Bachelor-Thesis

Applikationsentwicklung auf dem mobilen Android System zur Interaktion mit Prozessdaten auf einer Siemens SIMATIC SPS

Sascha Moecker
Matr. Nr. 522490

betreut durch
Prof. Dr. Wolfgang Lux (FH Düsseldorf)
Dr.-Ing. Axel Buch (Siemens AG Köln)

FH Düsseldorf
Abgabedatum: 13. August 2012

Fachhochschule Düsseldorf
University of Applied Sciences

SIEMENS

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Bachelor-Thesis selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Köln, den 13.08.2012 _____

(Sascha Moecker)

Kontakt

sascha.moecker@live.de

S

FH D
Fachhochschule Düsseldorf
University of Applied Sciences

Abstract

Deutsch

In dieser Bachelor-Thesis werden die Möglichkeiten eines Remote-Zugriffs auf die Prozessdaten einer Siemens SIMATIC speicherprogrammierbaren Steuerung (SPS) untersucht. Der Schwerpunkt liegt auf der Interaktion mit Prozessdaten über ein mobiles Android-Gerät unter Nutzung der WLAN-Schnittstelle. Als praktischer Teil der Thesis steht daher die Entwicklung einer Android-Applikation zum Erstellen, Editieren und Ausführen grafischer Oberflächen, welche Prozessdaten anschaulich über Anzeigeelemente darstellen und eine Interaktion mit dem Anwender über Bedienelemente auf dem Bildschirm des mobilen Geräts ermöglichen.

Hierzu geht die Thesis zu Beginn auf die grundlegenden Strukturen eines SIMATIC Automatisierungssystems hinsichtlich Hardware- und Softwarekomponenten sowie auf bestehende Remote-Zugriffsmöglichkeiten ein. Diese werden mit der App-Lösung verglichen und Vor- und Nachteile evaluiert. Anschließend werden die verwendete Kommunikationsbibliothek *Libnodave* und ein Abriss über das Android-Framework gegeben. Diese Technologien bilden die Grundlage zur Entwicklung eigener Applikationen zur Interaktion mit der SIMATIC Automatisierungshardware.

Mit der Untersuchung des Softwareentwicklungsprozesses und der Funktionsbeschreibung der entwickelten Applikation *AndroHmiS7* wird die Anwendung der erarbeiteten Themen gegeben.

Englisch

In this bachelor thesis the possibilities of a remote access to the process data of a Siemens SIMATIC programmable logic control (PLC) are examined. The main emphasis is on the interaction with process data over a mobile Android device under the use of the WLAN interface. The practical part of the thesis is therefore the development of an Android application for creating, editing and executing a graphical surface. This surface represents process data descriptive via display elements and makes an interaction with the user possible over the screen of the mobile device via operating elements.

To this, the thesis initially goes into the basic structures of a SIMATIC automation system with regard to hardware and software items as well as to existing remote access options. These are compared to each other and advantages and disadvantages are evaluated with the App-solution. The used communication library *Libnodave* and a survey about the Android framework are given. These technologies form the basis for the development of individual applications for the interaction with SIMATIC automation hardware.

With the examination of the software development process and the functional description of the developed application *AndroHmiS7* an implementation of the topics worked out in this thesis is explained.

Inhaltsverzeichnis

1 Einleitung

1.1 Zum Thema dieser Bachelor-Thesis

Remote Zugriff auf Prozessdaten

Die vorliegende Bachelor-Thesis stellt die innerhalb von 12 Wochen erstellte Abschlussarbeit meines sechssemestrigen Bachelorstudiengangs der Kommunikations- und Informationstechnik mit Fachrichtung Automatisierungstechnik an der Fachhochschule Düsseldorf dar. Sie beleuchtet das Feld des Remote-Zugriffs auf Prozessdaten einer Siemens SIMATIC SPS (Speicherprogrammierbare Steuerung) über ein mobiles Android-Gerät unter Nutzung der WLAN-Schnittstelle.

Zu diesem Zweck wurde die Applikation *AndroHmiS7* entwickelt, welche mit Hilfe der *Libnodave* Kommunikationsbibliothek eine Interaktion mit Prozessdaten ermöglicht und diese ansprechend über Anzeigeelemente auf dem Display darstellt und eine Zugriffsmöglichkeit über Bedienelemente bereitstellt.

1.2 Motivation und Idee

Automatisierungsanlagen überwachen

Speicherprogrammierbare Steuerungen finden in vielen Bereichen Einsatz. Von der einfachen Hausinstallation über die Steuerung von Werkzeugmaschinen bis hin zu komplexen, vernetzten Anlagen werden SPSen verwendet. Hier haben sich zur Steuerung und Überwachung der Prozesse HMI (Human Machine Interface) durchgesetzt, welche Prozessdaten visuell aufgearbeitet darstellen und eine Interaktion mit dem Bediener ermöglichen. Diese festinstallierten Bedienterminals sind zumeist standortgebunden und nicht mobil einsetzbar. Hieraus entwickelte sich die Idee, für Diagnosezwecke, Wartungsarbeiten, Inbetriebnahmen oder die Erfassung von Status- und Fehlermeldungen sowie wichtigen Statuswerten einer Anlage oder SPS, eine Möglichkeit des Remote-Zugriffs über ein mobiles Gerät zu ermöglichen. Die Idee erhebt dabei nicht den Anspruch, einen vollwertigen HMI Ersatz darzustellen, sondern soll vielmehr eine einfache und mobile Ergänzung zusätzlich anzubieten.

Leistungsfähige Android-Geräte

„Die jüngsten Entwicklungen im Bereich Computer-Hardware legen den Fokus auf die Entwicklung von mobilen Geräten. Lag vor einigen Jahren der Schwerpunkt noch auf Desktop Computern und kompakteren Notebooks, so sind es heute die Tablet Computer oder Smartphones, welche durch Ihre leistungsstarken Hard- und Softwarekomponenten schon an die Leistungsfähigkeit eines klassischen Desktop Computers heranreichen.

Dieser Entwicklung entsprechend eröffnen sich breite Möglichkeiten in der Nutzung dieser mobilen Geräte, sei es aufgrund der allgegenwärtigen

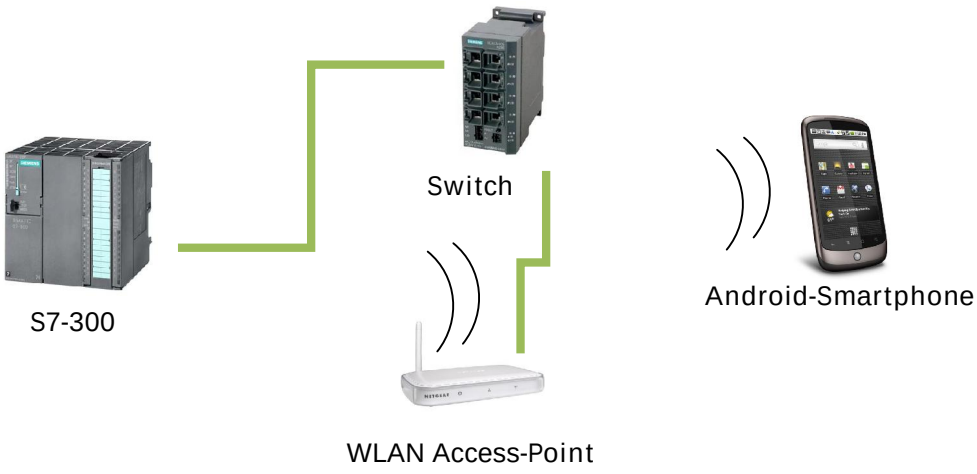
Verfügbarkeit, der kompakten Bauart oder der benutzerfreundlichen Ein- und Ausgabemöglichkeiten auf zumeist berührungsempfindlichen Bildschirmen. Dieses sich rasch entwickelnde Feld mit der ehre konservativen Automatisierungstechnik zu koppeln, galt mir dabei als besondere Motivaton und Herausforderung“ (Moecker, 2012 S. 6).

Interaktion mit Prozessdaten über ein mobiles Gerät und WLAN

Die Applikation *AndroHmiS7* soll eine einfache und schnell zu implementierende Möglichkeit bieten, mittels eines Android-Geräts über WLAN auf die Prozessdaten einer SIMATIC SPS zuzugreifen. Prozessdaten sind im Allgemeinen analoge und digitale Werte, welche aus einem technischen Prozess gewonnen werden. Im Rahmen dieser Bachelor-Thesis wird der Begriff auf jegliche interne Repräsentation von Variablen erweitert. Ähnlich eines professionellen HMI, werden diese Daten über verschiedene Steuer- und Anzeigeelemente dem Benutzer zugänglich gemacht. Als unterstütze Controller werden die klassischen SPS S7-300/400 sowie die Soft-SPS WinAC (Windows Automation Center) angestrebt.

Um eine Verbindung vom Android-Gerät auf die SPS herzustellen, sollen nur wenige Schritte notwendig sein. Die SPS muss sich in einem Ethernet-Netz befinden, welches über einen WLAN Access-Point für die verbindungslose Kommunikation zugänglich gemacht wird. Über die IP-Adresse des Kommunikationsmoduls der SPS kann dann mit der Applikation eine Verbindung hergestellt werden. Der Zugriff auf die Prozessdaten erfolgt anschließend zyklisch.

Abbildung 1-1 Verbindung zwischen einer S7-300 und einem Android-Smartphone



1.3 Vorgehen und Ziele

Aufbau dieser Bachelor-Thesis

Ausgehend vom Kapitel 2, welches die Grundlagen eines Siemens SIMATIC Automatisierungssystems hinsichtlich Hard- und Softwarekomponenten aufzeigt, wird in Kapitel 3 auf bestehende Möglichkeiten des Remote-Zugriffs auf die Prozessdaten einer SPS eingegangen.

Eine offene Schnittstelle zum Zugriff auf diese Daten bietet die Kommunikationsbibliothek *Libnodave*, welche in Kapitel 4 in ihrer Funktion und Anwendung erläutert wird. Sie bildet das Bindeglied zwischen SIMATIC Komponenten und eigener Software. Im Rahmen der Bachelor-Thesis wird diese Bibliothek als Bindeglied zwischen einer SIMATIC SPS und der Android-Applikation verwendet.

Hierfür wird im folgenden Kapitel 5 das Android-Framework untersucht. Neben der grundlegenden Systemarchitektur, wird auch auf die Nutzung des SDK (Software Development Kit) zur Erstellung eigener Applikationen eingegangen. Zur komfortablen Entwicklung der Softwarekomponenten (Portierung der *Libnodave* Bibliothek und Programmierung der Applikation) werden zudem genutzte Entwicklungsumgebungen vorgestellt.

Schließlich finden die evaluierten Ergebnisse und gemachten Erfahrungen im Kapitel 6 – dem Entwicklungsprozess der Applikation *AndroHmiS7* Anwendung. Hier werden im Rahmen des Software-Engineerings alle Phasen der Entwicklung vorgestellt und speziell auf die realisierte Implementierung näher eingegangen.

Im Fazit (Kapitel 7) werden abschließend die erreichten Ziele und gemachten Erfahrungen noch einmal zusammengefasst.

Ziele

Alle evaluierten Ergebnisse der untersuchten Technologien dienen als Grundlage zur Entwicklung der Applikation *AndroHmiS7*.

Das Ziel dieser Bachelor-Thesis ist die Erläuterung der Auswahlkriterien und Funktionsweise gewählter Komponenten, welche zur Realisierung der Applikation beitragen. Zudem soll das Vorgehen bei der Entwicklung der Applikation nachvollziehbar dargestellt werden.

Als Ziel der Applikationsentwicklung steht ein lauffähiger, stabiler, zu allen Android-Versionen kompatibler und gut dokumentierter Softwareprototyp zur performanten Interaktion mit Prozessdaten einer Siemens SIMATIC SPS. Dieser soll in seinem Entwicklungsprozess nachvollziehbar, modular und erweiterbar ausgelegt sein.

2 Siemens SIMATIC Automatisierungssystem

Übersicht über das SIMATIC System

Das Siemens SIMATIC System umfasst eine Vielzahl von Komponenten für die unterschiedlichsten Anwendungen der Prozess- und Fertigungsautomatisierung. Die Prozessautomation umfasst die Automation kontinuierlicher und vernetzte Prozesse wie z.B. die Papierherstellung; die Fertigungsautomation zielt auf wiederkehrende Arbeitsabläufe ab wie z.B. bei der Automobilproduktion.

Es beinhaltet Hard- und Softwarelösungen zur Bewältigung aller Automatisierungsaufgaben innerhalb des sogenannten Totally Integrated Automation (TIA) Konzepts. Neben den reinen Steuerungen (SIMATIC Controller), bietet das System unter anderem Produkte für die industrielle Kommunikation (SIMATIC NET) und eine Reihe von Bediengeräten und Software zur Visualisierung (SIMATIC HMI).

(Berger, 2000 S. 10)

2.1 Hardwarekomponenten

2.1.1 Speicherprogrammierbare Steuerungen (SPS)

Verdrängung der VPS durch SPS

Die speicherprogrammierbaren Steuerungen wurden Ende der 60er Jahre in den USA entwickelt und sollten die bis dahin dominierenden Verbindungsprogrammierten Relais-, Schütz- und Elektroniksteuerungen ablösen. Bei den Verbindungsprogrammierten Steuerungen (VPS) ist das ausgeführte Programm durch fest installierte Draht-, Leiterplatten oder Kabelverbindungen festgelegt und ist deshalb für jede Aufgabenstellung individuell zu erstellen. Eine SPS hingegen legt die Logik im Speicher ab und sein Programm kann jederzeit geändert werden. Durch die rasche Entwicklung der Halbleitertechnologie erreichte diese Steuerung Ende der 90er Jahren einen Anteil von fast 100% am Automatisierungsmarkt. Lediglich PC-basierte Automatisierungssysteme (SIMATIC WinAC) konnten insbesondere in Bereichen mit hohem Rechen- und Visualisierungsaufwand sowie der Forderung nach Datenweiterverarbeitung mithalten bzw. setzen sich heutzutage immer weiter durch.

(Gevatter H.J., 2006 S. 476f)

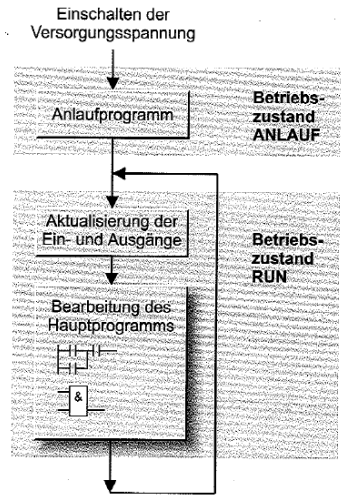
Aufbau und Zyklische Abarbeitung

Eine speicherprogrammierbare Steuerung besteht im einfachsten Fall aus einer Stromversorgung, einer CPU und digitalen Eingabe- und Ausgabebaugruppen. Das Herzstück einer SPS ist dabei die CPU. Diese bearbeitet die Anweisungen des im Arbeitsspeicher abgelegten Programms seriell ab und fängt beim Erreichen des Programmendes wieder von vorne an. Man spricht hier von einer zyklischen

Programmabarbeitung. Zu Beginn eines Zyklus werden die Eingangssignale in einem dafür reservierten Speicherbereich, dem sogenannten Prozessabbild der Eingänge, eingelesen. Ausgehend von den Anweisungen des Programms wird am Ende des Zyklus das Prozessabbild der Ausgänge bestimmt und zu den Ausgabebaugruppen übermittelt.

(Berger, 2000 S. 15)

Abbildung 2-1 Zyklische Abarbeitung des SPS-Programms



Echtzeitfähigkeit

Automatisierungssysteme müssen echtzeitfähige Systeme sein. Innerhalb der DIN 44300 wird der Begriff der Echtzeit definiert als der „Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen“ (Langmann, 1996).

Hieraus lässt sich bereits ein Grundgedanke der Automatisierungstechnik erkennen, in dem Performance und Leistungsfähigkeit zwar wichtig, aber Verfügbarkeit, Vorhersehbarkeit und Ausfallsicherheit die entscheidenden Merkmale sind.

Speicherbereiche

Eine SPS bietet verschiedene Speicherbereiche für jeweilig definierte Aufgaben an. Diese befinden sich im Systemspeicher der CPU. Neben Ein- und Ausgängen über Digital- oder Analogbaugruppen, spielen Merker als globale, temporäre oder remanente Informationsspeicher zur kurzen Zwischenspeicherung von Werten eine wichtige Rolle. Für strukturierte Daten empfiehlt sich die Speicherung innerhalb eines Datenbausteins.

