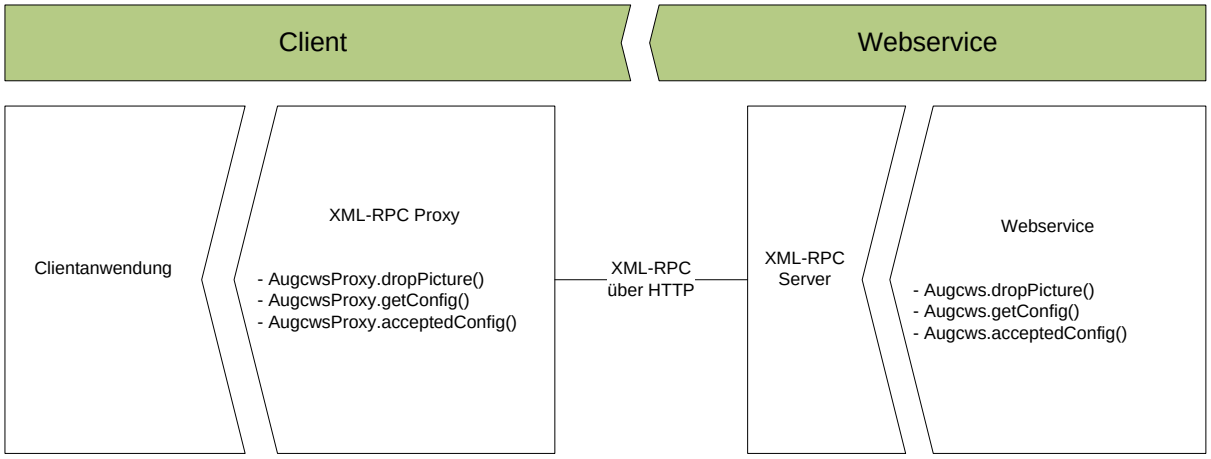

Abbildung 4.3 Clientzugriff über XML-RPC Proxy auf Webservice

Der Webservice stellt drei Methoden zur Verfügung. Die Parameter und Aufgaben der einzelnen Methoden sollen hier benannt werden, die Verarbeitung und Arbeitsweise werden weiter unten im Abschnitt 4.2.2 vorgestellt.

Die Methode `dropPicture()` nimmt als Parameter Jahr, Monat, Tag, Stunde, Minute der Aufnahme, den Dateinamen des Bildes, die Binärdaten des Bildes und die Liste der zusätzlichen Wertepaare entgegen. Als Antwort liefert die Methode einen Zahlenwert, der Aufschluss über den Erfolg der internen Verarbeitung der Daten liefert. Die Anwendung kann daran entscheiden, ob die Daten als erfolgreich übertragen behandelt werden können.

Die Methode `getConfig()` nimmt als Parameter eine ID als Identifikation der aktuellen Konfiguration entgegen. Der Webserver kann unter anderem anhand dieser ID bestimmen, ob eine neue Konfiguration an den Client gesendet werden soll. Ist dies der Fall, dann ist der Rückgabewert der Methode eine Zeichenkette, welche die neue in XML formulierte Konfiguration enthält. Wenn keine neue Konfiguration zu übertragen ist, dann ist die zurückgegeben Zeichenkette leer.

Hat ein Client eine neue Konfiguration erhalten, erfolgreich übernommen und gespeichert, dann kann er die Methode `acceptedConfig()` des Webservers

aufrufen. Die Methode erwartet als Parameter die ID der aktuell übernommenen Konfiguration. Durch diesen Aufruf kann auf dem Webserver verfolgt werden, welche Konfiguration gerade beim Client aktiv ist.

Um Fremdeinspielen von Daten oder die Veränderung von Daten auf dem Übertragungsweg zu erkennen, wird jedem Methodenaufruf noch ein Hashwert als vorangestellter erster Parameter übergeben. Dieser Hashwert wird aus allen Parametern gemeinsam mit dem Passwort des Webservers gebildet. Der Webserver macht das auch mit den Parametern und seinem Passwort. Wenn die Daten konsistent sind, dann kann davon ausgegangen werden, dass sie genau so vom Client gesendet worden sind. Wiedereinspielung von Daten wird durch diese einfache Sicherung nicht erkannt, stellt jedoch kein Problem dar. Der Webserver erkennt den identischen Inhalt und speichert ihn nicht erneut.

4.1.5 Benutzerschnittstelle

Die Benutzerschnittstelle, die vom Anwendungsthread angezeigt wird, darf für einen autonomen Dienst gerne rudimentär ausfallen, da die Interaktion mit einem Benutzer in der Regel nicht notwendig ist oder lediglich informativen Charakter hat. Aus diesem Grund besteht die Benutzerschnittstelle auch nur aus zwei Dialogen.

4.1.5.1 Statusanzeige/MainForm

Das Hauptfenster der Anwendung besteht aus einer Anzeige des aktuellen Zustandes des Workerthreads, dem Zeitpunkt zu dem die nächste Aufnahme geplant ist und einer Textbox, in der Fehler und Systemmeldungen ausgegeben werden.

Über Schaltflächen ist Benutzerinteraktion möglich. Der Benutzer kann die Anwendung beenden, das Aufnehmen eines Bildes auslösen oder den Konfigurationsdialog aufrufen.

4.1.5.2 Konfiguration

Im Konfigurationsdialog kann die Konfiguration einiger Verhaltensparameter der Anwendung eingestellt werden. Der Konfigurationsdialog ist ein View und Controller nach dem MVC-Paradigma auf die `Settings` Klasse.

Auf eine Konfiguration des Terminplanes zum bestimmen der Aufnahmezeitpunkte wurde in der Anwendung verzichtet. Dieser Terminplan kann nur durch den Administrator am Webserver vorgenommen und dem Client zum Abrufen bereit gestellt werden.

4.1.6 Automat im Workerthread

Der vom Anwendungsthread gestartete Workerthread muss eigenständig Aktionen auslösen und auf bestimmte Ereignisse abhängig von seinem eigenen Zustand reagieren. Um dieses Verhalten zu erreichen, wird im Workerthread ein Automat implementiert, der abhängig vom eigenen Zustand Aktionen auslöst, auf Ereignisse reagiert oder seinen Zustand ändert. In der folgenden Abbildung ist der ringförmige Zyklus von der Erzeugung einer Aufnahme bis hin zu nächsten Aufnahme dargestellt. Ein geeigneter Einstieg in den Zyklus ist der Zustand `PowerOffGSM`, in den nach dem Initialisieren automatisch gewechselt wird.

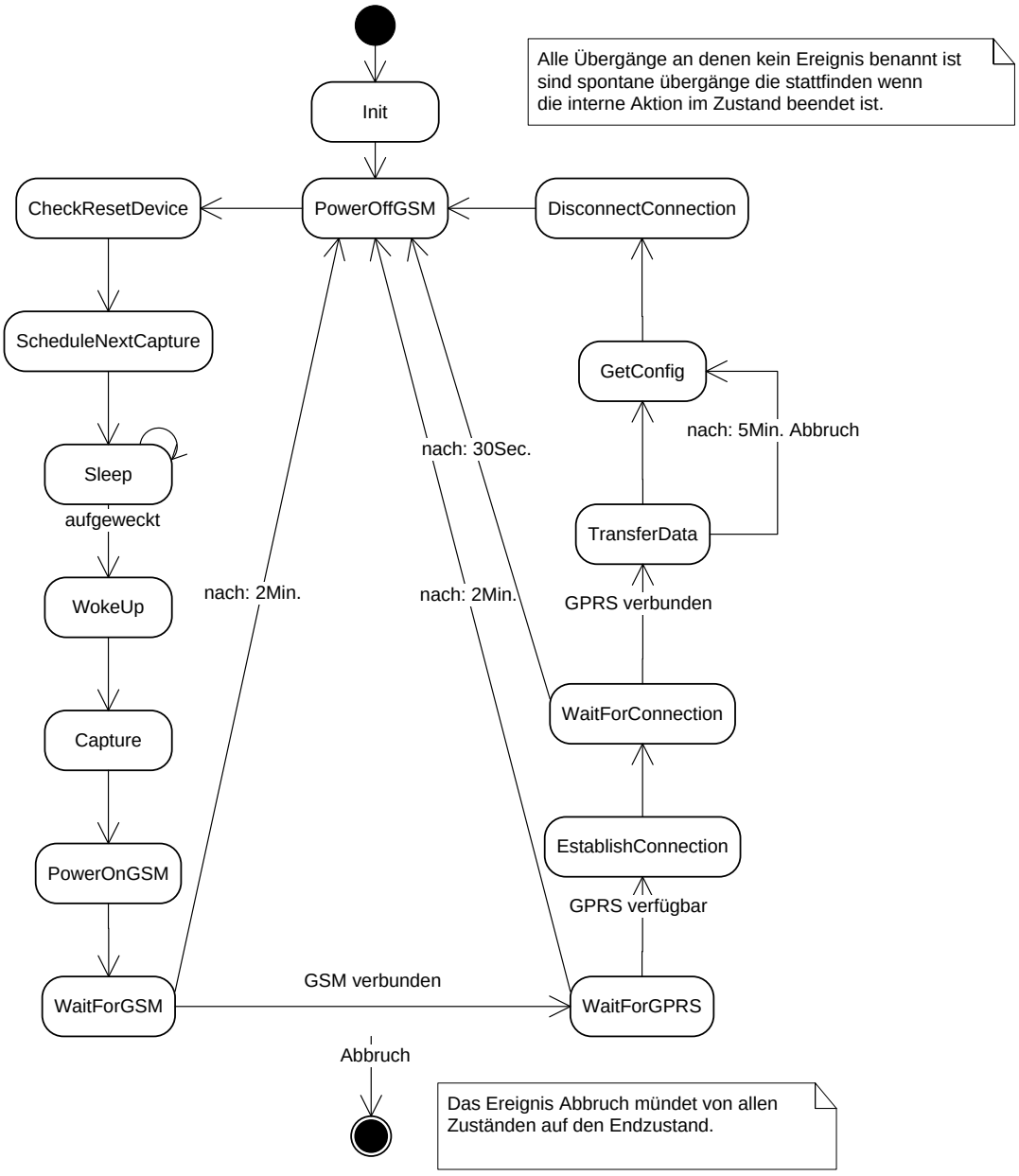

Abbildung 4.4 Zustandsdiagramm des Workerthread

4.1.6.1 Zustand Init

Dieses ist der erste Zustand, der beim Start des Automaten eingestellt wird. In diesem Zustand können Aktionen und Initialisierungen vorgenommen werden, welche einmal zum Start des Automaten ausgeführt werden sollen. Dieser Zustand ist derzeit ungenutzt und mündet sofort in den Zustand `PowerOffGSM`.