Tabelle 2-1 Speicherbereiche einer SIMATIC SPS und deren Verwendung

Speicherbreich	Verwendung
Merker (M)	Globale Informationsspeicher – entweder temporär bestehend bis zum Neustart oder remanent bestehend bis zum Urlöschen
Eingänge (E)	Prozessabbild der Digital- und Analogeingabebaugruppen
Ausgänge (A)	Prozessabbild der Digital- und Analogausgabebaugruppen
Zeitfunktionen (T)	Zeitglieder für Warte- oder Überwachungszeiten
Zählfunktionen (Z)	Softwarezähler für vorwärts und rückwärts zählen
Datenbausteine (DB)	Strukturierte Speicherbereich - entweder innerhalb eines Global-DB oder Instanz-DB eines Funktionsbausteins (FB)

(Berger, 2000 S. 38)

Betriebsarten einer SPS

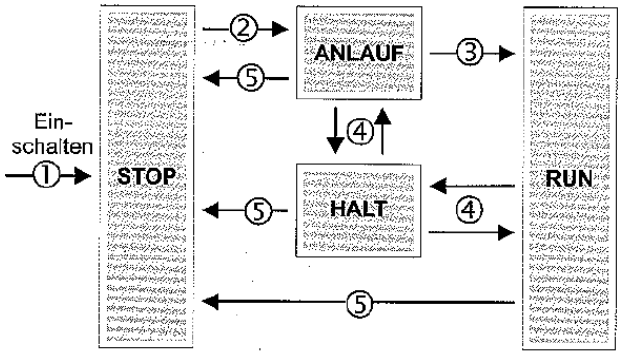
Die Betriebsart einer SPS kann vereinfacht in die Modi

- RUN und
- STOP

eingeteilt werden. Im RUN Zustand wird das Programm zyklisch abgearbeitet. Im STOP Zustand wird die Abarbeitung angehalten, ein Zugriff auf remanente Merker und Datenbausteine ist aber nach wie vor möglich. Zwischen beiden Modi erfolgt ein Übergang, welcher unterschieden wird in die Arten

- Warmstart und
- Kaltstart.

Abbildung 2-2 Betriebsarten und -übergänge einer SIMATIC CPU



Nach dem Einschalten ① befindet sich die CPU im Zustand STOP. Dreht man den Schlüsselschalter nach RUN oder RUN-P, wird zuerst ein Anlaufprogramm durchlaufen ② bevor die eigentliche Bearbeitung des Anwenderprogramms im RUN beginnt ③. Im ANLAUF und im RUN können Testfunktionen durchgeführt werden, die in den Betriebszustand HALT führen ④. Tritt in einem dieser Betriebszustände ein Fehler auf, fällt die CPU in den Zustand STOP zurück ⑤.

(Berger, 2000 S. 15)

2.1.2 SIMATIC Controller

Übersicht

Das Automatisierungssystem SIMATIC S7-300/400 ist modular aufgebaut und besteht aus den folgend genannten, wesentlichen Komponenten.

- *Baugruppenträger (Racks*) werden zur Aufnahme und Verbindung der Baugruppen über eine Profilschiene mit Rückwandbus benutzt

- *Stromversorgung (PS)* liefert die interne Versorgungspannung von zumeist 24V DC
- *Zentralbaugruppe (CPU)* dient zur Speicherung und Ausführung des Anwenderprogramms, Parametrierung der Baugruppen sowie zur Kommunikation mit dem Programmiergerät (PG), Baugruppen und anderen Stationen
- *Signalbaugruppen (SM)* stellen die Schnittstelle zum Prozess dar. Damit können digitale oder analoge Aktoren und Sensoren angeschlossen werden
- *Funktionsbaugruppen (FM)* bearbeiten komplexe und zeitkritische Prozesse und entlasten dadurch die CPU
- *Kommunikationsbaugruppen (CP)* dienen zum Anschluss des Systems an verschiedenen Kommunikationsnetze

(Berger, 2002 S. 22)

Erläuterung ausgewählter Controller

S7-300: Die modulare Kleinsteuerung S7-300 ist der meistverbreitete Controller der SIMATIC-Familie und ist insbesondere für die Anforderungen der Fertigungstechnik entworfen worden.

S7-400: Die S7-400 zeichnet sich durch große Leistung aus und ist für Datenintensive und anspruchsvolle Aufgaben der Prozesstechnik geeignet. Sie wird oft als Koordinator zwischen Automatisierungsanlagen betrieben.

WinAC: Das Windows Automation Center ist die softwarebasierte Steuerung der SIMATIC Familie und wird auf den IPCs (Industry PC) verwendet. Sie nutzt die Ressourcen des IPCs und zeichnet sich dadurch durch eine hohe Leistungsfähigkeit und großen Speichbereich aus.

2.1.3 HMI-Geräte

Geräte zum Bedienen und Beobachten

„Das Führen einer Maschine oder einer Anlage bedeutet, den Produktionsablauf zu beobachten und gegebenenfalls einzugreifen“ (Berger, 2000 S. 192). Human Machine Interfaces (Mensch-Maschine-Schnittstellen) übernehmen diese Aufgaben. Siemens stellt mit SIMATIC HMI benötigte Geräte wie

- Push Button Panels (PP),
- Textdisplays (TD),
- Operator Panels (OP) und
- Touch Panels (TP)

bereit. Das HMI-Gerät steht dabei in direkter Verbindung zum ablaufenden Programm auf der korrelierenden SPS. Hierdurch wird ein direktes Beobachten und Steuern von Prozessdaten ermöglicht.

Bedienoberfläche aus Bildern

Zentrale Elemente eines HMI-Projektes sind *Bilder*. Sie bestehen aus statischen und dynamischen Anzeige- und Bedienelementen, welche die Zustände einer Maschine oder Anlage visualisieren bzw. steuern. Ein Bild kann für ein bestimmten Anlagenteil oder eine bestimmte Funktion zugeschnitten sein. Eine Navigation ausgehend von einem Startbild ist entweder möglich durch HMI-Ereignisse vom Bediener durch Drücken von Navigationsbuttons und Funktionstasten oder durch das Auslösen von Ereignissen innerhalb des STEP7 Programms.

Statische oder dynamische *Anzeigeelemente* stellen Prozess-, Betriebs- und Störungszustände dar. Statische Elemente sind hierbei unveränderliche Texte und Grafiken; dynamische Elemente sind veränderbar und repräsentieren einen Variablenzustand z.B. durch LEDs, Ausgabefelder und Balkengrafiken.

Bedienelementen greifen direkt in den Prozessablauf z.B. durch Einstellungen von Sollwerten oder Auslösung von Ereignissen ein. Typische Vertreter von Bedienelementen sind Eingabefelder, Buttons oder analoge Schieberegler.

2.2 Softwarekomponenten

2.2.1 Projektierungssoftware

SIMATIC Manager STEP7 und TIA Portal

Zur Projektierung einer Automatisierungslösung mit SIMATIC Komponenten dient seit 1996 das Programm STEP7. Hier fließen alle notwendigen Tools, wie Hardwarekonfiguration, Programmierung, Kommunikation, Diagnose und Wartung zusammen. Jede Aufgabe wird dabei von einem eigenen Unterprogramm ausgeführt. So erfolgt die Hardwarekonfiguration zum Beispiel mit einem anderen Tool als die Bausteinprogrammierung. Das TIA Portal (Totally Integrated Automation) löst STEP7 seit 2011 schrittweise ab und implementiert alle Aufgaben nun einheitlich in einem einzigen Programm.

Modularer Aufbau mittels Bausteinen

Ein STEP7 Programm besteht aus verschiedenartigen Bausteinen. Diese dienen der Modularisierung des Programms und fördern die Widerverwendbarkeit von Code, Kapselung von Daten und erhöhen die Lesbarkeit des Programms.

Tabelle 2-2 Bausteinarten und deren Verwendung

Art	Bedeutung	Verwendung
OB	Organisations-baustein	• Schnittstelle zwischen Betriebssystem und Anwenderprogramm • Wird bei bestimmten Ereignissen vom BS aufgerufen • Vergleichbar zur `main()` Methode eines Hochsprachenprogramms
FB	Funktions-	• Parametrierbarer Teil des Anwenderprogramms

Art	Bedeutung	Verwendung
	baustein	• Speichert Daten der Funktion lokal in einem Instanz-Datenbaustein • Vergleichbar zu Hochsprachenmethoden mit statischen Feldern
FC	Funktion	• Dient zum Kapselung häufig wiederkehrender Automatisierungsfunktionen • Vergleichbar zu gewöhnlichen Hochsprachenmethoden
DB	Datenbaustein	• Enthält die Daten des Anwenderprogramms • Kann als Globaler-DB (Zugriff von allen Bausteinen aus möglich) oder Instanz-DB (einem FB zugeordnet) verwendet werden

(Berger, 2000 S. 143)

Tabelle 2-3 Wichtigste Organisationsbausteine

Organisationsbaustein	Verwendung
OB1	Der zentrale Organisationsbaustein, welcher automatisch vom SPS-Betriebssystem zum Beginn jedes Zyklus aufgerufen wird
OB35	Ein Weckalarm-Baustein, welcher zu bestimmten einstellbaren Zeiten aufgerufen wird
OB100	Dieser Organisationsbaustein wird nach dem Versetzen der SPS von STOP auf RUN aufgerufen und abgearbeitet

Wesentliche Schritte beim Projektieren

1. *Hardware-Konfiguration:* Hier werden die Baugruppen (CPU, Ein- und Ausgabebaugruppen usw.) auf dem Baugruppenträger (Rack) angeordnet und die einzelnen Baugruppen parametriert.
2. *Kommunikation:* Hier kann die Kommunikation der S7 mit anderen Systemen über die Komponenten aus SIMATIC NET festgelegt werden.
3. *Anwenderprogramm:* Das Anwenderprogramm kann in drei möglichen Programmiersprachen geschrieben werden und wird mit STEP7 auf die CPU geladen.

(Irmer, 2012 S. 13)

2.2.2 Datentypen und Programmiersprachen

Datentypen

STEP7 stellt eine Vielzahl an Datentypen für verschiedene Zwecke bereit. Eine Verwandtheit zu den Datentypen aus höheren Programmiersprachen wie C, C++ oder Java ist ersichtlich. Das Programm basiert auf einem 16 Bit Little Endian Datensystem, so ist der Datentyp `INT` grade diese 16 Bit groß. Little Endian meint, dass das höchst wertigste Bit (MSB – Most Signifcant Bit) an der binären Stelle 2^0 zu finden ist.

Besonderheiten im Gegensatz zu konventionellen höheren Programmiersprachen ist die Unterscheidung zwischen `INT` und `WORD`.

Der ebenfalls 16 Bit große Typ `WORD` stellt die vorzeichenlose, `INT` die vorzeichenbehaftete Version dar. `DINT` und `DWORD` erweitern diese Typen um ihre 32 Bit große Pendants.

Der Aufbau des Datentyps `REAL` entspricht weitgehend dem des bekannten `float` oder `double` Typs in `C` mit Mantisse, Basis und Exponent. Datum und Zeitangaben werden in speziellen STEP7 eigenen Datentypen `S5TIME`, `TIME`, `DATE` und `TIME_OF_DAY` gespeichert und enthalten codierte Informationen über einen bestimmten Zeitpunkt oder eine Zeitspanne. Eine vollständige Übersicht bietet die nachstehende Tabelle. Für die Entwicklung der Applikation ist dieses Wissen essentiell, da der Zugriff exklusiv auf diesen Datentypen basiert.

Abbildung 2-3 Elementare Datentypen in STEP7

Datentyp	(Breite)	Beschreibung	Beispiele zur Konstanten-Schreibweise
BOOL	(1 Bit)	Bit	FALSE TRUE
BYTE	(8 Bits)	8bit-Hexazahl	B#16#00 oder 16#00 B#16#FF oder 16#FF
CHAR	(8 Bits)	ein Zeichen (ASCII)	abdruckbares Zeichen, z.B. 'A'
WORD	(16 Bits)	16bit-Hexazahl	W#16#0000 oder 16#0000 W#16#FFFF oder 16#FFFF
		16bit-Binärzahl	2#0000_0000_0000_0000 2#1111_1111_1111_1111
		Zählwert, 3 Dekaden BCD	C#000 C#999
		2 × 8bit-Dezimalzahlen ohne Vorzeichen	B#(0,0) B#(255,255)
DWORD	(32 Bits)	32bit-Hexazahl	DW#16#0000_0000 oder 16#0000_0000 DW#16#FFFF_FFFF oder 16#FFFF_FFFF
		32bit-Binärzahl	2#0000_0000_..._0000_0000 2#1111_1111_..._1111_1111
		4 × 8bit-Dezimalzahlen ohne Vorzeichen	B#(0,0,0,0) B#(255,255,255,255)
INT	(16 Bits)	Festpunktzahl	–32 768 +32 767
DINT	(32 Bits)	Festpunktzahl	L#–2 147 483 648 L#+2 147 483 647 („L#" kann entfallen, wenn die Zahl außerhalb des INT-Zahlenbereichs liegt)
REAL	(32 Bits)	Gleitpunktzahl (Wertebereich siehe Text)	Exponentialdarstellung: +1.234567E+02
			Dezimaldarstellung: 123.4567
S5TIME	(16 Bits)	Zeitwert im SIMATIC-Format	S5T#0ms S5TIME#2h46m30s
TIME	(32 Bits)	Zeitwert im IEC-Format	T#–24d20h31m23s647ms TIME#24d20h31m23s647ms
			T#–24.855134d TIME#24.855134d
DATE	(16 Bits)	Datum	D#1990-01-01 DATE#2168-12-31
TIME_OF_DAY	(32 Bits)	Tageszeit	TOD#00:00:00 TIME_OF_DAY#23:59:59.999

(Berger, 2000 S. 163)

Programmiersprachen

Die Programmierung des STEP7 Programms erfolgt mittels einer der drei Programmiersprachen AWL, FUP und KOP. Diese orientieren sich jeweils an die Darstellung aus den Bereichen der Assemblersprache, Digitaltechnik und Stromlaufplänen.

- *AWL (Anweisungsliste):* Die Steuerungsaufgabe wird durch eine Folge von Anweisungen bestehende aus Operationen und Operanden realisiert. Es ist ein deutlicher Bezug zur prozessornahen Assemblersprache festzustellen.
- *FUP (Funktionsplan):* Hier wird mittels Verbinden von Boxen das Programm grafisch erstellt. FUP bietet Funktionsboxen zum Verknüpfen von Signalzuständen und Boxen zur Verarbeitung von Verknüpfungsergebnissen. Es ist verwandt mit der Darstellung von Gattern in der Digitaltechnik.
- *KOP (Kontaktplan):* Die Programmierung erfolgt durch das Anordnen grafischer Programmelemente wie Kontakte, Schalter, Spulen und Boxen. Sie lehnt sich an die Darstellung eines Stromlaufplans an.

2.2.3 Visualisierungssoftware

WinCC flexible

Die Projektierung von HMI-Geräten im Bereich der Fertigungsautomatisierung erfolgt zumeist über das Programm WinCC flexible. Es übernimmt neben der Parametrierung der Verbindung zur SPS und der Kommunikation mit Variablen, vor allen Dingen die Erstellung von Bildern. Für die Entwicklung der Android-Applikation spielt die Analyse dieses bestehenden Tools eine zentrale Rolle und soll sich an die zur Anwendung kommenden, etablierten Konzepte orientieren.

Entsprechend allgemeiner Windows Anwendungen und speziell grafikverarbeitender Programme, bietet WinCC flexible die Möglichkeit, per Drag&Drop Bedien- und Anzeigeelemente auf einen virtuellen Bildschirm zu platzieren. Größe und Aussehen dieses Bildschirms lehnen sich an das Original-HMI-Gerät an. Mittels eingebetteter Optionsfenster können platzierte Objekte parametriert werden. Wichtigste Eigenschaften sind:

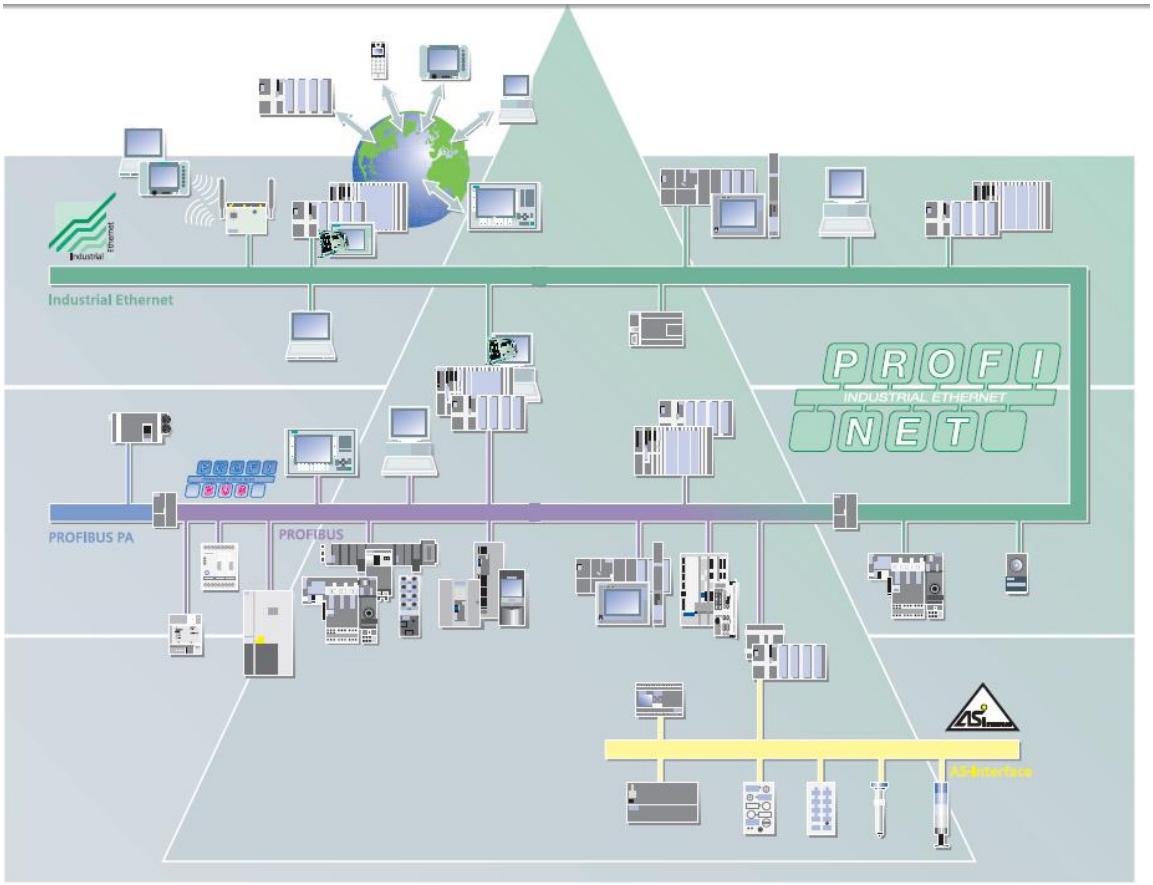
- *Verbindung:* Einstellung der Verbindung zu einer lokalen oder einer mit der SPS synchronisierten Variable
- *Darstellung:* Parametrierung von z.B. statische Größe, Position, Farbe, Schriftgröße
- *Events:* Auslösen oder Abfangen von Ereignissen und Reaktionen auf diese. Objekteigenschaften können hierdurch dynamisch verändert werden

2.3 Schnittstellen und Kommunikation

SIMATIC NET

„Die industrielle Kommunikation bildet das Rückgrat moderner Automatisierungslösungen. Die dort realisierten Kommunikationsnetze und -produkte ermöglichen eine durchgängige Kommunikation zwischen unterschiedlichsten Automatisierungskomponenten und –geräten“ (Siemens AG, 2008 S. 9).

Abbildung 2-4 Automatisierungspyramide von SIMATIC NET



(Siemens AG, 2008 S. 10)

Die Automatisierungspyramide teilt die Kommunikationsstruktur in drei Bereiche Feldebene, Zellebene und Leitebene ein:

- Die *Feldebene* übernimmt die Aufgaben der Prozess- und Feldkommunikation unter Nutzung der Feldbusse PROFIBUS und AS-Interface.
- In der *Zellebene* kommunizieren Automatisierungssysteme mit HMI-Geräten oder PCs zum Bedienen und Beobachten der erfassten Prozessdaten. Zur Anwendung kommen die Kommunikationsnetze PROFIBUS und Industrial Ethernet.

- Die *Leitebene* bietet den Rahmen zur Speicherung, Analyse und Weiterverarbeitung der Prozessdaten auf den zwei untergeordneten Ebenen. Industrial Ethernet eignet sich hierfür am besten.

(Siemens AG, 2008 S. 10)

Industrial Ethernet und PROFINET

Industrial Ethernet erweitert die Standard-Ethernet-Schnittstelle auf die industriellen Anforderungen unter dem Namen PROFINET (Process Field Ethernet). Entsprechend der Notwendigkeit eines deterministischen und echtzeitfähigen industriellen Bussystems, implementiert PROFINET im Gegensatz zum weit verbreiteten Ethernet z.B. im Büronetzwerken diese Eigenschaften.

PROFIBUS

PROFIBUS steht für Process Field Bus und bezeichnet den feld- und prozessnahen Bus von SIMATIC NET. Seine Hardwarespezifikation entspricht die eines für die anspruchsvollen Bedürfnisse der Industrie angepassten 9-poligen Sub-D Steckers. Abschlusswiderstände definieren die Enden der typischerweise linienförmigen Busstruktur. Die Teilnehmer werden über eindeutige Adressen adressiert.

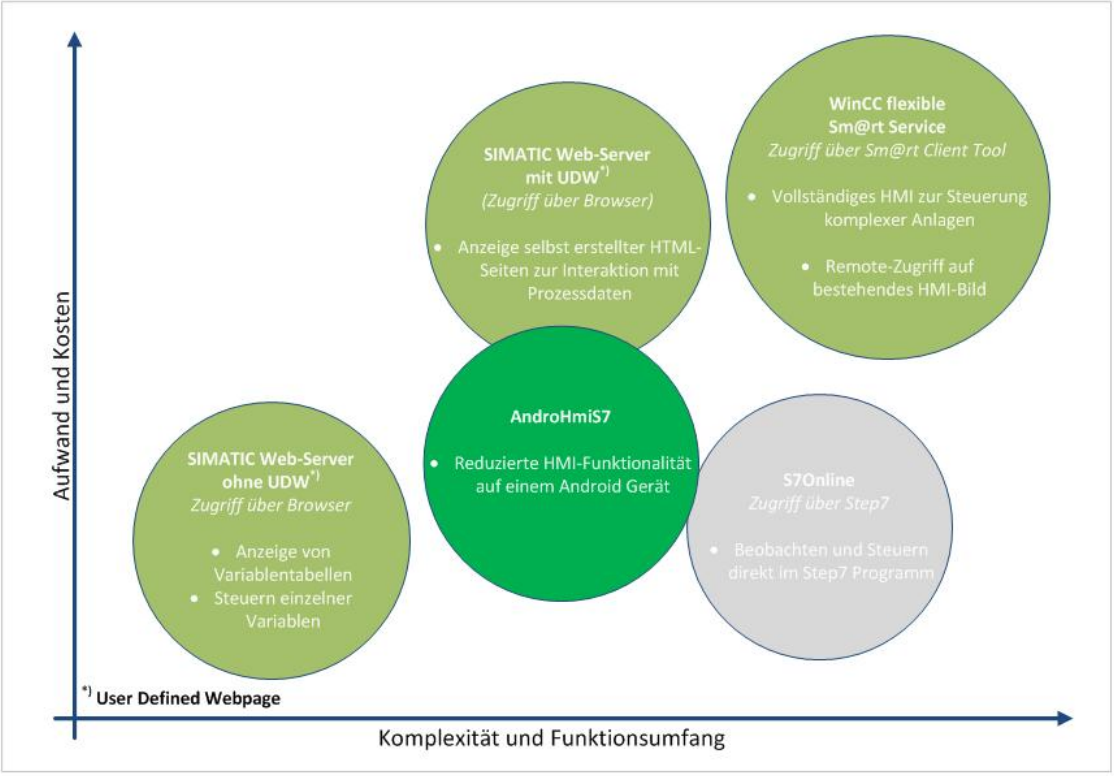
3 Bestehende Remote-Zugriffsmöglichkeiten auf Prozessdaten

3.1 Vergleich App-Lösung zu bestehende Lösungen

Einordnung und Abgrenzung

Verglichen wird die App-Lösung mit bereits existierenden Lösungen für den Remote-Zugriff auf SPS-Prozessdaten. Das folgende Diagramm zeigt die Einordnung der App-Lösung in Abhängigkeit von Komplexität und Funktionsumfangs auf der einen, sowie Aufwand und Kosten auf der anderen Seite.

Abbildung 3-1 Abgrenzung der App-Lösung zu bestehenden Lösungen



Die Analyse zeigt, dass die App-Lösung *AndroHmiS7* mit geringerem Aufwand und Kosten eine ähnliche Komplexität und Funktionsumfang besitzt, wie der SIMATIC Web-Server mit User-Defined-Webpages.

- Der *Aufwand* ist geringer, da die Applikation als Standalone-Lösung konzipiert ist und keinen Eingriff in das Programm benötigt.
- Die *Kostenreduzierung* wird durch das schlanke Design einer Android-Applikation und den Verzicht auf zusätzliche Hardware erreicht.
- Eine hohe *Komplexität* wird durch den performanten Zugriff auf die Prozessdaten in allen Speicherbereichen erreicht.
- Der Funktionsumfang ermöglicht die individuelle Erstellung von Bildern mit vielen Anzeige- und Bedienelementen.

3.1.1 SIMATIC Web-Server

Beschreibung

Der SIMATIC Web-Server ermöglicht die Bereitstellung von Prozessdaten über das lokale Netzwerk oder Internet unter Nutzung eines Web-Browsers.

Für einige unterstütze CPUs der SIMATIC Reihe kann in der Hardwarekonfiguration ein Web-Server aktiviert werden, welcher Diagnose- und Prozessdaten bereitstellt. Dieser wird beim Starten des Controllers automatisch mit gestartet. Über eine vorab festgelegte IP-Adresse innerhalb des gleichen Subnetzes, kann ein beliebiges Gerät über einen Browser auf den SIMATIC Web-Server zugreifen. Hierfür stellt dieser eine vorgefertigte Benutzeroberfläche bereit, über welches sich Variablentabellen, Diagnoseeinträge und - je nach Verfügbarkeit - auch *User-Defined-Webpages* anzeigen lassen.

User-Defined-Webpages sind selbsterstellte Webseiten, welche nach eigenen Vorstellungen gestaltet werden können und mit den Variablen aus der SPS über symbolischen Namen verknüpft werden. Die HTML Seite lässt sich dann über den Browser laden und die Prozessdaten der SPS grafisch aufbereitet einsehen und steuern. Ebenfalls bieten Variablentabellen eine Möglichkeit der Interaktion mit den Aktualdaten – allerdings ohne grafisches Interface.

Einschränkungen beim SIMATIC-Web-Server

Software

- STEP7 5.5 mit integrierter Erweiterung S7-Web2PLC notwendig
- Aufwand beim Projektieren einer User-Defined-Webpage, insbesondere nach Änderungen erfolgt ein Durchlauf der kompletten Phase der Projektierung mit den Schritten:
  - Konfiguration der SPS Station
  - Erstellen der Symbole in der Symboldatei
  - Symbole exportieren, Projekteinstellungen und Enumerationen mit S7-Web2PLC
  - Erstellen der HTML-Datei
  - Projekteinstellungen, Enumerationen und DB333 generieren mit S7-Web2PLC
  - Erstellung des S7-Programms
  - Aufruf der Webseite mit einem Webbrowser
- Es werden weitere Bausteine benötigt (DB333, DB334) und in die Programmstruktur eingegriffen (Aufruf von SFC99 im OB1)

Hardware

- Benötigt neuste CPU's mit speziellen CP's (Communication Processor)
- Viele CPUs unterstützen zwar den Web-Server aber keine User-Defined-Webpages

Funktion

- Web-Server dient nur als Diagnose-Möglichkeit mir rudimentären Funktionsumfang und zum einzelnen Abfragen von Variablen
- Keine performante zyklische Abfrage möglich

Vor- und Nachteile SIMATIC Web-Servers

Tabelle 3-1 Vor- und Nachteile des SIMATIC Web-Servers

Vorteile	Nachteile
+ Breite Möglichkeiten in der Nutzung von User Defined-Webpages durch Verwenden von bekannten HTML Strukturen	- Nur auf wenigen CPUs ist der Web-Server aktivierbar
+ Gesichertes Abfragen der Prozessdaten über eine interne Siemens-Schnittstelle	- Häufig ist nur eine Beobachtung über eine Variablentabelle und einzelnes Lesen/ Schreiben von Variablen möglich
+ Nutzung von Browsern zu Darstellung der Prozessdaten benötigt keine weitere Software	- Ständiges Abfragen (Polling) von Daten schwierig zu konzipieren
+ Autorisierungssystem ermöglicht die Vergabe von Berechtigungen	- schlechte Performance
	- Umweg über den Browser für mobile Geräte eher umständlich, daher kein einheitliches App-Gefühl möglich
	- Eingriff in die Programmstruktur durch Aufruf weiterer SFC und erstellen von DBs

3.1.2 WinCC Sm@rt-Service

Beschreibung

Der WinCC Sm@rt-Service ermöglicht das Einsehen und Steuern eines in WinCC erstellten und auf einem HMI ausgeführten Bildes auf einem entfernten Rechner. WinCC ist eine Software zur Erstellung und Projektierung von Oberflächen für SIMATIC HMI-Geräte – den sogenannten *Bildern*.

Die Steuerung einer komplexen Anlage erfolgt zumeist über HMIs, also Terminals mit Ein- und Ausgabemöglichkeiten über Bildschirme. Um von einem entfernten Rechner auf diese Bilder zugreifen zu können, kann die HMI-Runtime mit der Funktion Sm@rt-Service gestartet und mit einem Smart-Client auf diese zugegreifen werden. Hierbei entsteht eine Client-Server Verbindung. Der Benutzer sieht exakt den Inhalt des Bildes und kann mit diesem interagieren als wäre er vor Ort.

Einschränkungen beim WinCC Sm@rt-Service

Software

- Weitere Komponenten WinCC flexible sowie Sm@rt-Service sind notwendig

- Ein HMI Bild muss bereits vorhanden sein und ausgeführt werden, um es über den Sm@rt-Service öffentlich machen zu können, da ein direkter Zugriff auf die Prozessdaten einer SPS nicht möglich ist
- Es muss auf dem entfernten Rechner der Windows-basierte Smart-Client installiert sein

Hardware

- Es wird ein HMI-Panel oder zumindest ein Rechner benötigt, auf welchem die WinCC-Runtime läuft, mit welcher eine Verbindung aufgebaut werden soll

Funktion

- Funktional gibt es keine Einschränkungen, da der gesamte Pool eines WinCC-Bildes verwendet werden kann

Vor- und Nachteile des WinCC Sm@rt-Service

Tabelle 3-2 Vor- und Nachteile des WinCC Sm@rt Service

Vorteile	Nachteile
+ Großes Angebot von Anzeigeelementen, Bedienelementen und Funktionen	- Benötigt viele weitere Softwarekomponenten wie die Visualisierungssoftware WinCC, Sm@rt-Service und Smart-Client, sowie einen Runtime-Rechner oder ein Bedienterminal
+ Sichere Verbindung über Smart-Access bzw. Smart-Client mit inbegriffener Autoritätsüberprüfung	- Portierung ist auf ein mobiles Gerät nahezu unmöglich, da der spezielle Smart-Client benötigt wird
	- Ein bestehendes WinCC Bild muss erstellt sein und aktuell ausgeführt werden
	- Nur lesender Zugriff auf das bestehende Bild

3.1.3 STEP7

Zugriff über das Projektierungstool STEP7

Eine weitere Möglichkeit des Remote-Zugriffs bietet das Projektierungstool STEP7 im Diagnose- oder Beobachten-Modus. Hierbei kann auf die vorhandenen Bausteine der SPS lesend und schreibend zugegriffen und so das laufende Programm beobachtet werden. Über Variablentabellen können Variablen aller Speicherbereiche beobachtet und gesteuert werden. Die Möglichkeit, über die Schnittstelle S7Online auf Prozessdaten zuzugreifen wird allerdings ausschließlich für Inbetriebnahmen, Diagnose und Wartungsarbeiten verwendet und ist aufgrund der fehlenden Visualisierung der Prozessdaten nicht für die Steuerung einer Anlage während der Betriebs geeignet.

3.2 Verwendung und Nutzen der App-Lösung

AndroHmiS7 im Vergleich mit anderen Remote-Zugriffsarten

Zusammenfassend zeigt sich, dass die Applikation *AndroHmiS7* für mobile Android-Geräte eine Nische im Remote-Zugriff auf Prozessdaten einer SIMATIC SPS ausfüllen könnte.

Bereits vorhandene Systeme, wie der SIMATIC Web-Server und WinCC Sm@rt-Service sind etabliert für den klassischen Desktop-PC oder für Notebooks. Eine auf die Gegebenheiten des mobilen Geräts zugeschnittene Lösung existiert nicht.

Insbesondere die zusätzlichen Softwarekomponenten, die beim Sm@rt-Service notwendig sind, sowie die geringe Verfügbarkeit des Web-Servers mit häufig eingeschränkter Funktion reduzieren den Nutzen für mobile Geräte.