4.1.6.2 Zustand PowerOffGSM

Wenn in der Konfiguration eingestellt ist, dass die Anwendung die Energieversorgung des GSM Moduls steuern soll, dann wird überprüft ob das GSM Modul derzeit aktiv ist und ggf. deaktiviert. Danach wechselt der Automat in den Zustand `CheckResetDevice`.

4.1.6.3 Zustand CheckResetDevice

In diesem Zustand wird überprüft, ob bereits drei oder mehr aufeinanderfolgende Übertragungen des Inhaltes zum Webserver fehlgeschlagen sind. Fehlgeschlagene Übertragungen werden mit dem Zähler für Fehlverbindungen (`connectionErrors`) gezählt. Ist der Wert des Zählers größer zwei, dann wird ein Neustart des Gerätes ausgelöst. Ist dies nicht der Fall, dann wird in den Zustand `ScheduleNextExecution` gewechselt.

4.1.6.4 Zustand ScheduleNextExecution

Dieser Zustand ist für die Berechnung des nächten Termins zum Erzeugen von Inhalt verantwortlich. Der nächste Termin wird anhand der Termintabelle aus der Konfiguration ermittelt. Ist in der Tabelle kein Termin vorhanden, wird ein Termin in 24 Stunden angenommen. Würde kein neuer Termin gesetzt werden, dann würde eine vom Administrator auf dem Webserver bereitgestellte Konfiguration niemals mehr heruntergeladen und aktiv werden.

Ist der Termin ermittelt, wird ein Timer angelegt und auf den ermittelten Termin eingestellt. Das Eintreten des Termins löst ein Ereignis aus, welches den Automaten vom Zustand `Sleep` in den Zustand `WokeUp` wechseln lässt. Dieser Vorgang entspricht dem Ereignis Aufgeweckt aus dem Zustandsdiagramm.

Ist der Termin eingestellt, der Wechsel in den Zustand `Sleep` vorbereitet und der Timer aktiviert, kann das Gerät zum Energiesparen in den Standby Modus versetzt werden, sofern dies in der Konfiguration aktiviert ist und die Anwendung diesen Modus kontrollieren soll.

4.1.6.5 Zustand Sleep

Befindet sich der Automat im Zustand `Sleep` nimmt er keine aktiven Aufgaben wahr. Durch ein Ereignis vom Timer, welches bei Erreichen des Zeitpunktes zur nächsten Aufnahme ausgelöst wird, wechselt der Automat in den Zustand

`WokeUp`. Auch der Anwender kann an der Benutzerschnittstelle das Aufwecken aus dem `Sleep`-Zustand auslösen.

4.1.6.6 Zustand WokeUp

Der Automat gelangt nach jedem Reaktivieren aus dem Standby Modus in diesen Zustand. Der Timer zum Aufwecken wird an dieser Stelle angehalten und gelöscht. Der Automat wechselt anschließend in den Zustand `CheckEnergy`.

4.1.6.7 Zustand CheckEnergy

An dieser Stelle soll entschieden werden, ob die im Geräteakku vorhandene Energie derzeit ausreicht ein Bild zu machen und zum Webserver zu übertragen. Als Messgröße kann zur Entscheidung der Ladestand des Geräteakkus herangezogen werden. Eine mögliche einfache Strategie könnte bei einer unterschrittenen Restladung des Geräteakkus das Aufnehmen des Bildes unterbinden. Auch ein zweistufiges Kriterium könnte sinnvoll sein. Bei Unterschreiten der ersten Grenze z.B. 30% wird noch ein Bild aufgenommen, jedoch die zeit- und energieintensive Übertragung des Bildes wird unterbunden und erst bei einem Ladestand >30% nachgeholt. Unterschreitet der Ladestand eine weitere tiefere Grenze von z.B. 10%, wird das Aufnehmen des Bildes auch unterbunden und zum Schonen der Energiereserven ein langes Intervall bis zur nächsten geplanten Erfassung eingestellt. Hierfür könnte man die Uhrzeit mit hinzuziehen. Man könnte annehmen, dass am nächsten Tag um 12 Uhr wieder Energie vom Solargenerator bereitgestellt wird und der Ladestand wieder angestiegen ist.

Welches Verhalten nachträglich implementiert wird, kann erst nach längerer Beobachtung des Energieverlaufes im Zusammenhang mit dem Solargenerator entschieden werden.

4.1.6.8 Zustand Capture

Das eigentliche Erfassen des Inhaltes wird im Zustand `Capture` realisiert. Es wird ein Bild aufgenommen und in einer Instanz des Models `Capture` gespeichert. Des Weiteren werden noch die Betriebsdaten der Gerätebatterie erfasst und als Wertepaare im `Capture` gespeichert.

Ein `Capture` stellt ein Model nach dem MVC-Paradigma aus Abschnitt 3.6 dar. Es speichert den Dateinamen des aufgenommenen Bildes, den Zeitpunkt der Aufnahme und eine Liste von Paaren aus Name und Wert.

Dem MVC-Paradigma folgend gibt es einen Controller. Dieser konkrete `CaptureController` hat eine `Save()` Methode, mit deren Aufruf das erstellte

und mit Daten gefüllte `Capture` in eine Datei auf den internen Datenträger des Gerätes gespeichert wird. Auf den Zustand Capture folgt der Zustand `PowerOnGSM`.

4.1.6.9 Zustand PowerOnGSM

In diesem Zustand wird abhängig von der Konfiguration entschieden, ob das GMS-Modul eingeschaltet wird. Wenn das GSM-Modul eingeschaltet wird, ist `WaitForGSM` der nächste Zustand. Es wird festgelegt wie lange auf das Verbinden mit dem GSM-Netz gewertet werden soll. Erfahrungsgemäß ist bei Mobiltelefonen die Verbindung in wenigen Sekunden hergestellt. Ein Zeitrahmen von zwei Minuten sollte immer zum Verbinden mit dem GSM-Netz reichen. Wird das GSM-Modul nicht eingeschaltet, wechselt der Automat in den Zustand `EstablishConnection`.

4.1.6.10 Zustand WaitForGSM

Dieser Zustand überprüft zyklisch, ob die Wartezeit zum Etablieren der GSM-Verbindung abgelaufen ist oder die GSM-Verbindung zustande gekommen ist. Ist die Verbindung aufgebaut, wechselt der Automat in den Zustand `WaitForGPRS`. Ist die Verbindung nach Ablauf des Zeitrahmens nicht aufgebaut, wird ein Zähler für Fehlverbindungen `connectionErrors` inkrementiert und der Automat wechselt in den Zustand `PowerOffGSM` ohne eine Übertragung der Daten durchzuführen.

4.1.6.11 Zustand WaitForGPRS

Der Zustand `WaitForGPRS` verhält sich analog zu `WaitForGSM`. Einzige Unterschiede: Es wird nicht auf die GSM-Verbindung, sondern auf das Vorhandensein von GPRS gewartet. Und der Folgezustand im Erfolgsfall ist `EstablishConnection`.

4.1.6.12 Zustand EstablishConnection

Aufgabe dieses Zustandes ist das Herstellen einer Internetverbindung auszulösen. Der Verbindungsaufbau wird nur ausgelöst, wenn die Konfiguration die Steuerung der Verbindung durch die Anwendung vorsieht. Wenn die Anwendung die Steuerung nicht übernimmt, wechselt der Automat direkt in den Zustand `TransferData`. Kontrolliert die Anwendung den Aufbau der Verbindung, dann wechselt der Automat in den Zustand `WaitForConnection`.

4.1.6.13 Zustand WaitForConnection

Auch in diesem Zustand wird analog zu den Zuständen `WaitForGSM` und `WaitForGPRS` nur eine bestimmte Zeit auf das Eintreten eines Ereignisses

gewartet. Kommt die Internetverbindung vor Ablauf der Zeit zustande, wechselt der Automat in den Zustand `TransferData`. Läuft die Wartezeit ab bevor die Internetverbindung aufgebaut ist, wird auch von diesem Zustand in den Zustand `PowerOffGSM` gewechselt und auf ein Übertragen der Daten zu diesem Zeitpunkt verzichtet.

4.1.6.14 Zustand TransferData

Wenn der Automat diesen Zustand erreicht hat, ist die Internetverbindung etabliert und die Daten können zum Webserver übertragen werden. Bei Experimenten mit der Übertragung per XML-RPC ist der Aufruf der externen Methode gelegentlich blockiert. Damit solche Blockaden nicht zum Stillstand der Anwendung führen, wird für die Übertragung ein eigener Transferthread gestartet, der die Übertragung der Daten mit Hilfe der Methode `dropData()` eines von `ServerComm` geerbten Objektes durchgeführt. Des Weiteren ist die Übertragung ein zeitlich sehr ausgedehnter Vorgang. Würde die Übertragung im Workerthread laufen, bestünde in der Zeit keine Möglichkeit, den Thread von außen zu beeinflussen.