Vorteile der App-Lösung

- Interaktion mit Prozessdaten ohne die Notwendigkeit weiterer Softwarekomponenten
- Zugriff auf nahezu alle S7-300/400 und WinAC CPU-Typen ohne spezielle Controllereinstellungen
- Einfaches und schnelles Erstellen von Bildern und deren Ausführung auf einem Gerät
- Performante zyklische Abfrage der Prozessdaten über ISO on TCP
- Kein Eingriff in das STEP7 Programm notwendig

3.2.1 Anwendungsfälle und Zielbereiche

Tabelle 3-3 Anwendungsfälle und Zielbereiche der App-Lösung

Zielbereich	Nutzen
Anlage im Betrieb	+ Ausgabe wichtiger Statusdaten einer Anlage im Betrieb - wie z.B. Fehlermeldungen, Betriebsarten und Statusmeldungen + Diagnosemöglichkeit und Wartungshilfe z.B. durch mobiles Einlesen des Diagnosepuffer
Inbetriebnahme	+ Auslesen und Setzen bestimmter Variablen während der Inbetriebnahme z.B. zwecks mobiler Funktionstests von Anlagenteilen
Messemodelle	+ Einfaches HMI Gerät z.B. zur Steuerung von Demo-Projekten oder Testaufbauten auf Messen
Hausautomatisierung	+ Einfaches HMI Gerät z.B. zur Steuerung einfacher und risikoarmer Haussysteme wie Licht- und Fenstersteuerung
Experimentelle Anwendungen	+ Einfaches HMI Gerät z.B. zum Testen, Debuggen oder Ausgabe von Variablen

4 Kommunikationsbibliothek Libnodave

4.1 Übersicht

Allgemeines

„Zur Kommunikation mit Automatisierungsgeräten aus einer Applikation benötigt der Programmierer eine Schnittstelle, die ihm Zugang zu den Prozessdaten gewährt. Diese Schnittstellen auch Treiber genannt werden meist von den Geräteherstellern als [...] Bibliotheken zur Verfügung gestellt“ (Zroud, 2012 S. 23).

Open-Source Bibliothek zum Zugriff auf SPS Daten

Die kostenlose und quelloffene Bibliothek *Libnodave* von Thomas Hergenhahn ermöglicht den Zugriff auf Prozessdaten einer Siemens SIMATIC SPS und unterstützt laut Dokumentation die Typen S7-200/300/400. Bei der Analyse der Bibliothek konnte auch die WinAC zur Liste der unterstützen Typen ergänzt werden und voraussichtlich werden auch die neuen Typen S7-1200/1500 unterstützt.

Als Kommunikationsprotokolle (ISO/OSI Referenzmodell Schicht 3, 4) können verwendet werden:

- *ISO on TCP:* Paketorientierte Übertragung von Daten aus dem ISO-Transport-Protokoll verknüpft mit der Routing Fähigkeit des TCP/IP Protokolls.
- *S7OnIine:* Siemens eigenes Protokoll zur Kommunikation mit SIMATIC Hardware.

Unterstütze Schnittstellen sind (ISO/OSI Referenzmodell Schicht 1, 2):

- *Ethernet:* Schnittstelle zum Zugriff auf LAN oder WLAN aller unterstützter Controller.
- *MPI:* Serielle Schnittstelle über MPI Kabel zur S7-300/400.
- *PPI:* Serielle Schnittstelle über PPI Kabel zur S7-200.

(Hergenhahn, 2011)

Die Dokumentation, vollständiger Quellcode und kompilierte Versionen der *Libnodave* Bibliothek in der aktuellen Version 0.8.6 finden sich unter der Adresse: http://libnodave.sourceforge.net/.

Wichtigste Funktionen

- Verbindungsaufbau und -abbau zur SIMATIC Hardware S7-200/300/400 sowie WinAC
- Auslesen von Informationen über SPS Hard- und Software
- Einlesen von Bausteinlisten und Headerinformationen
- Lesen des Diagnosepuffers und der Systemzustandslisten (SZL)
- Einlesen und Übertragen von Code-Bausteinen und Programmteilen
- Eingänge, Ausgänge, Merker, Zähler, Timer und DBs lesen und schreiben

- Konvertierungen zwischen SIMATIC-Datentypen und C-Datentypen

(Zroud, 2012 S. 42)

Lizenzbedingungen

Die Bibliothek liegt als vollständige C-Quelle und vorkompilierte Bibliothek für C, C++, C#, Delphi, Pascal, Perl VB und VBA vor. Sie unterliegt der LGPL (GNU Lesser General Public License, ehemals GNU Library General Public License), welche eine Kopplung von proprietärem und freien Quellcode ermöglicht. Wird die Bibliothek unverändert übernommen und statisch oder dynamisch in den eigenen Quellcode eingebunden, so unterliegt lediglich die eingebundene Bibliothek weiterhin der LGPL Lizenz. Diese Lizenz muss nicht auf den selbst erstellten Quellcode ausgeweitet werden, wie es bei der GPL Lizenz der Fall wäre. Die Nutzung der *Libnodave* Bibliothek schränkt also rechtlich nicht die Entwicklung geschlossenen und proprietären Codes ein.

(GNU, 1991)

4.2 Vergleich zu anderen Bibliotheken

Prodave MPI/IE

Prodave MPI/IE ist eine Siemens-Schnittstelle zum Datenaustausch mit den SIMATIC-Controllern S7-200/300/400. Die Kommunikation erfolgt über die MPI-, PROFIBUS- oder Ethernet-Schnittstelle. Die Bibliothek kann über C, C++ und Visual Basic programmiert werden. Der Funktionsumfang entspricht der der *Libnodave* Bibliothek. Als Ursache hierfür ist anzunehmen, dass die Bibliothek auf Basis der Analyse der Prodave Bibliothek entwickelt wurde.

SAPI-S7

„SAPI-S7 steht für Simple Application Programmers Interface. S7 deutet auf die Verwendung des S7-Protokolls. Die Schnittstelle ist als C-Library implementiert und kann mit C bzw. C++ programmiert werden. Bevor eine Applikation mit einem Gerät kommunizieren kann, muss eine Verbindung mit STEP7 projektiert werden. Bei der Projektierung wird eine oder mehrere Verbindungen einem VFD zugeordnet. VFD ist eine Abkürzung für Virtual Field Device und dient als Abbildung eines Automatisierungsgerätes [...]. Die VFD's verbergen die Unterschiede zwischen den Geräten und erlauben eine einheitliche Sicht darauf“ (Zroud, 2012 S. 39).

Weitere Zugriffsmöglichkeiten auf Prozessdaten aus eigenen Anwendungen

OPC: Ole for Process Control ermöglicht die Kommunikation von Geräten unterschiedlicher Hersteller über eine abstrahierte und dem System übergeordnete Ebene. Dazu fordert ein Server die Prozessdaten über den Feldbus von der Automatisierungshardware an, bereitet die Daten auf und stellt diese für verbundene Clients zur Verfügung.

WinAC ODK: Speziell für die Soft-SPS WinAC existiert die ODK (Open Development Kit) Schnittstelle, welche eine Kommunikationsmöglichkeit

gekapselt über eine DLL mit dem Windows Betriebssystem bereitstellt. Die DLL wird im STEP7 Programm eingebunden und Funktionsaufrufe können hieraus im laufenden Betrieb getätigt werden. Das ausführende Betriebssystem kann die empfangenen Daten dann weiterverarbeiten und Rückgabewerte dem STEP7 Programm übergeben.

Fazit

Die Faktoren Kosten, Benutzbarkeit und Einfachheit der Implementierung waren bei der Wahl der Kommunikationsbibliothek ausschlaggebend. Der Funktionsumfang war bei allen untersuchten Bibliotheken ähnlich. Da die Hauptfunktion bei der Entwicklung der Applikation einschränkend auf den Zugriff auf Prozessdaten und Diagnoseeinträge lag, war der Funktionsumfang der *Libnodave* Bibliothek hierfür ausreichend. Obwohl kostenlos und nicht direkt von Siemens entwickelt, überzeugte die Bibliothek bei durchgeführten Tests durch die einfache Verwendung und stabile Ausführung.

4.3 Analyse und Funktionsweise

Libnodave bietet eine Vielzahl von Methoden für die Bereiche Verbinden, Lesen und Schreiben. Allen Methoden ist gemein, dass sie stets einen `int` Rückgabewert mit Fehlerinformationen zurückgeben. Dies erleichtert die Erkennung und Behandlung von Fehlern und ermöglicht die Weitergabe von Fehlerinformationen nach oben an die aufrufenden Instanzen.

4.3.1 Kommunikationsauf- und abbau

Socketkommunikation

Bei der Applikationsentwicklung kam das ISO on TCP Protokoll für die Verbindung zur SPS auf Port 102 zum Einsatz. Dieses eignet sich für die Verwendung mit der WLAN-Schnittstelle der mobilen Android-Geräte und verbindet die paketorientierte Kommunikation des ISO Protokolls mit der Routingfähigkeit des TCP/IP Protokolls. Hierfür wird eine bidirektionale Socketverbindung verwendet, welche eine standarisierte und plattformunabhängige Schnittstelle zwischen Betriebssystem und der Implementierung der Netzwerkschicht bildet.

Methoden zum Verbindungsaufbau

Zu Beginn des Verbindungsaufbaus wird ein Socket über `fds.rfd = openSocket(port, ip)` geöffnet. Dahinter verbirgt sich eine betriebssystemabhängige Implementierung der Socketverbindung, welche durch die Instanz `fds` des Typs `_daveOSserialType` repräsentiert wird.

Die Eigenschaften `wfd` und `rfd` stellen die ausgehenden bzw. eingehenden Kommunikationskanäle dar. Da es sich um eine bidirektionale Socketverbindung handelt, werden beide Kanäle mit `fds.wfd = fds.rfd` gleichgesetzt.

Die physikalische Verbindung zur SPS wird in der Instanz `di` des Typs `daveInterface` repräsentiert. Mit `di = daveNewInterface(fds, „IF1“, mpi, protocol, speed)` wird der Netzwerkadapter mit spezifischen Eigenschaften wie Geschwindigkeit und Protokolltyp sowie der aufgebauten Socketverbindung initialisiert.

Nach der erfolgreichen Initialisierung wird die eigentliche Verbindung zur SPS über `dc = daveNewConnection(di, mpi, rack, slot)` der Instanz `dc` des Typs `daveConnection` aufgebaut. Hier finden sich Siemens spezifische Eigenschaften wie die Rack- und Slot-Nummer der jeweiligen SPS.

Methoden zum Verbindungsabbau

Abgebaut wird die Verbindung in umgekehrter Reihenfolge stufenweise über `daveDisconnectPLC(dc)`, `daveDisconnectAdapter(di)` sowie der betriebssystemabhängigen Implementierung von `closeSocket(fds.rfd)`.

4.3.2 Lesen und Schreiben von Daten

Synchrones Schreiben und Lesen

Die einfachste Möglichkeit des Zugriffs auf die Prozessdaten der verbundenen SPS erfolgt über synchrone *read* bzw. *write* Aufrufe. Dabei wird für jeden Funktionsaufruf eine Anfrage über die bestehenden ISO on TCP Verbindung an die SPS gesendet und die empfangenen Daten beim Lesezugriff entweder in einen Buffer geschrieben oder im Falle einer Schreibanfrage aus diesem gelesen.

Der Lesemethode `daveReadBytes(dc, area, dbNum, start, length, buffer)` bzw. dessen Schreib-Pendant `daveWriteBytes(…)` werden dabei eine Referenz auf das `daveConnection` Objekt `dc` sowie der Buffer der zu lesenden oder schreibenden Daten übergeben. Beim Lesezugriff wird der Buffer mit den empfangenen Daten aus der SPS gefüllt, beim Schreiben wird aus diesem gelesen und der Inhalt in die SPS geschrieben. Auf welche Daten zugegriffen werden soll wird in den Parametern

- *area*: Speicherbereich (vgl. Kapitel 2.2.2),
- *dbNum*: Nummer des Datenbausteins,
- *start*: Die Startadresse als Byte-Offset und
- *length*: Die Länge der zu lesenden Daten

eindeutig festgelegt.

Konvertierung der Daten im Buffer

Aufgrund der unterschiedlichen Bit-Repräsentationen der Variablen in der SPS und Java, müssen die zu schreibenden Daten vor dem Senden bzw. empfangene Daten vor dem Lesen konvertiert werden.

Im Falle des Lesezugriffs, wird der an die Methode `daveReadBytes(…)` übergebenen Buffer mit Variablenwerten aus der SPS gefüllt. Auf diese

können mit den Methoden `daveGetS8from(buffer)` für vorzeichenbehaftete Variablen (signed) oder `daveGetU8from(…)` für vorzeichenlose Werte (unsigned) zugegriffen werden und liefern den konvertierten Wert als `int` zurück. Entsprechend der Größe der Variable (8, 16 oder 32 Bit) werden die äquivalenten Methoden `daveGetS16from(buffer)` bzw. `daveGetS32from(…)` und das entsprechenden Pendant für vorzeichenlose Variablen verwendet.

Beim Schreibzugriff auf Prozessdaten müssen die Werte vor der Übertragung in die SPS zuerst konvertiert in den Buffer gelegt werden. Hierzu bietet *Libnodave* die Methoden `davePut8(buffer, value)` und entsprechend der oben erläuterten Konvertierungsfunktionen im Lesezugriffsfall ihre Äquivalente an.

Multiple Requests

Um den Kommunikationsaufwand über die WLAN-Schnittstelle zu begrenzen, bieten sich *Multiple Requests* an. Hier wird nicht für jeden Lese- oder Schreibbefehl synchron eine Anfrage an die SPS gesendet, sondern bis zu 20 Befehle in einer Anfrage zusammengepackt und gemeinsam abgeschickt. Der Vorteil liegt im Zeitgewinn: Benötigt eine einzelne Anfrage unter normalen Bedingungen über WLAN ca. 10-20ms, so erhöht sich die Zeit eines Multiple Request nur marginal. Grund hierfür ist der große Overhead eines Kommunikationsauftrags durch Anfrage (Request), benötigte Headerinformationen, Routing und Bestätigung (Acknowledge). Eine Erhöhung der übertragenen Daten fällt dabei nicht derart ins Gewicht, da der Overhead bei geringer Zunahme der eigentlichen Daten konstant bleibt.

Libnodave bietet hierfür die eine Vorgehensweise durch schrittweisen Aufruf folgender Methoden:

- `davePreapareReadRequest(dc, pdu)`: Bereitet eine Leseanfrage für die aufgebaute `dc` Verbindung und einem leeren `pdu` vor. Einer PDU (Protocol Data Unit) entspricht dabei ein zu übertragendes zusammenhängendes Paket.
- `daveAddVarToReadRequest(pdu, area, dbNumber, start, length)`: Fügt eine Variable mit entsprechenden Adressinformationen zur Leseanfrage hinzu.
- `daveExecReadRequest(dc, pdu, resultSet)`: Führt die Anfrage an die SPS als gesammeltes Paket aus und speichert die empfangenen Werte in einem `resultSet`.
- `daveUseResult(dc, resultSet, index)`: Gibt den Wert der an der `index` Position hinzugefügten Leseanfrage unter Nutzung der Verbindung `dc` und des `resultSet` zurück.
- `daveFreeResult(resultSet)`: Gibt den reservierten Speicher wieder frei.

Mit Ausnahme der `daveUseResult(…)` Methode kann oben beschriebene Vorgehensweise auch auf die Zusammenführung von

Schreibanfragen unter Nutzung des Schlüsselwortes `Write` angewandt werden.

4.3.3 Diagnose und Steuerung

Lesen des Diagnosepuffers

Alle Diagnose und Betriebszustandsdaten werden bei SIMATIC Controllern einheitlich in SZL (Systemzustandslisten) gespeichert. Über einen Index lassen sich Teillisten aus dieser extrahieren. Einer der wichtigsten Teillisten ist der Diagnosepuffer, welcher alle Events und Fehler des laufenden Programms oder der Hardware aufzeichnet und für die Diagnose die erste Anlaufstelle ist. Die Methode für den Abruf dieser SZL ist `daveReadSZL(dc, id, index, buffer)`. Die über die `id` identifizierten Teillisten der SZL werden hierbei in den übergebenen `buffer` gelegt.

Steuerung der SPS

Die Methoden `daveStart(dc)` und `daveStop(dc)` senden eine Start- bzw. Stop-Anfrage an die SPS.

4.4 Portierung für die Verwendung mit Android

Die Verwendung der *Libnodave* Bibliothek unter Android erfordert aufgrund der unterschiedlichen Programmiersprachen eine Portierung. Als Ergebnis dieser Portierung steht eine Linux SO-Bibliothek (Dynamically Linked Shared Object Library), welche in das Android-Projekt dynamisch während der Laufzeit eingebunden werden kann.

Vorgehensweise

Ein selbst entwickeltes in C programmiertes Interfacemodul fungiert dabei als Schnittstelle zwischen der in Java programmierten Android-Applikation und der in C vorliegenden *Libnodave* Bibliothek. Sie nimmt Methodenaufrufe von der Java-Seite entgegen und leitet diese angepasst an die C-Seite weiter. Hierfür wird das JNI (Java Native Interface) verwendet, welches grade diese Verbindung von Java mit nativem C-Code durch Bereitstellungen von Methoden verwaltet.

Innerhalb des Java-Quellcodes wird die Bibliothek statisch eingebunden.

```
1. static {
2.     System.loadLibrary("s7com");
3. }
```

Die Methoden des Interfacemoduls müssen auf der Java Seite mit dem Schlüsselwort `native` deklariert werden.

```
1. public native int connect(String ip);
```

Innerhalb des Interfacemoduls ist dem Methodennamen zusätzlich das Schlüsselwort „Java_“ sowie die Aufrufherkunft – also der Paket- und Klassennamen der aufrufenden Klasse getrennt durch Unterstriche - vorgestellt. Neben der Möglichkeit selbst definierte Parameter zu übergeben, müssen zwei weitere zwingend mit übergeben werden.

```
1.  int Java_com_androhmis7_connection_S7Connection_connect(JNIEnv* env,
2.          jobject this, jstring ip);
```

“The first parameter, the *JNIEnv* interface pointer, points to a location that contains a pointer to a function table. Each entry in the function table points to a *JNI function.* [...] The second argument [...] is a reference to the object on which the method is invoked, similar to the *this* pointer in C++ (Liang, 2002).

Der `JNIEnv` Zeiger stellt also die JNI-Funktionen wie beispielsweise das Empfangen einer Speicheradresse (s.u.) bereit. Das `jobject` repräsentiert das aufrufende Objekt und bietet somit eine direkte Manipulationsmöglichkeit.

```
1.  mBufferSet[readRequestCount].pValue =
2.                  (uc*) (*env)->GetDirectBufferAddress(env, srcBuffer);
```

NDK - Native Development Kit

Zum Kompilieren des Interfacemoduls und zum Erstellen der SO-Bibliothek bietet Android das NDK (Native Development Kit) an. Hierzu ist eine Linux-Distribution wie beispielsweise *Ubuntu* notwendig. Eingebettet in das Android-Projekt werden die C-Quelldateien des Interfacemoduls im Ordner `jni` abgelegt. Über den Befehl `ndk-build` wird der Kompilierprozess automatisch unter Verwendung den GNU-C-Compilers und des JNI gestartet und die SO-Bibliothek erstellt.

Für diesen Prozess ist es notwendig, dem NDK die Struktur des Interfacemoduls mit allen einzubettenden Quelldateien, Bibliotheken und Flags mitzuteilen. Diese Funktion übernimmt der sogenannte *Makefile* `Android.mk` mit den wichtigsten Parametern:

- LOCAL_MODULE: Name des zu erstellenden Moduls
- LOCAL_CFLAGS: Flags zur Steuerung des Kompilierprozesses durch im Quellcode befindliche `#ifdef` Deklarationen zum bedingten Kompilieren von Quellcodedateien
- LOCAL_SRC_FILES: Einzubindende C-Quellcodedateien

Interfacemodul

Das Interfacemodul lagert die zentralen Aufgaben Verbindung, Lesen und Schreiben in eigene Quelldateien aus und erhöht hierdurch die Modularisierung.

Tabelle 4-1 Quelldateien des Interfacemoduls

Quelldatei	Verwendung
s7com.c	Globale Definition von Konstanten und globalen Variablen
s7connection.c	Verbindungsauf- und Abbau, Zugriff auf Diagnoseeinträge und Steuerung der SPS
s7read.c	Methoden zum synchronen und asynchronen Lesen mittels Multiple Request von Prozessdaten aus der SPS
s7write.c	Methoden zum synchronen und asynchronen Schreiben von Prozessdaten mittels Multiple Request in die SPS
s7util.c	Hilfsmethoden wie z.B. Zeitmessung

5 Mobiles Android-Framework

5.1 Einführung Applikationen für mobile Geräte

Merkmale Applikationen für mobile Geräte

„Der Anwender einer Applikationen für mobile Geräte erwartet eine intuitive Bedienung, ohne zuvor eine Anleitung oder eine Handbuch lesen zu müssen. Dies bedeutet zwangsläufig, dass die Funktionalität auf ein, zwei wohl definierte Funktionen reduziert sein muss, um den Anwender nicht zu überfordern. Hier unterscheidet sich die mobile Applikation grundlegend von einer Desktop Anwendung – beispielsweise Word – bei welcher eine hohe Funktionalität gekapselt in einem einzigen Programm erwünscht ist“ (Moecker, 2012 S. 20).

Besonderheiten im Gegensatz zu Desktopanwendungen

Zu den Besonderheiten mobiler Applikationen zählen die Punkte:

- Geringe Rechenleistung im Gegensatz zu Desktop Computern
- Knapper Speicherplatz und geringer Arbeitsspeicher
- Beschränkte Bildschirmgröße
- Störempfindlichkeit, insbesondere durch
  - Empfang von Anrufen, SMS oder E-Mail
  - Unterbrechung der Netzwerk- oder Funkverbindung
  - Änderung der Orientierung (Landschaft/ Porträt)
  - Unterbrechung von Prozessen zwecks Ressourcenfreigabe

(Moecker, 2012 S. 21)

Grundsätze für die Entwicklung mobiler Applikationen

Um diesen Vorrausetzungen gerecht zu werden, sollten folgende Grundsätze bei der Entwicklung mobiler Applikationen beachtet werden:

- Vermeidung rechenintensiver Operationen
- Defensive Nutzung der Arbeitsspeicherressourcen durch optimierten Code
- Manuelles Freigeben von Speicherbreichen zusätzlich zum Garbage Collector (Tool zur automatischen Freigabe von Speicherbereichen)
- Sinnvolle Nutzung des Bildschirms durch zugängliche Bedienelemente
- Vermeidung von Überladungen durch zu viele UI-Elemente
- Entwurf der Applikation für verschiedene Bildschirmgrößen und Auflösungen durch eine Beschreibung der Struktur, anstatt pixelgenauer Positionierung der UI Elemente

(Moecker, 2012 S. 21)

Grundlagen des Android-Frameworks

Obwohl Android immer Google zugerechnet wird, wurde das Android-Framework tatsächlich von der *Open Handset Alliance*, einem Zusammenschluss aus ca. 60 Firmen, dem auch die großen Smartphone Hersteller HTC und Samsung beiwohnen, entwickelt. Das Framework beinhaltet das Android-Betriebssystem, Middleware sowie weitreichende Bibliotheken auf Basis der Programmiersprache Java.

Zu den grundlegenden Features zählen:

- Kapselung innerhalb einer virtuellen Maschine (Dalvik) - ähnlich der Java Runtime
- Integrierter Web Browser
- Unterstützung von Medienformaten (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF), sowie 2D und 3D Grafiken
- SQLite für die strukturelle Datenspeicherung
- Kommunikation über GSM Telefonie,3G, EDGE, Bluetooth und WiFi Implementierungen
- Unterstützung von Kamera, GPS, und weitere Sensoren

(Google, 2012)

Ein wichtiger Aspekt der Android Philosophie ist die weitgehende Trennung vom bereitgestellten Framework und dessen hardwaremäßige Implementierung auf den eigentlichen Geräten. Die *Open Handset Alliance* stellt demnach lediglich ein Open-Source-Betriebssystem bereit und ermöglicht einer Vielzahl von Herstellern, dieses in ihre Geräte zu implementieren.

Dieser Ansatz ermöglicht auf der einen Seite eine verschiedenartige Implementierung und fördert die schnelle Verbreitung, stößt auf der anderen Seite aber dadurch auf Kompatibilitätsprobleme. Deutlich wird dies zum Beispiel anhand der verschiedenen Bildschirmauflösungen und Bildschirmgrößen: Es werden neben einem handflächengroßen Smartphone Display auch große, an Netbooks heranreichende Displaygrößen, wie sie bei Tablets zu finden sind, unterstützt.

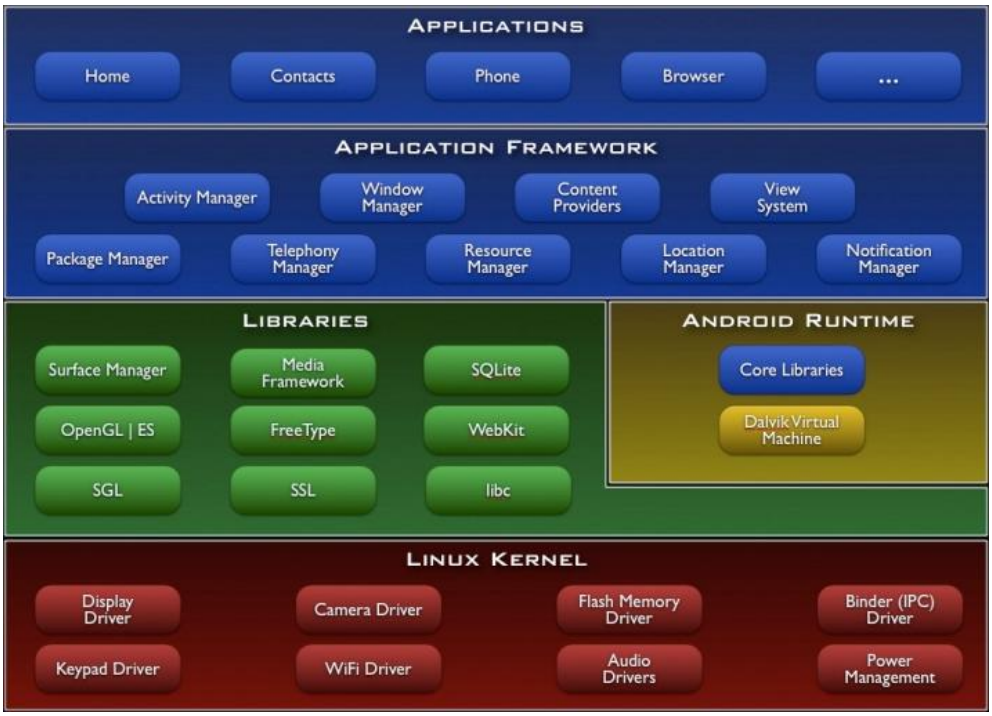
Verweis auf die Android-Developer-Seite

Bei den Recherchen und der Entwicklung der Applikation stellte die Android-Developer-Seite unter http://developer.android.com die zentrale Anlaufstelle dar. Der raschen Entwicklung der Plattform kann nur diese Online-Dokumentation in englischer Sprache gerecht werden. Die drei zentralen Rubriken *Design*, *Develop* und *Distribute* behandeln jeglichen Aspekt des Android-Frameworks und stammen als Primärquelle von dem Entwicklerteam selbst. Im Folgenden soll lediglich ein Einblick in das Android-Framework zum Grundverständnis der Funktionsweise der entwickelten Applikation gegeben werden.

5.2 System Architektur

Grundlegender Aufbau des Android-Frameworks

Abbildung 5-1 System Architektur des Android-Frameworks



Basis der Architektur

Die fünf Ebenen *Linux-Kernel*, *Libraries*, *Android-Runtime*, *Application-Framework* und *Applications* bilden das modulare Grundgerüst der Android Architektur. Als Basis dient dabei ein Linux-Kernel 2.6, welcher an die Gegebenheiten mobiler Hardware angepasst ist. Außerdem sind viele Treiber und Bibliotheken angepasst oder ausgetauscht, da ein normales Linux die mobile CPU zu stark belasten und den Akku zu schnell entleeren würde. Ohnehin ist die Notwendigkeit eines effizienten Energiemanagement Ursache vieler Besonderheiten des Android-Frameworks.

Jede Anwendung läuft unter Android als eigener Prozess, welche untereinander jedoch kommunizieren können. Diese Interprozesskommunikation (IPC) erfordert normalerweise einen hohen Aufwand an „Marshalling und Unmarshalling [...] – die auszutauschenden Daten müssen zwischen ihrer internen Repräsentation und einem zur Übermittlung geeigneten Format hin- und hergewandelt werden“ (heise mobile, 2012 S. 1). Mittels Shared Memory wird eine ressourcenschonende Variante benutzt, welche lediglich Referenzen auf existierende Objekte übermittelt. Ein virtuelles Speichermanagement vereinfacht dessen Implementierung und bietet darüber hinaus die Vorteile des zusammenhängenden Speicherbereichs, „der sich physikalisch über viele verstreute Speicherstellen verteilen kann“ (heise mobile, 2012 S. 2), sowie die Trennung und das Vermeiden der direkten Alloziation.

Dalvik virtuelle Maschine

Neben der Kapselung jeder Anwendung in einem eigenen Prozess, wird diese noch in einer eigenen virtuellen Maschine - der sogenannten Dalvik Virtual Machine (DVM) - ausgeführt. Die Programmierung von Android-Applikationen erfolgt in Java und der Compiler des Java-SDK erstellt im Kompilierprozess Java-Bytecode in Class-Dateien. Um unter der DVM zu laufen, wird dieser allerding noch in den DEX-Bytecode (Dalvik Executable Bytecode) umgewandelt.

Speicher

Eine installierte und gestartete Applikation läuft in einer Sandbox mit geschütztem Speicherbeich. Jede Anwendung verfügt über eine eigenes Verzeichnis mit im Ordner *data/data/* gefolgt vom eigenen Paketnamen. Als Dateisystem wird das YAFFS (Yet Another Flash File System) verwendet und als Datenbanksystem kommt SQLite zum Einsatz.

Ein weiterer sicherheitsrelevanter Aspekt ist das Anlegen eines eigenen Linux-Benutzers für jede Anwendung. Dies schränkt das Zugriffsrecht auf Anwendungsverzeichnisse von Außen ein und erhöht die Sicherheit der Android-Architektur.

5.3 Android SDK

Android Software-Development-Kit

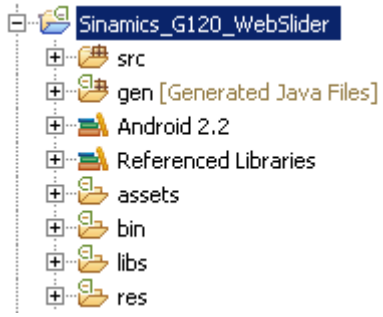
Das Android SDK ist ein von den Android-Entwicklern für die Erstellung von eigenen Android-Applikationen bereitgestelltes Softwarepaket. Es enthält alle notwendigen Tools, Bibliotheken, Compilern und Ressourcen zur effizienten Entwicklung von Android-Applikationen. Die Entwicklung mittels der IDE Eclipse bietet sich schon alleine auf Grund des ADT-Plugins (Android Development Tools) an. Dieses erweitert Eclipse zur komfortablen Entwicklungsumgebung mit den Erweiterungen zur einfachen Projektierung von Android-Projekten, notwendigen Compilern, Debug- und Monitoring-Tools. Zusätzlich liefert das ADT-Plugin den AVD-Manager (Android Virtual Device), welcher Emulatoren zum Testen und Debuggen erstellter Applikationen bereitstellt. Notwendige Pakete der Android APIs können im SDK-Manager heruntergeladen und installiert werden. Diese APIs dienen als Grundlage der Entwicklung eigener Apps.

5.3.1 Projektstruktur

Ordnerstruktur

Jedes Android-Projekt verfügt über eine festgelegte Ordnerstruktur, welche für den Kompilierprozess notwendig und im Folgenden erläutert ist. Jeder Ordner beinhaltet weitere Unterordner und Dateien, welche für den jeweiligen Zweck benötigt werden.

Abbildung 5-2 Ordnerstruktur eines Android-Projektes



Erläuterung der einzelnen Ordner

Tabelle 5-1 Aufgaben der Ordnerstruktur

Ordner	Aufgabe
`src`	Beinhaltet die vom Anwender erstellten Java-Quellcodedateien strukturiert innerhalb von Paketen
`gen`	Beinhaltet die automatisch durch das Android-SDK erstellte Klasse R, welche alle Referenzen auf die Ressourcen enthält
`assets`	Ermöglicht das Anfügen von externen Dateien, die zur erstellten APK-Datei angefügt werden
`bin`	Beinhaltet die erstellten Klassen und die aus dem Ordner `res` stammenden Ressourcen als gepacktes Archiv in einer APK-Datei
`libs`	Externe Bibliotheken werden in diesem Ordner abgelegt und automatisch im Kompilierprozess mit berücksichtigt
`res`	Beinhaltet alle selbst erstellten oder eingefügte Ressourcen – wie z.B. Layouts, Arrays oder Strings - welche während des Kompilierprozesses automatisch ins Projekt eingebunden werden

Manifest als globale Beschreibung der Applikation

In der sogenannten Manifest-Datei werden die wichtigsten Eigenschaften und Merkmale der Applikation zusammenfassend in einem XML-Dokument beschrieben. Dieses Dokument dient zum einen dem Betriebssystem vor der Installation zur Überprüfung von Kompatibilität wie Betriebssystemversion, Displaygröße oder benötigte Hardwarekomponenten. Zum anderen bietet es dem Anwender eine Übersicht über verwendete Betriebssystemfunktionen und dient als Vertrag zwischen App und Anwender betreffend des erlaubten Zugriffs auf Ressourcen wie z.B. Internet, Ortung und Kamera.

Der Google Play Store nutzt darüberhinaus hier definierte Eigenschaften der Applikation zum Filtern von Suchergebnissen.