Der Automat wartet darauf, dass die zur Verfügung stehende Zeit für die Übertragung abläuft oder die Übertragung und der Transferthread erfolgreich beendet werden. Wird die Übertragung nicht rechtzeitig fertig, wird der Zähler für Fehlverbindungen `connectionErrors` inkrementiert. Um dem Anspruch an Robustheit gegen Übertragungsfehler gerecht zu werden, werden Inhalte, die aufgrund von Übertragungsfehlern nicht auf die zum Webserver übertragen werden konnten, beim nächsten Verbindungsaufbau erneut mit übertragen. Jedoch werden keine Daten übertragen werden, die älter als 24 Stunden sind. Diese Daten werden gelöscht, da sie nicht mehr aktuell sind.

Ist der Transferthread jedoch fertig geworden und hat alle Daten übertragen können, wird der Zähler für Fehlverbindungen wieder zurück auf null gesetzt und die erfolgreich übertragenen Daten werden auf dem Client gelöscht. Im Anschluss wechselt der Automat in den Zustand `GetConfig`.

4.1.6.15 Zustand GetConfig

Dieser Zustand versucht eine evtl. beim Webserver hinterlegte Konfiguration abzurufen. Es wird die Methode `fetchConfig()` eines von `ServerComm` geerbten Objektes aufgerufen. Wenn eine neue Konfiguration empfangen wird, dann wird diese im Dateisystem des Clients gespeichert und in die Anwendung eingelesen. Im Anschluss wird dem Webserver die erfolgreiche Anwendung der neuen Konfiguration noch mit einem Aufruf des Webservices

`acceptedConfig()` mitgeteilt. Nachdem alle Übertragungen abgeschlossen, sind wechselt der Automat in den Zustand `DisconnectConnection`.

4.1.6.16 Zustand DisconnectConnection

Wenn die Anwendung laut Konfiguration die Kontrolle über die Internetverbindung hat, dann wird die bestehende Internetverbindung getrennt. Der Automat wechselt anschließend in den Zustand `PowerOffGSM` und der Zyklus beginnt von vorne.

4.1.6.17 Abbrechen des Automaten

Während der Automat läuft, kann ihm jederzeit von extern durch das Setzen einer der Membervariablen signalisiert werden, die Arbeit einzustellen. Der Automat endet darauf und der Workerthread terminiert.

4.2 Webserver

Unter dem Begriff Webserver soll in diesem Zusammenhang nicht die Hardware oder Grundinstallation des Betriebssystems und einer Webserversoftware verstanden werden. Der Begriff Webserver meint in diesem Kapitel die ganzheitliche Erscheinung der Dienste und Anwendungsteile, die direkt zur Lösung der Problemstellungen dieser Arbeit dienen. Auf dem Webserver ist bereits das CMS Joomla installiert und eine Homepage wird bereits betrieben.

Realisiert werden müssen auf dem Webserver für den Inhalt die drei neuen Aufgaben Daten entgegenzunehmen, Daten zu speichern und gezielt wieder zur Anzeige zu bringen. Im Gegenzug stellt der Webserver im Backend des CMSs Funktionen zur Verfügung, mit denen der Administrator Konfigurationen erstellen und verwalten kann. Diese Konfigurationen werden für den Client zum Abruf bereitgehalten.

4.2.1 Datenspeicherung mit Active Record Pattern

Da sich in dieser Arbeit alles um den Inhalt, die eigentlichen Daten dreht, soll deren Speicherung und Struktur hier näher definiert werden.

Die Daten werden in der Datenbank MySQL des Webservers gespeichert. Der Zugriff auf die Datenbank erfolgt über die Joomla API. Die Joomla API greift mit einem Klassenmodell nach dem Active Record Pattern auf die Datenbank zu. Dieses Entwurfsmuster beschreibt einen Zugriff auf die Datenbank, bei dem ein Datensatz in der Tabelle immer genau einem Objekt entspricht. Das Objekt stellt die Methoden `load()`, `store()` und `delete()` bereit. Über diese Methoden wird das Objekt mit dem zugehörigen Datensatz der Datenbank synchronisiert. Weitere Methoden ermöglichen das Finden von Datensätzen

anhand bestimmter Kriterien. Die Membervariablen des Objektes repräsentieren die Datenbankwerte aus den Spalten. Eine Dokumentation und Implementierung des Active Record Pattern in PHP findet sich im Buch von George Schlossnagle (22). Es stellt einen Bruch zum MVC-Pattern da, weil das Model und der Controller in einem Objekt bestehen. Dennoch kann mit einem View auf das Model zugegriffen werden und es zur Anzeige gebracht werden.

In der folgenden Tabelle ist dargestellt aus welchen Feldern ein Datensatz zur Repräsentation einer Aufnahme besteht.

Feld	**Typ**	**Beschreibung**
id	Integer	Die id ist künstlicher Primärschlüssel.
capturetime	Datetime	Das Datum, an dem die Aufnahme gemacht wurde
imagefile	Varchar (255)	Der Dateinamen der Datei, in der die zugehörigen binären Bilddaten gespeichert sind
meta	Text	In diesem Feld werden die zusätzlichen Wertepaare als serialisiertes PHP Array gespeichert. Das verletzt sehr wohl die Normalform, aber da sich an den Daten nichts verändert und sie immer fest diesem Datensatz zugeordnet sind, beschleunigt es den Zugriff erheblich.
published	tinyint(1)	Ein Flag, mit dem die Daten als veröffentlicht oder nicht gekennzeichnet werden können.

Tabelle 3 Tabellenstruktur für einen Inhaltsdatensatz

Auch die Konfigurationen für den Client befinden sich als eigene Tabelle mit folgender Struktur in der Datenbank.

Feld	Typ	Beschreibung
id	integer	Die id ist künstlicher Primärschlüssel.
name	Varchar (255)	Das Datum, an dem die Aufnahme gemacht wurde
meta	text	In diesem Feld wird die eigentliche Konfiguration als serialisiertes assoziatives PHP Array gespeichert. Das verletzt sehr wohl die Normalform, aber da die Daten der Konfiguration nicht zum Selektieren herangezogen werden und auch zu keinen andren Teilen dieser Datenbank in Relation stehen, ist so ein einfaches übersichtliches Tabellenschema möglich.
active	tinyint(1)	Ein Flag, mit dem die derzeit beim Client aktive Konfiguration markiert wird. Es ist immer nur maximal eine Konfiguration als active gekennzeichnet.
published	tinyint(1)	Ein Flag, mit dem die Konfiguration, die zum Client übertagen werden soll, gekennzeichnet wird. Es ist immer nur maximal eine Konfiguration als published gekennzeichnet.

Tabelle 4 Tabellenstruktur für einen Konfigurationsdatensatz

4.2.2 Joomla Komponente

Joomla verfolgt zum Erweitern der Funktionalität ein Konzept von einzelnen nachinstallierbaren Softwareelementen. Diese Elemente haben Zugriff auf die von Joomla bereitgestellte API. Für die Implementierung der Schnittstelle zum Client, die Speicherung und Anzeige der Daten, ist eine Joomla Komponente als Erweiterung vorgesehen.

4.2.2.1 Webservice

Der Webservice stellt den Webserverteil der XML-RPC Schnittstelle zum Client bereit. Er wird gemeinsam mit der Komponente installiert. Er ist aber anders als die Komponente direkt aufrufbar und importiert die Joomla API, um auf die Datenbank zuzugreifen. Die bereitgestellten Methoden, deren Aufgabe und Signatur vorher unter Abschnitt 4.1.4 bereits beschrieben wurden, sind im Webserver beherbergt. Nach erfolgreicher Überprüfung der Konsistenz der Daten verarbeiten die Methoden die Aufrufe wie folgt:

`dropPicture()` nimmt die Daten vom Client entgegen. Die Binärdaten des Bildes werden unter dem mit übergebenem Dateinamen in das Bilderverzeichnis der Komponente geschrieben. Es wird ein Objekt angelegt

und mit Datum und Uhrzeit der Aufnahme gefüllt. Die übergebenen Wertepaare werden in den Metadaten gespeichert und das Objekt wird persistent in der Datenbank gespeichert.

`getConfig()` überprüft die Datenbank nach einem Konfigurationseintrag, dessen Flag published gesetzt ist. Wenn das zutreffend ist, dann werden die Konfigurationsdaten aus dem meta Feld als XML-String aufbereitet und zum Client übertragen. Ansonsten erhält der Client einen leeren String als Antwort und wird mit seiner aktuellen Konfiguration weiterarbeiten.

`acceptedConfig()` wird vom Client aufgerufen, wenn dieser erfolgreich eine neue Konfiguration übernommen hat, um dem Webserver die ID der nun aktiven Konfiguration mitzuteilen. Der Webserver löscht dann das Flag published der entsprechenden Konfiguration und markiert sie mit dem Flag active als derzeit aktive Konfiguration.

4.2.2.2 Ansicht Einzelbild

Nun wo bekannt ist wie die Daten in die Datenbank gelangen, kommt der Teil, der für den Anwender nach außen hin sichtbar wird. Die Anzeige der generierten Inhalte auf der Homepage. Die Komponente wird in der zentralen Inhaltsfläche der Homepage dargestellt. Sie beinhaltet eine rudimentäre Navigation mit der die verschiedenen Ansichten aufgerufen werden können. Die eine Ansicht stellt das aktuelle Bild dar, die anderen Ansichten stellen alle Bilder des aktuellen und vergangenen Tages dar.

Wird die Einzelbildansicht ohne Parameter aufgerufen, zeigt sie das letzte aufgenommene und übertragene Bild in Originalgröße mit seinen Daten an. Die Einzelbildansicht kann auch dazu verwendet werden eine bestimmte Aufnahme anzuzeigen. Dazu ist beim Aufruf die ID der Aufnahme als Parameter mit zu übergeben. Von dieser Möglichkeit macht die Tagesübersicht Gebrauch und ermöglicht es sich einzelne Bilder des Tages in der Einzelbildansicht anzeigen zu lassen.

4.2.2.3 Ansicht Tagesübersicht

Die Tagesübersicht zeigt verkleinerte Bilder aller Aufnahmen des ausgewählten Tages untereinander an. Auch bei der Tagesansicht kann ein Parameter zur Auswahl des Tages mit übergeben werden. Wird kein Tag ausgewählt, kommt die Übersicht des aktuellen Tags zur Ansicht. Jedes Bild in der Übersicht ist verlinkt und führt zur Einzelansicht der entsprechenden Aufnahme.

4.2.2.4 Download aktuelles Bild

Nicht jeder möchte immer die ganze Homepage aufrufen um ein einziges aktualisiertes Bild zu erhalten. Auch ist die Weiterverarbeitung der Daten kompliziert, wenn man sie nur eingebettet in die Homepage liefert. Aus diesem Grund wird noch ein weiterer Zugriff auf das aktuelle Bild bereitgestellt. Ein direkter HTTP-Aufruf, der immer das aktuelle Bild liefert. Dieses Bild kann dann komfortabel vom Anwender in Widgets oder anderen Anwendungen verwendet werden ohne auf die ganz Seite zugreifen zu müssen.

4.2.2.5 Komponente Administration im Backend

Das Backend ist der Bereich der den Administratoren des CMSs vorbehalten ist. Auch eine Komponente kann Konfigurationsmöglichkeiten im Backend integrieren. Für diese Komponente sind zwei Aufgabenbereiche im Backend vorgesehen.

Der erste Aufgabenbereich befasst sich mit der Verwaltung der Inhalte. Dafür werden alle erfassten Inhalte in einer Liste dargestellt und können einzeln oder gemeinsam markiert für die Anzeige auf der Seite gesperrt oder gelöscht werden. Auch dies ist ein View auf das Model, allerdings ist dieser den Administratoren vorbehalten. Die ausgewählten Aktionen werden von der Komponente verarbeitet, was der Funktion des Controllers gleich kommt. Im Anschluss werden etwaige Veränderungen persistent durch das Model in der Datenbank gespeichert.

Der zweite Aufgabenbereich ist das Erstellen und Verwalten der Konfigurationen für den Client. Auch hier werden alle bereits vorhandenen Konfigurationen namentlich in einer Liste angezeigt. Es stehen die Aktionen zum Bearbeiten, Löschen, Kopieren und neu Anlegen einzelner oder mehrerer Konfigurationen zur Verfügung.

Wählt man eine Konfiguration zum Bearbeiten aus werden alle Konfigurationsfelder und die Zeittabelle zum Verändern angezeigt. Nach dem Bearbeiten und Speichern einer Konfiguration kann dieses dann markiert und veröffentlicht werden. Veröffentlicht meint in diesem Fall für den Client zum Download bereitgestellt.

4.3 Anwender

Auf der Anwenderseite ist keine weitere Realisierung notwendig. Der Anwender kann den Inhalt einfach über einen Webbrowser von der Homepage aufrufen und ansehen.

Für Nutzer der Sidebar von Microsoft Windows Vista gibt es ein „Webcam Tracker“ Widget (23), welches im einstellbaren Intervall ein Bild aus dem Internet herunterlädt und anzeigt. Mit diesem oder anderen Widgets lässt sich der aktuelle Inhalt immer im Auge behalten.

Abbildung 4.5 Screenshot WebcamTracker Widget

5 Realisierung

Der im Kapitel 4 erarbeitete Entwurf soll nun in diesem Kapitel in einem Prototypen und einer Beispielimplementierung umgesetzt werden. Es sollen die Software für den Client und den Webserver entwickelt werden.

5.1 Client

Als Hardware für den Prototypen wird ein HTC P4350 SmartPhone mit Windows Mobile 6.1 verwendet. Neben dem Betriebssystem ist auf dem Gerät bereits das .Net Compact Framework 2.0 installiert. Entwickelt wird die Anwendung im Microsoft Visual Studio 2005. Aus der Entwicklungsumgebung heraus wird die Anwendung über eine USB-Verbindung auf die Clienthardware übertragen und dort ausgeführt.

5.1.1 Hautpanwendung und Dialoge

Die Dialoge für die Hauptanwendung werden in der Entwicklungsumgebung erstellt. In der grafischen Oberfläche können diese übersichtlich mit den Elementen zur Anzeige und Interaktion bestückt werden.

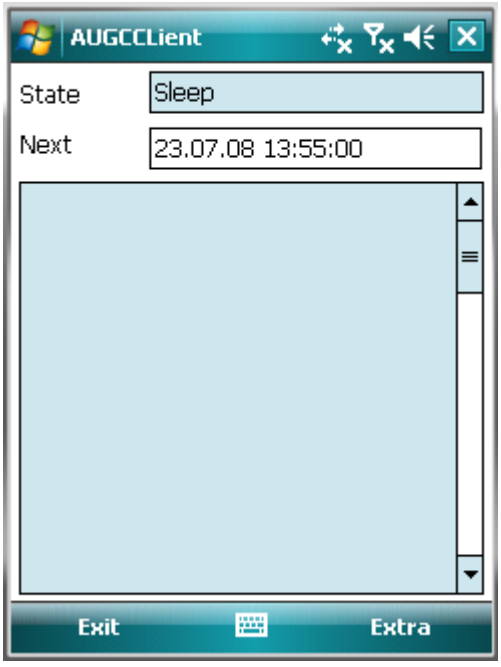

Abbildung 5.1 Statusanzeige der Clientanwendung

Abbildung 5.2 Konfigurationsdialog der Clientanwendung

5.1.2 HWController

Um die Zugriffe auf die Windows Mobile API zu kapseln und in anderen Projekten wieder zu verwenden entsteht eine DLL, die die Singleton Klasse `HWController` kapselt. Die Anwendung, insbesondere der Automat des Workerthread, kann mit kurzen prägnanten Aufrufen seine Einstellungen an der Hardware vornehmen.

Folgende Methoden und Eigenschaften werden vom `HWController` bereitgestellt, über die der Zustand des Gerätes und der GSM Hardware gesteuert und abgefragt werden kann.

```
// public methods
Public int SetSystemPowerState(string pwsSystemState,
                               PowerState StateFlags,
                               PowerRequirement Options)
                                  // Einstellen des Gerätezustands
                                  // StateFlags ON oder SUSPEND
                                  // Otions = FORCE
public void GSM_on()              // GSM-Hardware einschalten
public void GSM_off()             // GSM-Hardware ausschalten
public void LineRegister()        // Mit GSM-Netz verbinden
public void LineUnregister()      // Vom GSM-Netz trennen
public void Connection_on()       // GPRS-Verbindung aufbauen
public void Connection_off()      // GPRS-Verbindung trennen

// public properties
public bool Phone                 // GSM-Hardware on/off
public bool GSM                   // Verbindung mit dem GSM-Netz
on/off
```

```
public bool GPRS                    // GPRS Vorhanden yes/no
public bool Connection              // GPRS Verbunden yes/no
public SYSTEM_POWER_STATUS_EX2 SystemPowerStatus
                                // liefert den aktuellen Zustand der
                                // Batterie.
```

Quelltext 1 Öffentliche Methoden und Eigenschaften des HWController

5.1.3 Settings

Die Konfigurationen werden in XML formuliert. Am folgenden Beispiel kann man die beiden Blöcke der Konfiguration für das Verhalten der Anwendung und den Zeitplan für das Erfassen der Aufnahmen sehen. Die Konfiguration wird auf dem Client als XML-Datei gespeichert, jedoch beim ersten Zugriff in ein `Settings` Objekt eingelesen und der Anwendung zu Verfügung gestellt. Die eingelesenen Anwendungseinstellungen werden vom `Settings` Objekt als Eigenschaften zur Verfügung gestellt und können von der Anwendung bequem abgefragt werden.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration id="0">
  <appSettings>
    <add key="controlPower" value="0" />
    <add key="controlGSM" value="1" />
    <add key="controlGPRS" value="1" />
    <add key="webservicePath"
         value="http://www.doodiac.dyndns.org/.../webservice.php" />
    <add key="webserviceDropMethod" value="Augcws.dropPicture" />
    <add key="webserviceGetConfigMethod" value="Augcws.getConfig" />
    <add key="webserviceAcceptedConfigMethod"
         value="Augcws.acceptedConfig" />
    <add key="webservicePSK" value="TopSecret!" />
  </appSettings>
  <scedule>
    <day id="0">
      <hour id="0" value="0"/>
      <hour id="1" value="0"/>
      <hour id="2" value="0"/>
      <hour id="3" value="0"/>
         …
      <hour id="22" value="0"/>
      <hour id="23" value="0"/>
    </day>
    <day id="1">
      <hour id="0" value="0"/>
      <hour id="1" value="0"/>
         …
      <hour id="22" value="0"/>
      <hour id="23" value="0"/>
    </day>
       …
  </scedule>
</configuration>
```

Quelltext 2 XML-Konfigurationsdatei des Clients

5.1.4 ServerComm

Die Beispielimplementierung des Prototyps beschränkt sich auf die Implementierung der abstrakten Basisklasse `ServerComm` und einer konkreten Ableitung `ServerCommRPC`, in der die Kommunikation mit dem Webservice in XML-RPC implementiert ist.

Die abstrakte Basisklasse ServerComm gibt das Interface für die konkreten Klassen vor. Es sind nur die Aufgaben Inhalte übertragen und Konfiguration empfangen vorgegeben. Wie die konkreten Klassen das Verhalten implementieren ist nicht weiter vorgegeben. Darin liegt der Sinn des Strategiemusters, dass man unterschiedliche Strategien über das gleiche einheitliche Interface ansprechen kann.