5.3.2 Ressourcen

Trennung von User-Interface und Funktion

Ein wichtiges Merkmal des Android-Frameworks ist die strikte Trennung von User-Interface und Funktion. Dies wird dadurch erreicht, dass alle Elemente, die zur Darstellung von Inhalten dienen, in XML-Dateien

ausgelagert werden. Ähnlich der HTML-Syntax wird also das User-Interface vielmehr beschrieben und zur Laufzeit interpretiert, als pixelgenau festgelegt. Zudem bietet zum Beispiel das Auslagern des Layouts die Möglichkeit, das Design Experten zu überlassen, die nicht unbedingt Erfahrung in der Java Programmiersprache mitbringen müssen. Ein im SDK mitgelieferter grafischer Editor hilft zusätzlich mittels Drag&Drop von Anzeige- und Bedienelementen bei der Erstellung von Oberflächen.

Arten von Ressourcen

Alle Ressourcen werden in festgelegten Ordnern abgelegt. Folgende Tabelle zeigt die wichtigsten Ressourcenarten und deren Verwendung und Nutzen.

Tabelle 5-2 Ressourcenarten und deren Verwendung

Ressourcenart	Verwendung und Nutzen
`animator`	XML files that define property animations.
`anim`	XML files that define tween animations.
`color`	XML files that define a state list of colors.
`drawable`	Bitmap files (.png, .jpg, .gif) or XML files that are compiled into the drawable resource subtypes.
`layout`	XML files that define a user interface layout.
`menu`	XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu.
`raw`	Arbitrary files to save in their raw form.
`values`	XML files that contain simple values, such as strings, integers, and colors. Whereas XML resource files in other res/ subdirectories define a single resource based on the XML filename, files in the values/ directory describe multiple resources. For a file in this directory, each child of the <resources> element defines a single resource. For example, a <string> element creates an R.string resource and a <color> element creates an R.color resource. • arrays.xml for resource arrays (typed arrays). • colors.xml for color values • dimens.xml for dimension values. • strings.xml for string values. • styles.xml for styles.
`xml`	Arbitrary XML files that can be read at runtime by calling Resources.getXML().

(Google, 2012)

Die Referenzen aller Ressourcen werden während des Kompilierprozesses unter einer freiwählbaren, alphanummerischen ID in der Klasse R gespeichert. Über die Funktionen `getRessources().get<Ressourcenart>(id)` wird die entsprechende Ressource unter der angegebenen ID zurückgegeben.

Um verschiedenen Displaygrößen, -auflösungen, Pixeldichten, Sprachen, Betriebssystemversionen und Bildschirmorientierungen berücksichtigen

zu können und möglichst an das vorhandene Gerät angepasste Layouts, Grafiken und Dimensionen bereitzustellen, erlaubt das SDK die Erweiterung der Ordnernamen mit Kennzeichnern (Qualifier). Ein Beispiel wäre die Unterstützung verschiedener Sprachen durch das Anhängen von Qualifiern in dem Format

- `values-de` für deutsche und
- `values-en` für englische Texte.

Zur Laufzeit wählt Android dann automatisch die durch die Qualifier beschriebenen, passendsten Ressourcen aus dem Pool der bereitgestellten Ressourcen. Dies lässt sich auf alle Ressourcenarten anwenden und ermöglicht z.B. das Laden eines Bildes in hoher Auflösung bei hoher Displayauflösung oder das Laden von Texten in der jeweiligen Landesprache. Zu den wichtigsten Qualifier zählen die Folgenden.

Tabelle 5-3 Qualifier und deren Verwendung

Verwendung	Beispiele	Erläuterung
Sprache und Region	`de` `fr`	The language is defined by a two-letter ISO 639-1 language code

Minimale Breite	`sw<N>dp`
	
	`sw320dp`
	`sw600dp`
		The fundamental size of a screen, as indicated by the shortest dimension of the available screen area. Specifically, the device's smallestWidth is the shortest of the screen's available height and width.
		You can use this qualifier to ensure that, regardless of the screen's current orientation, your application's has at least <N> dps of width available for it UI.

Vorhandene Breite / Höhe	`w<N>dp`
	`h<N>dp`
	
	`w720dp`
	`h720dp`
	`w1024dp`
		Specifies a minimum available screen width / height, in dp units at which the resource should be used - defined by the <N> value.
		This configuration value will change when the orientation changes between landscape and portrait to match the current actual width.
		When your application provides multiple resource directories with different values for this configuration, the system uses the one closest to (without exceeding) the device's current screen width.

Bildschirm-größe	`small`
	`normal`
	`large`
	`xlarge`
		small: Screens that are of similar size to a low-density QVGA screen. The minimum layout size for a small screen is approximately 320x426 dp units.
		normal: Screens that are of similar size to a medium-density HVGA screen. The minimum layout size for a normal screen is approximately 320x470 dp units.
		large: Screens that are of similar size to a medium-density VGA screen. The minimum layout size for a large screen is approximately 480x640 dp units.
		xlarge: Screens that are considerably larger than the traditional medium-density HVGA screen. The minimum layout size for an xlarge screen is approximately 720x960 dp

Verwendung	Beispiele	Erläuterung
		units.
Screen orientation	`port` `land`	port: Device is in portrait orientation (vertical) land: Device is in landscape orientation (horizontal) This can change during the life of your application if the user rotates the screen.

Bildschirm Pixeldichte (Density)	`ldpi`
	`mdpi`
	`hdpi`
	`xhdpi`
	`nodpi`
	`tvdpi`
		ldpi: Low-density screens; approximately 120dpi.
		mdpi: Medium-density (on traditional HVGA) screens; approximately 160dpi.
		hdpi: High-density screens; approximately 240dpi.
		xhdpi: Extra high-density screens; approximately 320dpi.
		nodpi: This can be used for bitmap resources that you do not want to be scaled to match the device density.
		tvdpi: Screens somewhere between mdpi and hdpi; approximately 213dpi.

(Google, 2012)

5.3.3 Kompilierprozess und Debugging

Stufenweises kompilieren

Die Trennung von Funktion und User-Interface über XML, das Nutzen der Java Programmiersprache sowie die Lauffähigkeit innerhalb der Dalvik virtuellen Maschine erfordert einen komplexen Kompilierprozess. Für eine Android-Applikation erfolgt dieser in den vier Schritten:

1. *Android Resource Manager*
2. *Android Pre Compiler*
3. *Java Builder*
4. *Android Package Builder*

Erläuterung

Der *Android Resource Manager* hat die Aufgabe, jegliche Änderung an Dateien im `res` Verzeichnis zu überwachen. Er erstellt auf Basis der Ressourcen, Java-Programmcode, welcher in der Klasse `R` im Ordner `gen` gespeichert wird. Diese Klasse enthält Referenzen – also relative Adressinformationen – zu den jeweiligen angefügte Ressourcen, welche hierüber in der Applikation verwendet werden können.

Der *Java Builder* erstellt unabhängig vom Android-Framework aus den Java-Dateien die jeweiligen Class-Dateien. Dabei werden benötigte Klassen der angestrebten Version der Android-Bibliothek automatisch mit eingebunden.

Der *Android Package* Builder fasst die vom Resource Manager, Pre Compiler und Java Builder erstellten und kompilierten Dateien zu einem Paket zusammen. Das Paket trägt die die Endung *.apk* und kann auf allen kompatiblen Android-Geräten installiert werden. Es enthält somit alle

notwendigen Dateien, die zur Lauffähigkeit der Applikation benötigt werden.

DDMS - Dalvik Debug Monitor Server

Der DDMS bietet eine in Eclipse integrierte Möglichkeit, ein angeschlossenes Gerät oder Emulator zu analysieren. Es bietet einen Prozess- und Speicherressourcenmanager, Dateiexplorer und die Funktion zur Interaktion mit dem Gerät z.B. zur Simulation eines Anrufs. Zu Debugging- und Diagnosezwecken eignet sich die *LogCat* Ausgabe, in welche alle Meldungen zur Laufzeit einer Applikation ausgegeben werden.

5.4 Aufbau und Struktur einer Applikation

Die grundlegenden Applikationstypen

Android unterscheidet grundlegend die drei verschiedenen Applikationstypen

- Activity,
- *Service und*
- *Content Provider.*

Hierbei stellt die *Activity* als einziger Typ eine auf dem Bildschirm sichtbare Applikation dar, mit derer der Benutzer direkt interagieren kann. Die Activity ist daher die wichtigste und komplexeste Form und soll im Folgenden in einem eigenen Kapitel näher betrachtet werden.

Ein *Service* läuft im Gegensatz zu der Activity ausschließlich im Hintergrund und besitzt kein User-Interface, um mit dem Benutzer zu interagieren. Ein Service stellt - wie es der Name schon vermuten lässt - eine Dienstleistung für eine oder mehrere laufende Activitys dar.

Beispiele hierfür wären ein Wetter-Service, welcher stündlich die neusten Wetterdaten aus dem Internet lädt, verarbeitet und in geeigneter Form zur Verfügung stellt. Laufende Activitys, die auf solche Wetterdaten zugreifen möchten, können dann diesen Service nutzen. Der Service stellt die Daten also nur zur Verfügung, ohne sie selbst darzustellen.

Als dritter Typ kann ein *Content Provider* entwickelt werden. Dieser dient als Datenspeicher und Datenprovider und kann so für verschiedene Activitys und Services gleichzeitig und persistent Daten bereitstellen.

5.4.1 Activitys

Übersicht

Activitys bilden das Grundgerüst einer jeden Applikation. Sie verbinden im Softwaremodell des Model-View-Control (MVC) alle drei Komponenten miteinander. Jede Activity interagiert mit dem *Model* (Informationsspeicher in Form von Datenbanken oder Arrays), verfügt über ein eigenes *View* (in XML definiertes UI) und implementiert den *Control* durch Abfrage von Benutzereingaben über den Bildschirm.

Jede Activity läuft in einem eigenen Thread. Entsprechend des auf Sicherheit bedachten Android-Frameworks, erhöht dies zwar die Nutzbarkeit im Fehlerfall und Absturzsicherheit des Betriebssystems, schränk aber die Kommunikation und den Datenaustausch zwischen zwei parallel laufenden Activitys stark ein. Abhilfe schaffen die sogenannten Intents – Anfragen zwischen Activitys. Hierbei schickt eine Activity einen Intent gezielt an eine andere Activity oder als Broadcast ab. Dem Intent können neben dem Hauptzweck (z.B. Starten einer Activity) auch Daten in einer Hash-Map mittels `pull` angefügt bzw. mittels `push` entnommen werden. Um eine Klasse als Activity zu deklarieren, genügt die Ableitung von der Superklasse `Activity`.

Lebenszyklus einer Activity

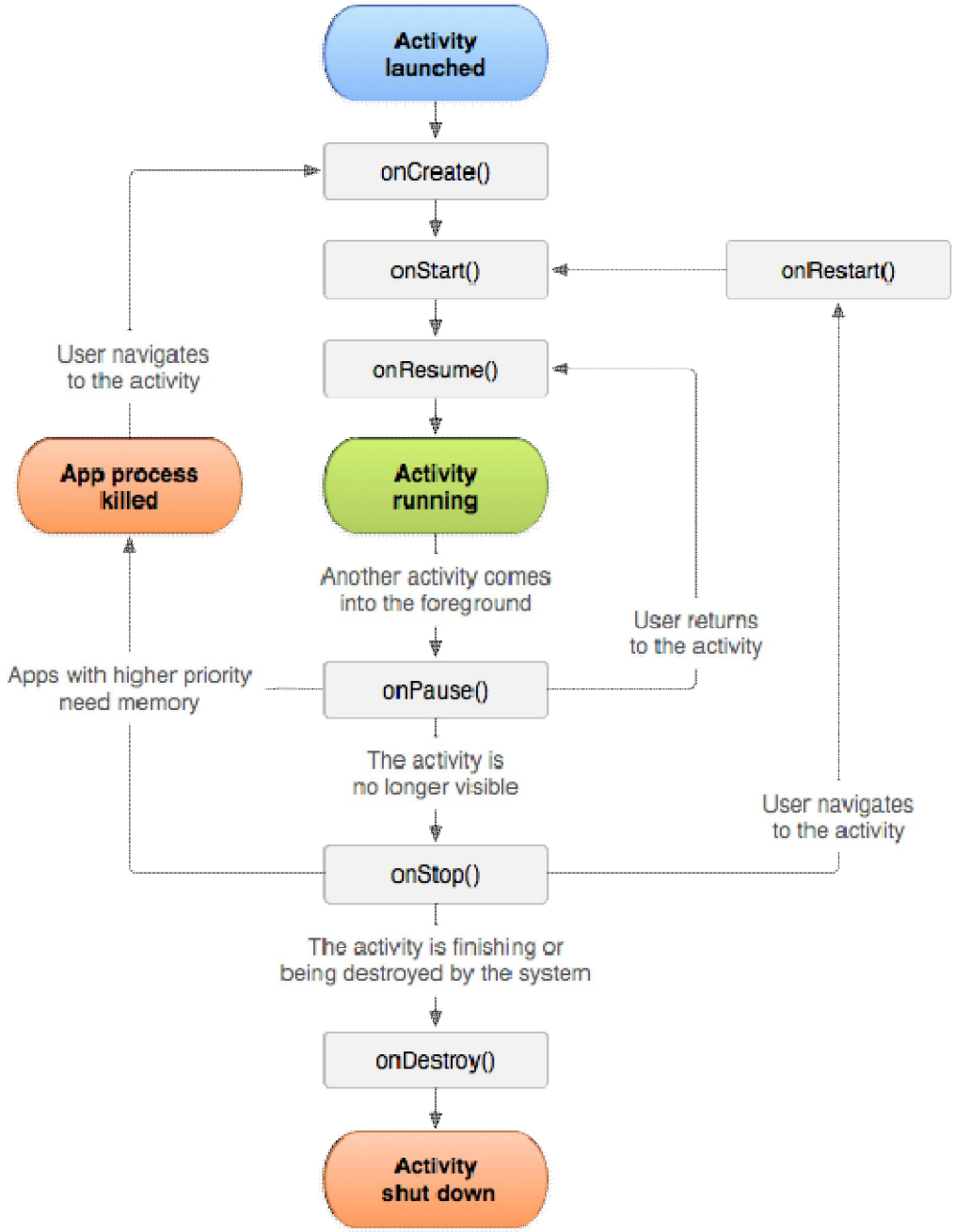
Ein wichtiger Aspekt bei der Implementierung von Activitys ist der Lebenszyklus. Dieser beschreibt den Zustand einer Activity und Übergänge zwischen diesen. Übergänge werden mittels Events vom Betriebssystem initiiert und können vom Programm abgefangen und eine entsprechende Verarbeitung angestoßen werden.

Ein derartiger Aufwand ist notwendig, da ein Activity nicht als beendet gilt, wenn es nicht mehr sichtbar ist. Vielmehr läuft der Activity-Prozess solange weiter, bis belegte Arbeitsspeicherressourcen für andere Prozesse benötigt werden. Erst dann wird der Prozess komplett beendet. Dies ermöglicht ein schnelles Wiederstarten und schont die CPU Auslastung.

Eine Anwendung für mobile Geräte unterliegt - wie im Kapitel 5.1 erläutert - im Gegensatz zu Desktopanwendungen einer Vielzahl von Störquellen. Um auf diese adäquat reagieren zu können, bietet das Android-Framework sieben Events an, welche zu ganz bestimmten Zustandsübergänge ausgelöst werden.

Das wichtigste Event ist `onCreate()`, welches beim Initialstart des Activitys und nach Beenden des Activity-Prozesses beim Neustart aufgerufen wird und grundsätzliche Dinge wie beispielsweise die Zuordnung eines Layouts übernimmt. Unmittelbar vor und nach dem Sichtbarwerden der Activity werden die Events `onResume()` und `onPause()` aufgerufen und dienen prinzipiell zur Zwischenspeicherung von Zuständen und Daten im Falle eines anderen, in den Vordergrund tretenden Activitys. Eine vollständige Übersicht des Lebenszyklus zeigt das folgende Diagramm aus dem Developer-Guide.

Abbildung 5-3 Lebenszyklus einer Activity



(Google, 2012)

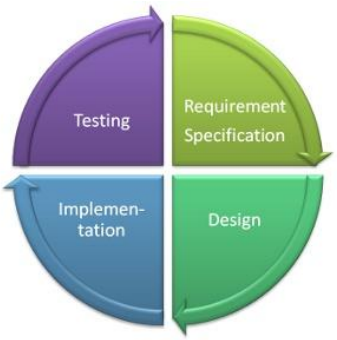
6 Entwicklungsprozess der Applikation AndroHmiS7

Vorgehensmodell Inkrementelle Softwareentwicklung

Als Vorgehensmodell wurde bei der Entwicklung der Applikation die inkrementelle Softwareentwicklung gewählt. Zu Beginn steht ein „attraktives Teilsystem mit den Phasen Planung, Entwicklung, Integration und Qualitätssicherung“ (Lux, 2008 S. 18). Das System wird anschließend mit jeweiligem Durchlaufen der vier Phasen schrittweise erweitert. Der „Kunde ist bei Planung und Qualitätssicherung jeder Erweiterung beteiligt“ (Lux, 2008 S. 18). Die Vorteile bei diesem Modell liegen darin, dass der Termindruck nicht die Qualität, sondern den Funktionsumfang reduziert. Da die inkrementelle Entwicklung in häufigen Lieferterminen neuer Versionen resultiert, lässt sich die genaue Entwicklung anfangs nicht vorhersehen. Es können lediglich Prioritäten vergeben werden.

(Lux, 2008)

Abbildung 6-1 Modell der inkrementellen Softwarenentwicklung



Innerhalb jedes Inkrements werden die aufgezeigten Phasen durchlaufen. Die Phasen des ersten Inkrements, welche in dieser Bachelor-Thesis den Rahmen der praktischen Arbeit bilden, sollen in diesem Kapitel beschrieben und erläutert werden.

6.1 Anforderungen ermitteln

6.1.1 IST-Analyse

Momentane Systeme und die Entscheidung Kaufen / Mieten / Entwickeln

Eine Analyse der bestehenden Remote-Zugriffsmöglichkeiten auf Prozessdaten einer Siemens SIMATIC SPS ist im Kapitel 3.1 behandelt.

Diese hat ergeben, dass über den Web-Server oder Sm@rt Access bereits Tools und Funktionen existieren und genutzt werden. Eine auf eine mobile Plattform wie Android zugeschnittene Lösung existiert bislang nicht und könnte mit Blick auf das günstige Verhältnis zwischen Aufwand/ Kosten und Funktionsumfang/ Komplexität, der raschen Technologieentwicklung mobiler Systeme sowie dem innovativen Charakter eine App eine Nische ausfüllen (vgl. Kapitel 3.2). Aus diesem Grund fiel die Entscheidung von Beginn an auf die eigenständige

Entwicklung mit den Vorteilen der passgenauen Lösung und flexiblen Funktionalität.

Projektteam

Entsprechend der Vorgabe der Bachelor-Thesis, welche ein eigenständiges Vorgehen fordert, entwickelte ich die Software nicht in der Umgebung eines Teams sondern als selbstständiges Projekt. Allerdings flossen Erfahrungen andere Teammitglieder hilfreich in die Thesis mit ein. Der experimentelle Charakter dieser Arbeit und speziell der Entwicklung der Applikation führte zudem dazu, dass die klassische Auftraggeber/ Auftragnehmer Definition nicht zum tragen kam, sondern eine prototypische Realisierung ohne direkten Kundenauftrag angestrebt wurde.

6.1.2 Entwicklungsumgebung

Integrated Development Environment

Als IDE (Integrated Development Environment) wurde das für Java populäre Tool Eclipse in der Version Indigo verwendet. Neben den üblichen für eine IDE typischen Merkmale wie

- Quelltexteditoren für gängige Programmiersprachen,
- Syntaxerkennung und Farbhervorhebung von Schlüsselwörtern,
- automatische Formatierung,
- Projektnavigator, Projektmanager und
- integrierter Compiler und Debugger

(Moecker, 2012)

bietet es durch Ergänzungen (Plugins) weitere Features, welche auf die Arbeit mit dem Android-SDK zugeschnitten sind und die Entwicklung einer Applikation deutlich vereinfachen. Hierzu zählt das ADT-Plugin, welches neben den notwendigen Compilern (vgl. Kapitel 5.3.3) auch Diagnosetools zur Analyse von laufenden Applikationen auf realen Android-Geräten oder Emulatoren bietet.

Emulatoren spielen bei der Entwicklung indes eine wichtige Rolle. Sie bilden ein Android-Gerät auf dem Entwicklungsrechner mit allen Funktionen und übereinstimmender Benutzerschnittstelle nach. Dadurch dienen sie als Testplattformen für Geräte mit unterschiedlicher Betriebssystemversionen, Hardwaremerkmalen und Displaygrößen.

6.1.3 Anforderungsbeschreibung - Vorstudie

Zielbestimmung und Produkteinsatz

Ziel ist die prototypische Entwicklung einer auf dem Android-System basierenden Applikation zur Interaktion mit Prozessdaten auf einer Siemens SIMATIC SPS. Als Einsatzmöglichkeiten bieten sich jegliche Szenarien, in denen ein mobiler Zugriff auf Prozessdaten einer SPS innerhalb einer Automatisierungsanlage sinnvoll ist. Anwendungsfelder wären die Inbetriebnahme, Wartung, Diagnose, Hausautomatisierung

oder auch die Steuerung von Modellaufbauten wie z.B. auf Messen (vgl. Kapitel 3.2.1).

Produktübersicht

Die Applikation soll eine performante Möglichkeit bieten, über ein mobiles Android-Gerät und unter Nutzung der WLAN-Schnittstelle mit den Prozessdaten einer SPS zu interagieren. Hierbei ist ein lesender und ein schreibender Zugriff möglich. Unterstütze SIMATIC Produkte sind die S7-300/400 und die WinAC auf einem IPC wie Microbox oder Nanobox.

Tabelle 6-1 Kommunikationsstruktur und Teilnehmer

SPS	*Netzwerk*	*Android-Gerät*
 S7-300  Microbox  Nanobox  S7-400	 Switch  WLAN Access-Point	 Android Smartphone  Android Tablet

Produktfunktionen

Innerhalb der Applikation sollen drei Teilfunktionen realisiert werden, welche in Ihrem Funktionsumfang ausbaufähig und erweiterbar konzipiert sind.

- Bilderverwaltung
  - *Auswahlmenü:* Erstellte Bilder können über eine Liste bearbeitet, gelöscht und ausgeführt werden.
  - *Editor:* Bilder können über einen integrierten Editor neu erstellt und vorhandene bearbeitet werden. Anzeige- und Bedienelemente

werden hierbei mittels Drag&Drop frei angeordnet und ihre Eigenschaften und Verbindung zu den SPS Prozessdaten wie gewünscht parametriert. Ein Raster zur Ausrichtung der Elemente sowie eine Bearbeitungshistorie zum Nachverfolgen der Änderungen erleichtern das bearbeiten.

- *Runtime:* Die Runtime führt ein erstelltes Bild aus. Es wird eine Verbindung zur SPS hergestellt und die parametrierten Prozessdaten werden zyklisch eingelesen oder geschrieben. Bedienelemente können im bekannten Android Look&Feel bedient und Anzeigeelemente eingesehen werden.

- Diagnose-Panel
  Das Diagnose-Panel dient zur Abfrage des Diagnosepuffers und zum Steuern des Betriebszustandes der SPS.
- Web-Server Zugriff
  Ein integrierter Web-Server Zugriff soll - sofern in der SPS unterstützt und aktiviert - die Verbindung mit dem Web-Server vereinfachen und sich in das Look&Feel der Applikation integrieren.

Produktdaten, -leistungen und Qualitätsanforderungen

Langfristig zu speichernde Daten sind die erstellten Bilder sowie vom Benutzer getätigte Applikations-Einstellungen wie z.B. die Wahl der IP-Adresse.

Zu den nichtfunktionalen Leistungen zählen die performante und stabile Ausführung der Applikationen, die Kompatibilität auf allen unterstützten Android Betriebssystemversionen ab der Version 2.1 (API 8) sowie die funktionale Erweiterbarkeit. Unter *stabil* ist zu verstehen, dass bei einem Fehler aussagekräftige Fehlerinformationen ausgegeben und im Absturzfall nur das entsprechende Modul betroffen ist. Mit *performant* wird ein UI mit geringen Reaktionszeiten angestrebt.

Im Bereich der zyklischen Abfrage und dem Zugriff auf die Prozessdaten der SPS soll eine möglichst geringe Verzögerung, hoher Datendurchsatz und performante Abfrage im Millisekundenbereich erreicht werden. Sicherheitsrelevante Aspekte wie die Verbindungssicherheit über das WLAN Netzwerk und dessen Verschlüsselung sowie die Unterbindung nicht deterministischer Schreibzugriffe auf die Prozessdaten fordern höchste Qualitätsstufen.

Zu den weiteren Qualitätsanforderungen zählen die an dem Android Look&Feel angelehnte Oberfläche und Benutzbarkeit sowie die Erfüllung von Erwartungen an Bedienung und Verhaltensweise der Applikation. Die im Bereich der mobilen Applikationsentwicklung geforderte defensive Nutzung der Prozessor-, Arbeitsspeicher- und vor allen Dingen Energieressourcen spielen ebenfalls eine zentrale Rolle.

6.1.4 UML Modelle

Anwendungsfallmodell – Use-Case-Modell

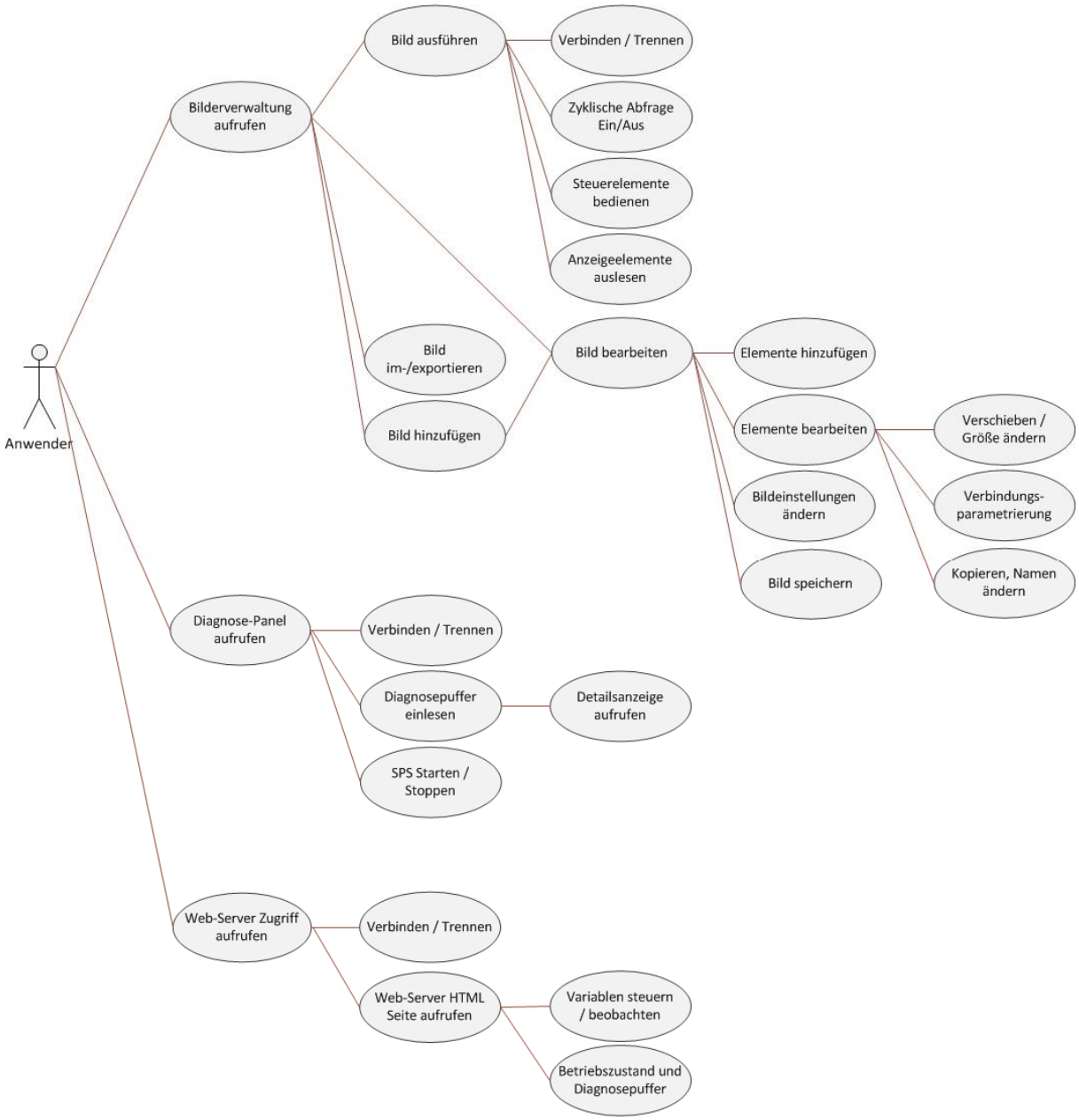
UML Anwendungsfalldiagramme modellieren Systemoperationen und bestehen aus

- Aktoren, welche die Anwendung benutzen und
- Anwendungsfälle, welche die von der Anwendung angebotenen Operationen beschreiben.

(Lux, 2011)

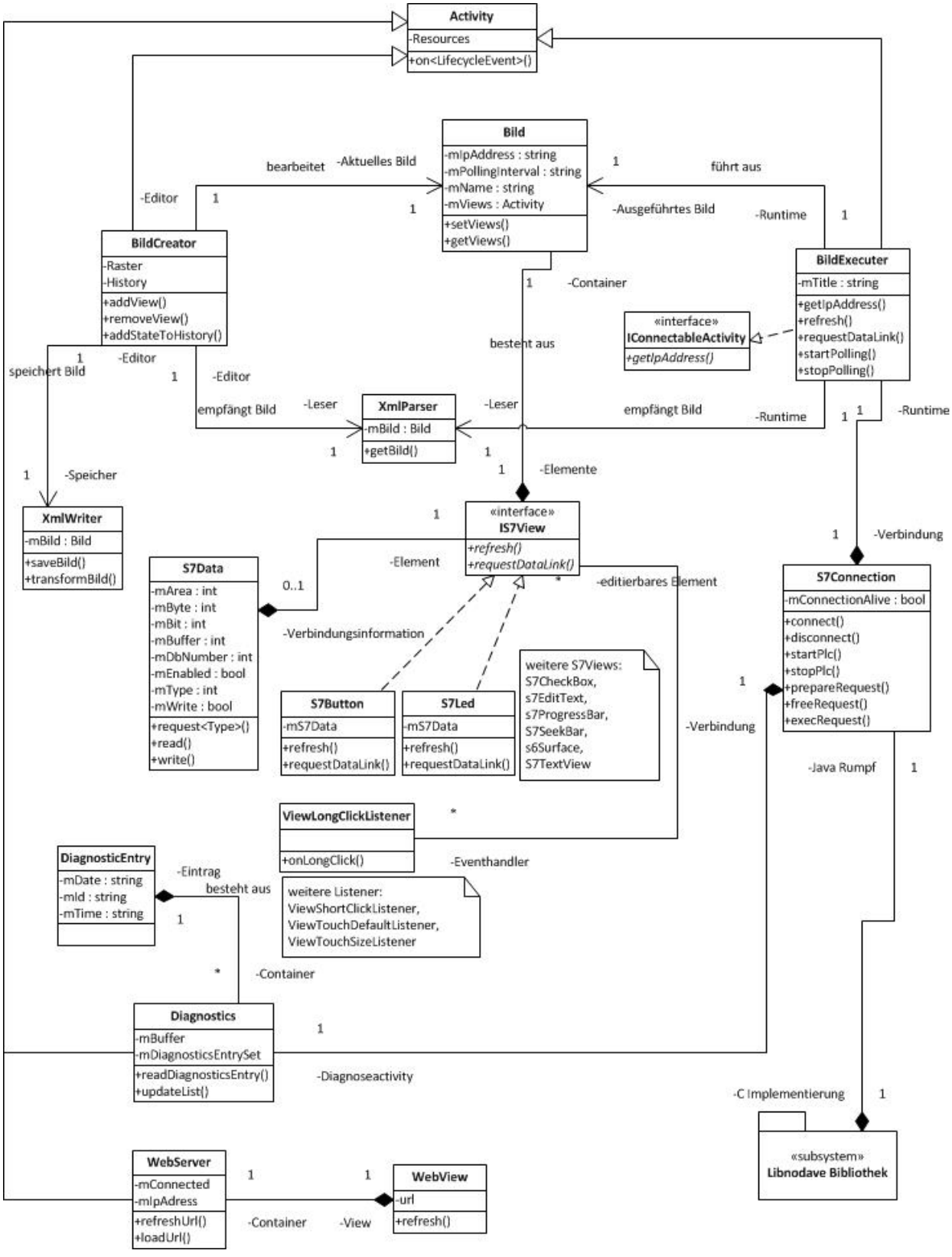
Sie „beschreiben die (geplante) Funktionalität eines Systems und damit dessen Verhalten gegenüber der Außenwelt“ (Michael Richter, 2007).