```
public abstract class ServerComm
{
    abstract public bool dropData(Capture cap);
    abstract public bool fetchConfig();
}
```

Quelltext 3 Abstrakte Klasse ServerComm als Interface

5.1.5 XML-RPC

Für den Zugriff auf entfernte Methoden kapselt man den Aufruf in einem so genannten Proxy Objekt. Dieses Proxyobjekt stellt lokal die gleichen Methoden mit der gleichen Signatur wie der entfernte Webservice zur Verfügung. Nach einem Aufruf der Methode auf dem Proxyobjekt ruft diese über die Mechanismen des Webservices, hier XML-RPC, die Methode auf dem Webservice auf. Die Antwort des Webservices wird als Antwort der Methode des Proxyobjektes zurückgegeben. Die Erzeugung des Hashwertes ist im Prototyp nicht implementiert worden. Bevor eine Version den Livebetrieb aufnimmt muss das noch nachgeholt werden.

Für den konkreten Webservice ergibt sich folgende Klasse für das Proxyobjekt. Es ist hier auf die Darstellung aller Methoden verzichtet worden. Die nicht gezeigten Methoden sind aber analog zu der hier gezeigten `dropPicture()` Methode implementiert.

```
class AugcwsProxy : XmlRpcClientProtocol
{
    public struct MetaData
    {
        public String key;
        public String value;
    }

    [XmlRpcMethod]
    public int dropPicture(string md5dummy,
                           int yy, int mm, int dd,
```

```
                        int h, int m,
                        string filename,
                        byte[] binary,
                        MetaData[] meta)
    {
        this.Url = Settings.getInstance().WebservicePath;
        this.XmlRpcMethod =
Settings.getInstance().WebserviceDropMethod;
        this.Timeout = 30000;
        return (int)Invoke("dropPicture", new Object[] { md5dummy,
                           yy, mm, dd,
                           h, m,
                           filename,
                           binary,
                           meta });
    }
    …
}
```

Quelltext 4 Anwendungsschnittstelle des Webservice als Proxy Klasse

Im folgenden Quelltextauszug ist die einfache Nutzung der Proxyschnittstelle in der Methode `dropData()` der konkreten Klasse `ServerCommRPC` dargestellt. Es wird einfach eine Instanz der `AugcwsProxy` Klasse angelegt. Auf dieser Instanz kann dann die Methode `dropPicture()` aufgerufen werden. Die Methode erwartet die Angaben zum Zeitpunkt der Aufnahme, den Dateinamen des Bildes, die Binärdaten des Bildes und ein Array mit den Metadaten. Der erste Parameter ist ein MD5 Hash über alle Parameter und ein zwischen Client und Webserver vereinbartes Passwort. Damit wird der Client beim Webserver authentifiziert. Dieser MD5 Hash wird aber erst intern im `AugcwsProxy` berechnet, muss aber hier als Pseudowert mit übergeben werden, damit die Signatur der vom Webservice entspricht.

```
public class ServerCommRPC : ServerComm
{
  override public bool dropData(Capture cap)
  {
    // Senden der Daten und des Bildes als XMLRPC
    AugcwsProxy proxy = new AugcwsProxy();
    …
    try
    {
      int result = proxy.dropPicture("dontCareMd5",
                                     cap.Time.Year, cap.Time.Month,
                                     cap.Time.Day,cap.Time.Hour,
                                     cap.Time.Minute,

Path.GetFileName(cap.ImageFile),
                                     binary,
                                     meta);
    }
    catch (Exception)
    {
      return false;
    }
    return true;
  }
```

```
    …
}
```

Quelltext 5 Nutzung des AugcwsProxy durch ServerCommRPC

5.1.6 Workerthread

Die eigentliche Arbeit macht der Automat im Workerthread. Der Automat wird nicht als vollständiger Automat nach Harel implementiert, sondern lediglich als Schleife mit einer Abbruchbedingung. Der Zustand des Automaten wird in einer lokalen Variablen gespeichert und eine Switch-Anweisung führt nur die Aktionen für den aktuellen Zustand aus.

```
…
while (!abortWorker)
{
    this.setMainFormStateWrapper(workerState.ToString());
    switch (workerState)
    {
        case WorkerStates.Init:
            workerState = WorkerStates.PowerOffGSM;
            break;
        case WorkerStates.ScheduleNextCapture:
            …
        …
    }
}
…
```

Quelltext 6 Schleife und Switch-Anweisung des Automaten im Workerthread

Der Workerthread führt in den einzelnen Zuständen die Aktionen aus, die im Abschnitt 4.1.6 als Verhalten für den Automaten definiert wurden. Aufgrund der Kapselung von Zugriff auf die Hardware, Zugriff auf die Konfiguration und Übertragung der Daten hat der Automat nur die Aufgabe zu reagieren und auszulösen. Es sind keine umfangreichen Abläufe in den einzelnen Zuständen implementiert. Die einzelnen Zustände bestehen aus kurzen prägnanten Bedingungen und Aufrufen der unterliegenden Abstraktionsschichten.

5.2 Webserver

Dieser Abschnitt beschreibt die Besonderheiten bei der Implementierung der Software für den Webserver. Das CMS Joomla ist bereits installiert. Es wird der Webservice und die Komponente in PHP programmiert und dabei an die Gegebenheiten der Joomla API angepasst. Die Komponente bekommt den Namen Augcws. Ein Akronym aus der Bezeichnung „Automatic User Generated Content Web Service“ Dieser Name findet aber nur in der Verwaltung von Joomla Anwendung und kann an allen Stellen, die Anwender zu sehen bekommen, anders benannt werden.

5.2.1 Webservice mit XML-RPC bereitstellen

Die erste Hürde, die es zu nehmen gilt, ist die Bereitstellung des Webservices um die Inhalte vom Client entgegen zu nehmen. Für die Bereitstellung der XML-RPC Serverschnittstelle sind nur wenige Zeilen zum Definieren der Webservicemethoden notwendig und auf die lokal verarbeitende Methode zu verweisen. Den Rest erledigt die XML-RPC Bibliothek. Am folgenden Beispiel ist gezeigt wie der Webservice Server konfiguriert und aufgerufen wird.

```
<?
// Einbinden der Joomla API
define( '_VALID_MOS', true );
define( 'YOURBASEPATH', dirname(__FILE__) );
require_once( YOURBASEPATH . '/../../globals.php' );
require_once( YOURBASEPATH . '/../../configuration.php' );
require_once( YOURBASEPATH . '/../../includes/joomla.php' );

// Einbinden der XML-RPC Bibliothek
require_once ('xmlrpc.inc');
require_once ('xmlrpcs.inc');
require_once ('xmlrpc_wrappers.inc');

// Einbinden des Model für die speicherung der Inhalte
require_once ('./augcws.class.php');
…
// create dropPicture Method signature and documentation
$dropPicture_sig=array(array($xmlrpcInt, $xmlrpcString, $xmlrpcInt,
                        $xmlrpcInt, $xmlrpcInt, $xmlrpcInt,
$xmlrpcInt,
                        $xmlrpcString, $xmlrpcBase64, $xmlrpcArray));
$dropPicture_doc='dropPicture is used to deliver … ';
…
$a=array("dropPicture" => array(
           "function" => "wrapper_for_Augcws", // called function
           "signature" => $dropPicture_sig,    // used signature
           "docstring" => $dropPicture_doc),   // reported
documentation
           …
           );

     $s=new xmlrpc_server($a, false);          // create Service
     $s->service();                            // process Service
?>
```

Quelltext 7 Bereitstellung des XML-RPC Webservice

5.2.2 Daten speichern mit der Joomla API und dem Active Record Pattern

Sind die Daten nach dem Aufruf beim Webservice angekommen, ist es notwendig diese Daten in der Datenbank des CMS zu speichern. Das von der Joomla API verwendete Active Record Pattern stellt eine Basisklasse `mosDBTable` zur Verfügung. Von dieser Basisklasse wird nun eine eigene Klasse abgeleitet. Die Membervariablen dieser Klasse werden beim Aufruf von `mosDBTable::store()` in die gleichnamigen Felder der Datenbanktabelle übernommen. Somit ist es für die Methode des Webservices einfach die

empfangenen Daten zu speichern. Es ist nur notwendig eine Instanz der Datenklasse anzulegen, diese mit den Werten zu füllen und mit dem Aufruf der Methode `store()` zu speichern.

Zum Empfangen der Daten, die mit der Webservicemethode `dropPicture()` übergeben werden, wurde die Datenklasse auch mit einer Methode `dropPicture()` ausgestattet. Im folgenden Beispiel ist zu sehen wie die übergebenen Daten verarbeitet werden. Erst werden die Binärdaten in eine Datei geschrieben, danach werden die eigenen Membervariablen mit den Werten gefüllt und dann der Datensatz persistent in die Datenbank geschrieben.

```
// Class mosAugcws
public function dropPicture($yy, $mm, $dd, $h, $m,
                            $filename, $binary, $meta)
{
     global $database;
     $dst = $GLOBALS['mosConfig_absolute_path']
       .'/images/augcws/'.$filename;
     $fp = fopen($dst, 'w+');
     fwrite($fp, $binary);
     fclose($fp);
     // Eintrag in der Datenbank anlegen
     $this->id=null;
     $this->capturetime= sprintf('%04d-%02d-%02d %02d-%02d',
                                 $yy, $mm, $dd, $h, $m );
     $this->imagefile=$filename;
     // metadaten in array key=>value umbauen
     $this->meta = array();
     foreach ($meta as $key_val)
     {
           $this->meta[$key_val['key']]=$key_val['value'];
     }
     $this->published = true;
     $this->store();
     return self::ERR_NOERR;
}
```

Quelltext 8 Speichern der Daten in der Datenbank

5.2.3 Daten aus der Datenbank verwenden mit der Joomla API

Auch für das Auslesen von Daten aus der Datenbank stellt die Joomla API Methoden zur Verfügung. Gezeigt wird hier der Zugriff auf die aktuelle Konfiguration, die für den Client vorbereitet ist und vom Client mit der Webservicemethode `getConfig()` abgerufen wird. Hierzu wird die Anfrage mit dem Kriterium `published=1` an den MySQL Server formuliert. Liefert die Anfrage keine Ergebnisse liegt keine neue Konfiguration vor und es wird eine leere Zeichenkette zurückgeliefert. Wenn die Antwort einen Datensatz enthält, ist dieser erst mal vom Typ der Basisklasse `mosDBTable`. Um spezielle Operationen darauf auszuführen, wird das Objekt in ein Objekt des Typs der Datenklasse übertragen. Auf dieser Instanz können dann spezielle Operationen

der Datenklasse durchgeführt werden. In diesem Fall ist es konkret das Erzeugen der Konfiguration in XML Syntax.

```
 public function getConfig($id)
 {
     // haben wir eine Konfiguration zu senden?
     $query = 'SELECT * FROM #__augcws_configs '
              .' WHERE published="1" LIMIT 1 ';
     $this->_db->setQuery($query);
     $this->_db->query();
     $rows = $this->_db->loadObjectList();
     if ( count($rows) < 1)
     {
           // nichts zum publishen da
           return '';
     }
     else
     {
           $publish = new mosAugcwsConfigs($this->_db, $rows[0]);
           return $publish->getConfigXML();
     }
}
```

Quelltext 9 Lesen bestimmter Daten aus der Datenbank

5.2.4 Das Erzeugen der Konfiguration in XML Syntax

Das Erzeugen einer Clientkonfiguration in XML Syntax erfolgt durch die Methode mosAugcwsConfigs::getConfigXML(). Diese Methode liefert eine Zeichenkette mit der in XML formulierten Konfiguration. Um die XML Zeichenkette zu erzeugen wird eine Instanz der Klasse `XMLWriter` aus PHP verwendet. Mit der Klasse werden einfach sequentiell die Elemente und Attribute hinzugefügt und am Ende die fertige XML zeichenkette ausgegeben. Vorerst werden die Konfigurationsattribute der Clientanwendung ausgegeben, dann die Werte der Zeittabelle.

```
private function xmlAddKeyValue( &$xml, $key, $value)
{
$xml->startElement('add');
$xml->writeAttribute('key', $key);
$xml->writeAttribute('value', $value);
$xml->endElement();
}
public function getConfigXML()
{
$meta=($this->meta!=''?unserialize($this->meta):array());
$xw = new XMLWriter();
     $xw->openMemory();
     $xw->startDocument('1.0', 'utf-8'); // start document
     $xw->startElement("configuration");
     $xw->writeAttribute('id', $this->id);
     $xw->startElement("appSettings");
     $this->xmlAddKeyValue($xw, 'controlPower',
                           $meta['controlPower']);
     $this->xmlAddKeyValue($xw, 'controlGSM',   $meta['controlGSM']);
     $this->xmlAddKeyValue($xw, 'controlGPRS',
                           $meta['controlGPRS']);
```

```
…
      $xw->endElement();//appSettings
      $xw->startElement("scedule");
      for ($day=0; $day<7;$day++)
      {
            $xw->startElement('day');
            $xw->writeAttribute('id', $day);
            for($hour=0; $hour<24; $hour++)
            {
                  $xw->startElement('hour');
                  $xw->writeAttribute('id', $hour);
                  $xw->writeAttribute('value',
                  $meta['sced'][$day][$hour]);
                  $xw->endElement();
            }
            $xw->endElement();
      }
      $xw->endElement();//scedule
      $xw->endElement();//configuration
      $xw->endDocument(); // end document
      return $xw->outputMemory(); // output of generated xml
 }
```

Quelltext 10 Erzeugung der XML-Konfigurationsdatei

5.2.5 Inhalte aus der Komponente heraus anzeigen

Der Einstiegspunkt für die Joomla API um eine Komponente anzuzeigen, ist immer die gleichnamige PHP Datei. In diesem Fall ist das die Datei augcws.php, die von Joomla aufgerufen wird, wenn ein Anwender auf den entsprechenden Menupunkt in der Navigation geklickt hat.

Prinzipiell ist man von dieser Einstiegsstelle frei in der Architektur der eigenen Anwendung. Alles was in PHP geht kann gemacht werden. Ich halte mich jedoch an das Beispiel zum Erstellen einer Komponente aus (16). Überzeugend ist an dem Beispiel die Trennung von Anwendungslogik und Präsentation der Anzeige. So gibt es zusätzlich zur Datei der Komponente noch eine augcws.html.php Datei. In dieser Datei sind die verschiedenen Ansichten auf die Dateien implementiert. Der überwiegende Teil der Datei besteht aus HTML, welches direkt zur Anzeige kommt. Die PHP Anteile sind ausschließlich Logik zur Anzeige der Daten, z.B. Schleifen über Datenarrays mit sich wiederholender Darstellung.

Direkt nach dem Einlesen werden die für die Auswahl der Ansicht und der anzuzeigenden Daten relevanten Parameter eingelesen.

Anhand des Parameters `task` wird entschieden, ob eine einzelne Aufnahme oder die Aufnahmen eines ganzen Tages angezeigt werden sollen. Im Anschluss werden die Daten anhand der Kriterien aus der Datenbank ausgelesen. Sind keine Kriterien angegeben wird die letzte aktuelle Aufnahme als Einzelansicht angezeigt.

Die ermittelten Daten werden dann an die Methoden aus der `augcws.html.php` zur Anzeige übergeben und zum Anwender übertragen.

5.2.6 Konfiguration und Verwaltung im Backend

Der Zugriff auf den Backendteil der Komponente durch Joomla verhält sich analog zum Zugriff auf die Komponente aus Abschnitt 5.2.5. Den Dateinamen wird lediglich ein Präfix `admin.` vorangestellt.

Auch hier wird in der admni.augcws.php die Anwendungslogik abgebildet. Anders als bei der Ansicht für die Anwender wird hier noch anhand des Parameters `act` zwischen den beiden Hauptaufgaben Inhalt und Konfigurationen unterschieden. Im Anschluss wird die gewählte Aktion anhand des Parametes `task` ausgewählt. Die Aktionen erzeugen, verändern, löschen die Daten oder lesen sie aus der Datenbank aus. Im Anschluss an die Anwendungslogik wird eine der drei Anzeigevarianten aus `admin.augcws.html.php` aufgerufen um die Anzeige im Backend auszugeben. Die möglichen Anzeigen sind im Folgenden abgebildet.

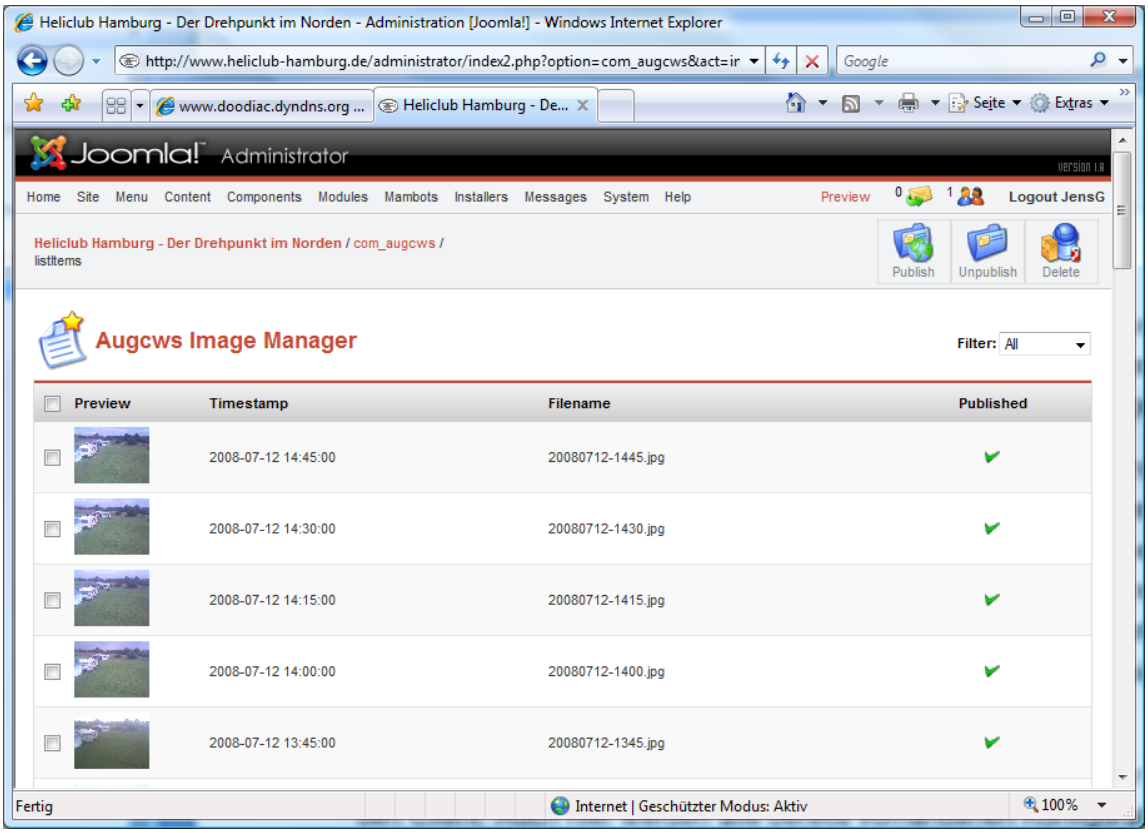

Abbildung 5.3 Backend Listenansicht der Inhalte

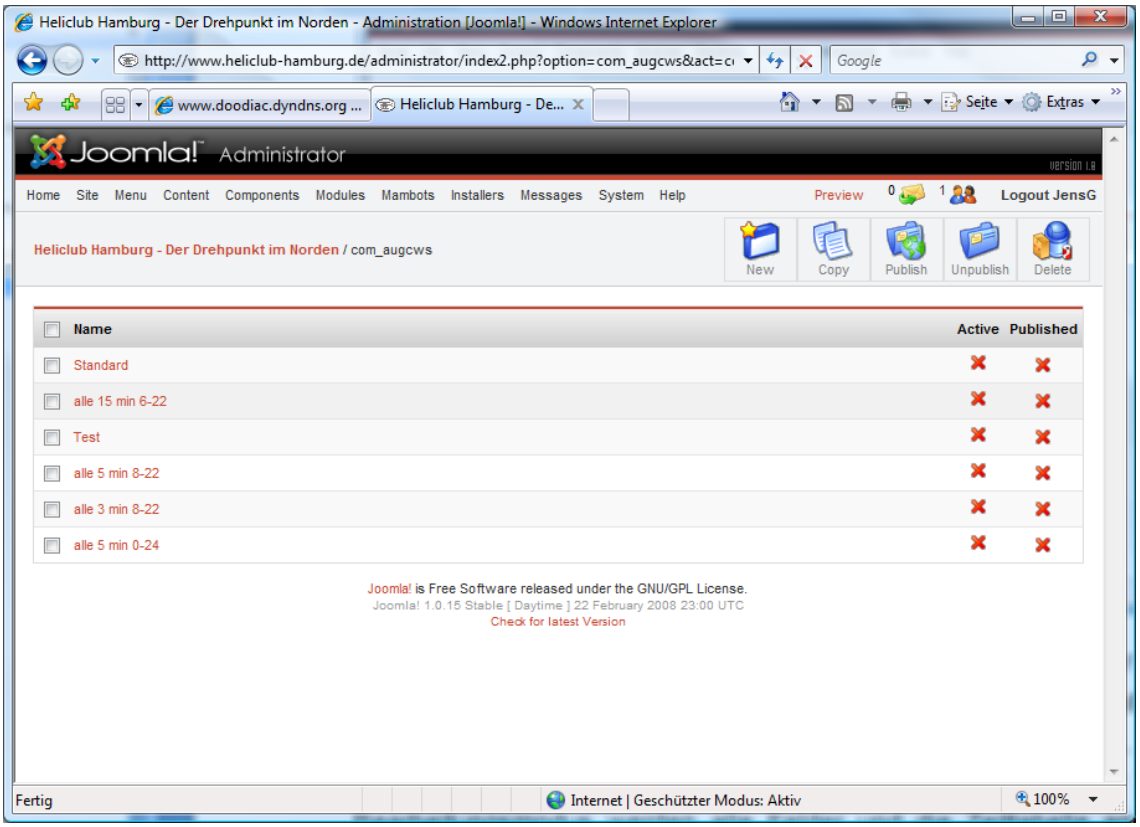

Abbildung 5.4 Backend Listenansicht der Clientkonfigurationen

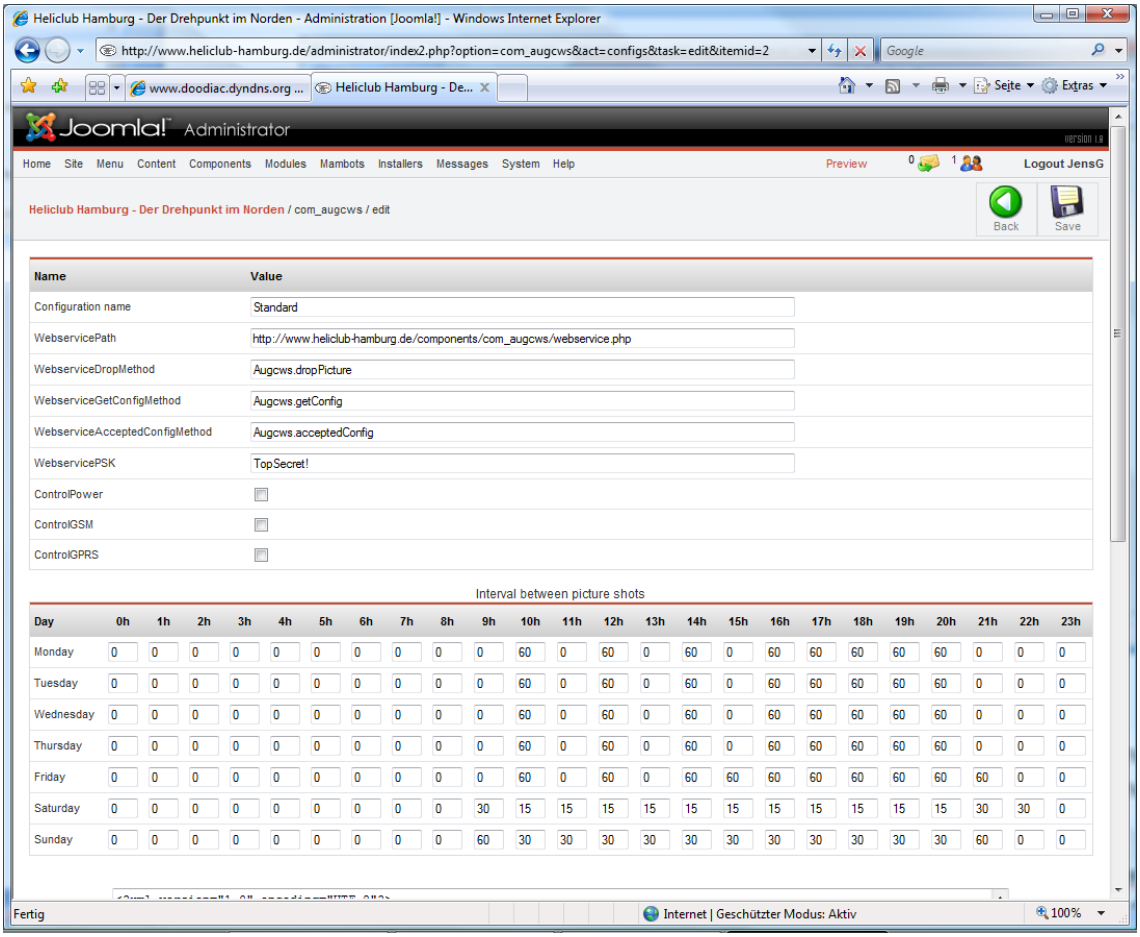

Abbildung 5.5 Backend Bearbeitungsansicht einer Clientkonfiguration

5.3 Tests und Resultate

Es ist leider nicht ganz einfach geeignete Testverfahren und Szenarien zu entwickeln, bei denen das angestrebten Autonomieverhalten und die angestrebte Robustheit gegen Störungen auf lange Zeit überprüft werden können.

Mein Test des Prototyps bestand im mehrwöchigen Betrieb unter verschiedenen Bedingungen. Es wurden verschiedene GSM Anbieter verwendet. Es wurde mit ständiger, wechselnder und ohne Energieversorgung getestet. Es wurden unterschiedliche Zeitpläne und Konfigurationen zum Gerät übertragen und getestet. Einen Feldversuch hat es gegeben. Bei diesem ersten Außeneisatz des Prototyps wurde er in einer wasserdichten Silikontasche mit Klebeband am zukünftigen Zielort betrieben. Leider stellte der Client die Funktion nach nur 1,5 Tagen ein. Vermutungen nach ist durch die versuchsweise Montage Druck auf die Bedienelemente ausgeübt worden und die Anwendung quasi durch Benutzereingabe beendet worden. Die endgültige Montage sollte das Problem nicht erzeugen. Auch ist über eine Sicherung gegen unbeabsichtigtes Beenden nachzudenken.

Ein weiteres Phänomen konnte beobachte werden. Wenn man, während der Übertragung der Daten, das Gerät über das GSM-Netz anruft kommt es gelegentlich zu einer Ausnahme, die nicht abgefangen wird in der TAPI Implementierung (20). Dieses Problem muss noch untersucht werden und die fehlerhaften evtl. nicht einmal benötigten Programmteile müssen repariert oder entfernt werden.

Um langfristig ein stabiles und wartungsfreies Verhalten zu bekommen sollte der Client noch eine Watchdog[29] Funktion bekommen. Eine Software, die ebenfalls auf dem Client läuft und den Status der Clientanwendung überwacht, diese ggf. beendet und neu startet oder das ganze Gerät neu startet.

[29] **Watchdog** ist die engl. Bezeichnung für Wachhund. In der Computertechnik wird damit eine Komponente beschrieben, die ein andere Komponente oder deren Funktion überwacht.

6 Zusammenfassung

Hier soll der Inhalt der Arbeit kurz zusammengefasst werden, ein Fazit über die Arbeit gezogen werden und ein Ausblick auf mögliche Weiterentwicklung gegeben werden.

6.1 Zusammenfassung

Ziel dieser Arbeit war es den Mitgliedern des Heli-Club Hamburg e.V. eine Lösung zu schaffen, mit der es möglich ist einen halbwegs aktuellen Eindruck der Situation auf dem Flugplatz auf der Homepage darzustellen. Es sollte eine Lösung entwickelt werden, die autonom und zuverlässig arbeitet und bei Bedarf aus der Ferne zu konfigurieren ist.

Zuerst wurden die Situation und die Anforderung geschildert. Das Problem lag vor allem in der Abgeschiedenheit des Flugplatzes. Durch die Abgeschiedenheit sind am Flugplatz keine feste Energieversorgung und Internetanbindung vorhanden. Es galt also eine Lösung zu finden, die mit wenig Energie aus einem Solargenerator und der Internetanbindung über Mobilfunk auskommt.

Nachdem die Technologien hinsichtlich der Anforderungen ausgewählt waren, wurde ein Design entwickelt, mit dem es möglich ist regelmäßig Bilder zu erstellen und auf der Homepage zu veröffentlichen. In diesem Design wurden die Teilsysteme, und deren Schnittstellen zueinander, aus dem dargestellten Szenario benannt und einzeln entwickelt.

Bei der Beispielimplementierung wurden die im Design entwickelten Teilsysteme und Designangaben umgesetzt. Es wurde eine Anwendung für den Client entwickelt, die regelmäßig den Inhalt erfasst und über die Schnittstelle zum Webserver überträgt. Für den Webserver wurde eine Komponente für das CMS Joomla erstellt, die einen Webservice als Schnittstelle für den Client bereitstellt, die Inhalte in der Datenbank speichert und über verschiedene Ansichten für den Anwender verfügt.

6.2 Fazit

Die Realisierung der Beispielimplementierung hat letztendlich gezeigt, dass es möglich ist bei durchdachter Wahl der Technologien eine kleine und Ressourcen schonende Clientlösung zu realisieren. Die Lösung ist sowohl sparsam im Umgang mit der zur Verfügung stehenden Energie, als auch bei der Nutzung der Internetverbindung im Umgang mit dem Übertragungsvolumen.

Die erstellte Komponente für das CMS des Webservers ist zwar eine individuelle Lösung, genau auf die Anforderungen zugeschnitten. Jedoch ist mir am Ende aufgefallen, dass man für die derzeitige Funktion auch eine der vielen fertigen Bildergalerien- und Fotoalbenkomponenten zur Verwaltung und Anzeige der Inhalte hätte verwenden können. Die eigene Komponente hätte den Inhalt auch per Webservice entgegennehmen können, dann jedoch in ein fertiges Fotoalbum eingepflegt. Man hätte etwas mehr Aufwand und Design beim Anpassen und Einpflegen der Inhalte in die Struktur der Fotoalbenkomponente betreiben müssen, sich damit aber das Erstellen der Ansichten auf die Inhalte sparen können.

Aus autodidaktischer Sicht war es für mich allerdings sinnvoll auch das Erstellen von Ansichten in einer Komponente für das CMS Joomla zu bearbeiten.

6.3 Ausblick

Bei zukünftig weiter fallenden Preisen für mobile Internetanbindungen gepaart mit zunehmend höheren Übertragungsraten ist ein Erhöhen der Bildfrequenz bis hin zum Live-Bild mit mehreren Bildern pro Sekunde denkbar. Bei gleich bleibender Internetanbindung ist kurzfristig die Erweiterung des Systems um die Erfassung von Umweltdaten wie Windrichtung und Windgeschwindigkeit, Regensensor und Temperatur über externe Sensoren ein interessantes Szenario. Derzeit müssen alle diese Informationen vom Anwender selber aus dem Bild interpretiert werden, was sicher auch schon eine Verbesserung darstellt, aber noch großen Schwankungen unterliegt.

Literaturverzeichnis

1. **Deutscher Bundestag.** *Kunst Urheber Gesetz (KunstUhG).* Deutschland : s.n., 2001.

2. **Stafford-Fraser, Quentin.** *The Trojan Room Coffee Pot Resources.* [Online] [Zitat vom: 16. 07 2008.] http://www.cl.cam.ac.uk/coffee/qsf/index.html.

3. **Müller, Volker.** Financial Times Deutschland. *Dossier Nokia gibt Betriebssystem Symbian frei.* [Online] 26. 06 2008. [Zitat vom: 18. 07 2008.] http://www.ftd.de/technik/it_telekommunikation/:Kampf%20Google%20Nokia%20Betriebssystem%20Symbian/377479.html.

4. **Symbian Software Ltd.** *Symbian OS.* [Online] Symbian Software Ltd, 2008. [Zitat vom: 18. 07 2008.] http://www.symbian.com/files/rx/file8929.pdf.

5. **Symbian Software Ltd.** *Symbian Developer Network.* [Online] Symbian Software Ltd, 2008. [Zitat vom: 10. 07 2008.] http://developer.symbian.com.

6. **Microsoft Corporation.** MSDN. *Microsoft Developer Network.* [Online] Microsoft Corporation, 2008. [Zitat vom: 12. 07 2008.] http://msdn.microsoft.com.

7. —. *Windows Mobile 6.* [Online] Microsoft Corporation, 25. 3 2008. [Zitat vom: 18. 07 2008.] http://msdn.microsoft.com/en-us/library/bb158486.aspx.

8. —. MSDN. *Mobile and Embedded Development.* [Online] Microsoft Corporation, 2008. http://msdn.microsoft.com/en-us/library/ms376734.aspx.

9. **Esmertec AG.** *Jbed Java Phone Engine.* [Online] Esmertec AG, 2008. [Zitat vom: 18. 07 2008.] http://www.esmertec.com/40.html.

10. **Grabowski, Rudolf.** *Solarstromerzeugung.de.* [Online] 2008. [Zitat vom: 27. 07 2008.] http://www.solarstromerzeugung.de.

11. **SCHOTT Solar GmbH.** *Schott Solar GmbH.* [Online] 2008. [Zitat vom: 20. 06 2008.] http://www.schott.com/photovoltaic/german/.

12. **Petzold, Charles.** *Windows-Programmierung: Das Entwicklerhandbuch zur WIN32-API.* D-85716 Unterschleißheim : Microsoft Press Deutschland, 2000. 3-86063-188-8.

13. **Gamma, Erich; et al.** *Designpattern.* USA : Addison-Wesley Publishing Company, 1995.

14. **Netcraft Ltd.** *Market Share for Top Servers Across All Domains August 1995 - June 2008.* [Online] Netcraft Ltd., 06 2008. [Zitat vom: 19. 07 2008.] http://news.netcraft.com/archives/web_server_survey.html.

15. **Herold, Helmut; Klar, Michael und Klar, Susanne.** *Objekt-orientierung.* München : Addison Wesley Verlag, 2001. ISBN 3-8273-1651-0.

16. **Ebersbach, Anja; Glaser, Markus und Kubani, Radovan.** *Joomla! Das Handbuch für Einsteiger.* Bonn : Galileo Press, 2006. ISBN 3-89842-632-7.

17. **Postel, J. und Reynolds, J.** *File Transfer Protocol (FTP), RFC959.* [Online] 10 1985.

18. **Franks, J.; et al.** *HTTP Authentication: Basic and Digest Access Authentication, RFC2616.* [Online] IETF, 06 1999.

19. **Salzer, Mario.** *XML+RPC - XML marshalled Remote Procedure Calls, Draft.* [Online] IETF, 07 2004.

20. **Feinman, Alex.** *TAPI 1.1 Download.* [Online] 2003. [Zitat vom: 04. 06 2008.] http://www.alexfeinman.com/download.asp?doc=tapi1.1.zip.

21. **Cook, Charles;.** *XML-RPC.NET.* [Online] Charles Cook, 04 2008. [Zitat vom: 19. 06 2008.] http://www.xml-rpc.net/.

22. **Schlossnagle, George.** *Professionelle PHP5 Programmierung.* München : Addison-Wesley Verlag, 2005. 3-8273-2198-0.

23. **Oekosoft.** *Oekosoft.* [Online] Oekosoft, 2007. [Zitat vom: 08. 06 01.] http://www.oekosoft.ch/.

24. **O'Reilly, Tim.** *What Is Web 2.0?* [Online] 30. 09 2005. [Zitat vom: 16. 07 2008.] http://www.oreilly.de/artikel/web20.html.

25. **Berners-Lee, T.; Fielding, R. und Masinter, L.** *Uniform Resource Identifier (URI): Generic Syntax, RFC3986.* [Online] IETF, 01 2005.

26. **Postel, Jon.** *Transmission Control Protocol (TCP), RFC793.* [Online] IETF, 09 1981.

27. **Josefsson, S.** *The Base16, Base32, and Base64 Data Encodings, RFC4648.* [Online] IETF, 10 2006.

28. **Postel, Jonathan B.** Simple Mail Transfer Protocol (SMTP), RFC812. [Online] IETF, 03 1982.

29. **Myers, J.; Mellon, Carnegie und Rose, M.** *Post Ofoofice Protocol - Version 3 (POP3), RFC1939.* [Online] IETF, 05 1996.

30. **The PHP Group.** *PHP: Hypertext Preprocessor.* [Online] The PHP Group, 2008. [Zitat vom: 16. 07 2008.] http://www.php.net.

31. **Joomla.org.** *Joomla.* [Online] 2008. [Zitat vom: 16. 07 2008.] http://www.joomla.org/.

32. **Postel, Jon.** *Internet Protocol (IP), RFC791.* [Online] IETF, 09 1981.

33. **Fielding, R.; et al.** *Hypertext Transfer Protocol -- HTTP/1.1, RFC2616.* [Online] IETF, 06 1999.

34. **MySQL AB.** http://www.mysql.com/. [Online] MySQL AB, 2008. [Zitat vom: 16. 07 2008.] http://www.mysql.com/.

Anhang

A1: Quellcode auf CD

Bei der gedruckten Version der Arbeit befindet sich an dieser Stelle der Datenträger mit den Quellcodes für die Beispielimplementierung der Clientanwendung und der Webserverkomponente.

Sollte der Datenträger nicht vorhanden sein, oder diese Arbeit nur online vorliegen kann der Inhalt unter

http://www.gabrikowski.de/diplomarbeit/

heruntergeladen werden.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 08. August 2008

Ort, Datum	Unterschrift