Abbildung 6-2 Anwendungsfallmodell der Applikation AndroHmiS7



Problembereichsmodell – Klassendiagramm

Abbildung 6-3 Klassendiagramm der Applikation AndroHmiS7

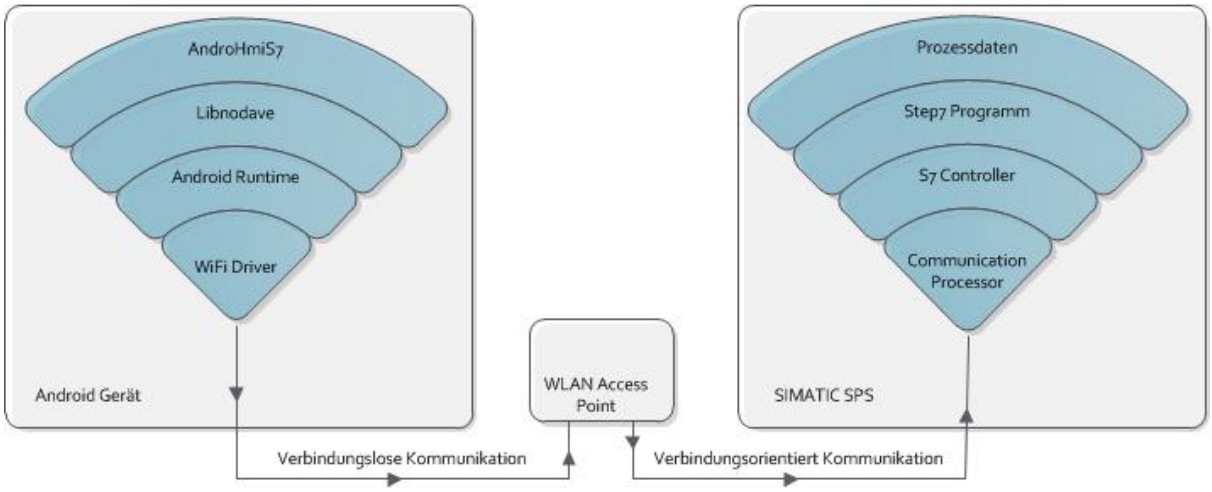


Erläuterung

Das oben abgebildete Klassendiagramm deckt nicht die vollständige Anzahl an Klassen und Verbindungen untereinander ab, sondern soll reduziert die wesentlichen Aspekte ausgehend von der Superklasse *Activity* verdeutlichen. Wichtig ist hierbei die zentrale Rolle der Klasse *Bild*, welche von der Klasse *BildCreator* zum Editieren und *BildExecuter* zum Ausführen verwendet wird. Zudem dient sie als Grundlage für die Speicherung mittels der Klasse *XmlWriter* und zum Laden über die Klasse *XmlParser*. Die Anzeige- und Bedienelemente vom Typ *ISView* bilden eine Aggregation mit *S7Data,* in welcher alle Verbindungsinformationen gespeichert sind. S7Connection verfügt über Verbindungsmethoden und wird daher von *Diagnostics* und *BildExecuter*, also der Runtime und des Diagnose-Panels verwendet.

Schnittstellemodell - Systemschnittstellen

Abbildung 6-4 Systemschnittstellen zwischen der Applikation und den Prozessdaten



6.2 Analyse

6.2.1 Pflichtenheft

Als Grundlage für das Pflichtenheft steht die Vorstudie, welche aufbauend auf die dort getroffenen Entscheidungen und Erfahrungen erweitert und konkretisiert wird.

Benutzeroberfläche

Um ein einheitliches Look&Feel aller Applikationskomponenten zu erreichen, sollen alle Designelemente wie Farben, Abstände und Schriftarten zentral definiert und von allen Komponenten eingebunden werden. Am oberen Rand jeder, den vollständigen Bildschirm ausfüllenden UI-Seite, soll eine *Action-Bar* Informationen zur aufgerufenen Komponente und Buttons zum Schnellzugriff auf die wichtigsten Funktionen bereitstellen. Der Hardware-Menü-Button eröffnet weitere Interaktionsmöglichkeiten über den Aufruf eines Menüs. Dabei

soll die Zahl der Menüeinträge und Funktionen auf ein Minimum reduziert sein.

Generell sollen für Auswahlmöglichkeiten oder zur Vorwärts-Navigation Listen verwendet werden. Die Rückwärts-Navigation erfolgt über den Hardware-Back-Button. Für Eigenschaften oder elementspezifische Funktionen sollen Kontext-Menüs verwendet werden.

Als Anzeige- und Bedienelemente kommen Standard-Android-Elemente zum Einsatz. Dies erhöht den Widererkennungswert bekannter Bedienstrukturen und fügt sich in das Betriebssystemdesign ein. Aus der Analyse bestehender HMI-Software – sowohl Editor als auch Runtime – ergeben sich die Grundanforderungen an die Applikation (vgl. Kapitel 2.2.3). Die benutzerfreundlichen Eingabemöglichkeiten über einen Touch-Screen eröffnen dabei neue Interaktionsmöglichkeiten, welche es auszunutzen gilt.

Gliederung in Teilprodukte mit Prioritäten

Hohe Priorität: Runtime, Bilderverwaltung

Mittlere Priorität: Editor

Niedrige Priorität: Diagnose-Panel, Web-Server Zugriff

Die hoch priorisierte Runtime bildet den Teil der Applikation, welcher letztendlich die wichtigsten Aufgaben der Verbindung, zyklischen Abfrage, Anzeige und Steuerung von Prozessdaten bereitstellt.

Die Trennung von Runtime und Editor eröffnet die Möglichkeit zur Unterstützung anderer Automatisierungssysteme z.B. durch Nutzen des vorhandenen Editors und Anpassen der Runtime.

6.2.2 Statisches Modell

Strukturmodell

Die objektorientierte Programmiersprache Java ermöglicht eine modulare Softwarestruktur durch die Verwendung von Paketen, Klassen und Objekten. Dies fördert die Widerverwendbarkeit und erleichtert die Wartung, Dokumentation und Pflege des Quellcodes. Auch bei der erstellten Applikation *AndroHmiS7* kam dieses grundlegende Prinzip von Java zum Einsatz. Klassen mit ähnlichen oder verwandten Aufgaben werden in gleichen Paketen verwaltet.

Tabelle 6-2 Einteilung in Pakete

Paket	Verwendung
com.androhmis7.activities	Alle verwendeten Activitys wie Bilderverwaltung, Runtime, Editor, Diagnose-Panel und Web-Server Zugriff
com.androhmis7.connection	Klassen zum Verbindungsauf- und abbau sowie zur zyklische Abfrage der Prozessdaten
com.androhmis7.constants	Konstanten - entsprechend ihrem Verwendungszweck in eigenen Klassen
com.androhmis7.dialogs	Dialoge zur Interaktion mit dem Anwender und

Paket	Verwendung
	Kontext-Menüs
com.androhmis7.listeners	Ereignisverarbeitende Klassen
com.androhmis7.preferences	Einstellungen für jede Activity
com.androhmis7.s7views	Anzeige- und Bedienelemente
com.androhmis7.s7views.interfaces	Schnittstellen für S7Views
com.androhmis7.s7views.util	Hilfsfunktionen für S7Views
com.androhmis7.util	Allgemeine Hilfsklassen für die Anwendung
com.androhmis7.xml	XML Schreib- und Leseklassen zur Speicherung und Empfangen in bzw. von XML-Dateien

Bei der Implementierung der Klassen soll darauf geachtet werden, dass eine zentrale Aufgabe oder ein zentraler Nutzen im Vordergrund steht und Seiteneffekte oder eine unerwartete Funktionalität möglichst vermieden werden.

Activitys

Activitys spielen die zentrale Rolle als Bindeglied zwischen Funktion und User-Interface. Für jeden Funktionsbereich der Applikation mit eigenem UI wurde eine eigene Activity entwickelt. So erhöht sich die Modularisierung des Projektes und ermöglicht die Kapselung und Trennung von Aufgabenbereichen. Als Beispiel sei die Activity *BildCreator* zu nennen. Diese wird bei der Erstellung eines neuen Bildes sowie beim Editieren eines bereits vorhandenen verwendet. Alle Activitys sind von der Superklasse Activity abgeleitet, welche die Grundfunktionalität und die Lebenszyklusevents bereits bereitstellt.

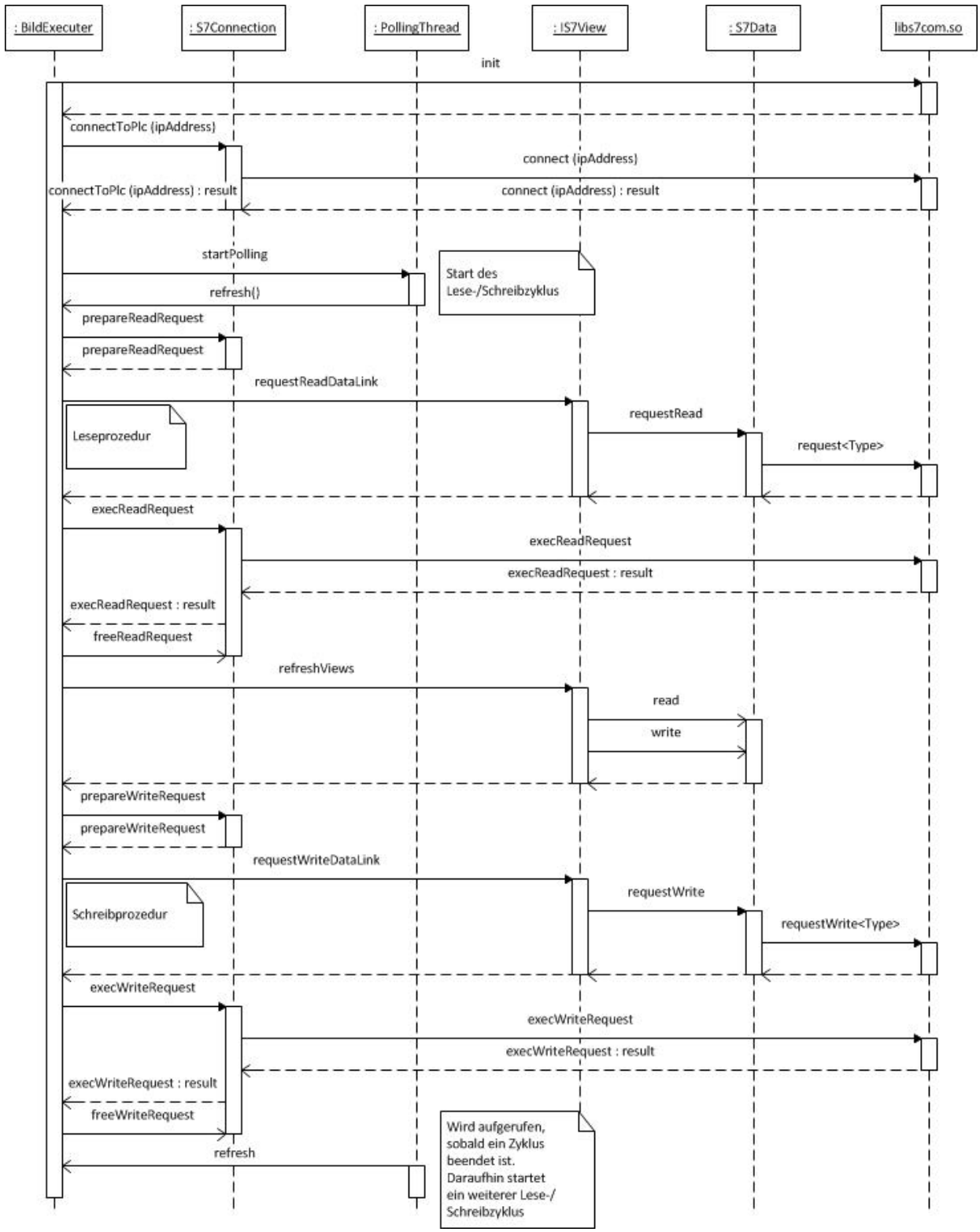
Tabelle 6-3 Ausführbare Activitys

Activity	Verwendung
ActivitySwitcher	Auswahl zum Starten der jeweiligen Hauptactivitys Bilderverwaltung, Diagnose-Panel und Web-Server Zugriff
BildCreator	Editor zum Bearbeiten eines Bildes
BildExecuter	Runtime zum Ausführen eines Bildes
BildSelector	Bilderverwaltung zum Managen, Starten und Bearbeiten bestehender Bilder
Diagnostics	Diagnose-Panel zum Zugriff auf den Diagnosepuffer und zum Steuern der SPS
StartScreen	Startbildschirm - aufgerufen beim Starten der Applikation
WebServer	Zugriff auf den Web-Server der SPS

6.2.3 Dynamisches Modell

Sequenzdiagramm – Zyklischer Zugriff auf Prozessdaten

Abbildung 6-5 Sequenzdiagramm – Zyklischer Zugriff auf Prozessdaten



Erläuterung

Das oben stehendes Sequenzdiagramm verdeutlicht die Kommunikation zwischen der Applikation und der Bibliothek unter Einbeziehung der wichtigsten Klassen *BildExecuter*, *S7Connection*, *PollingThread*, *IS7View* und *S7Data* auf der Java Seite und dem Interfacemodul innerhalb der *libs7com*.so Bibliothek auf der anderen. Wichtig hierbei ist die schrittweise Ausführung einer Lese bzw. Schreibanfrage in den Schritten *Prepare*, *Exec* und *Free* (vgl. Kapitel 4.3.2).

6.3 Entwurf

6.3.1 Grundsatzentscheidungen

Datenhaltung

Die Datenhaltung soll unter Nutzung des XML-Formates erfolgen. Zum einen bietet Android bereits intern eine komfortable Möglichkeit der Speicherung in XML. So nutzt das Framework diese Technik eingehend bei der Erstellung der Oberflächen und der Speicherung von applikationsinternen Einstellungen. Zum anderen ist das XML-Format plattformübergreifend kompatibel und anerkannt. So bereitet beispielsweise ein Browser die XML-Daten visuell auf und ermöglicht unter Nutzung von XSLT (Stylesheets für XML-Dokumente) eine Formatierung und ersetzt eine notwendige Umwandlung in ein visuell darstellbares Format.

Benutzeroberfläche

Tabelle 6-4 Benutzeroberflächen der einzelnen Komponenten

Komponente	Emulator-Screenshot	Beschreibung
Auswahlmenü	*Abbildung 6-6 Auswahlmenü* 	Ermöglicht die Navigation zu den drei Hauptaufgaben-bereichen: *Bilderverwaltung*, *Diagnose-Panel* und *Web-Server Zugriff.*

Komponente	Emulator-Screenshot	Beschreibung
Bildverwaltung	*Abbildung 6-7 Bilderverwaltung* 	Listet alle vorhandenen Bilder auf und bietet die Ausführung in der *Runtime*, Bearbeitung im *Editor* sowie Kopier- und Löschfunktionen an.
Diagnose-Panel	*Abbildung 6-8 Diagnose-Panel* 	Stellt die Oberfläche zur Abfrage des Diagnosepuffers sowie Steuermöglichkeiten der SPS bereit.

Komponente	Emulator-Screenshot	Beschreibung
Web-Server Zugriff	*Abbildung 6-9 Web-Server Zugriff* 	Dient zum Zugriff auf den SIMATIC Web-Server und greift auf die von der SPS bereitgestellte Standard-HTML Seite oder User-Defined-Webpages zu.
Editor	*Abbildung 6-10 Editor* 	Bedienoberfläche zum Bearbeiten eines Bildes. Die Elemente können beliebig angeordnet, verändert und einer dynamischen Toolbox entnommen werden.

Komponente	Emulator-Screenshot	Beschreibung
Runtime	*Abbildung 6-11 Runtime* 	Die im Editor erstellen Bilder werden hier ausgeführt und alle Anzeige- und Bedienelemente nach Herstellung einer Verbindung zur SPS zyklisch mit den Prozessdaten synchronisiert.

6.3.2 Detailentwurf

MVC – Model View Control

Das User-Interface (*View*) wird vom Informationsobjekt (*Model*) getrennt und dynamisch beim Starten der Applikation bzw. des Activitys geladen. So wird beispielsweise einer Activity (hier repräsentiert durch `this`) in der `onCreate()` Methode ein Layout aus dem Ressourcenpool zugewiesen.

```
1. this.setContentView(R.layout.bild_selector);
```

Zudem können zur Laufzeit vorab definierte Ressourcen geladen und als Java-Objekte verwendet werden. So werden im folgenden Beispiel Layout-Paramater wie Höhe und Breite eines `S7Button` geladen (vgl. Kapitel 5.3.2).

```
1. this.setLayoutParams(new LayoutParams((int) context.getResources()
2.          .getDimension(R.dimen.s7_button_default_width), (int) context
3.          .getResources().getDimension(R.dimen.s7_button_default_height)));
```

Die Steuerung (*Control*) wird von Listener-Objekten übernommen, welchen Elementen des User-Interfaces zum Verarbeiten eines Ereignisses dynamisch zugeordnet werden können. Hier wird beispielsweise einem `View` ein `OnTouchListener` zur Verarbeitung von Touch-Events zugeordnet.

```
1. view.setOnTouchListener(mViewTouchDefaultListener);
```

Der verwendete `OnTouchListener` implementiert dann die eigentliche Event-Methode `onTouch()` und kann auf verschiedene Eventarten reagieren.

```
1.  public boolean onTouch(View contextView, MotionEvent event) {
2.      action = event.getAction();
3.      switch (action) {
4.      case MotionEvent.ACTION_DOWN:
5.          break;
6.      case MotionEvent.ACTION_MOVE:
7.          break;
8.      case MotionEvent.ACTION_UP:
9.          break;
10.     }
11. }
```

Dieses Vorgehen der Trennung von *Model, View und Control* zieht sich durch das gesamte Projekt und kommt immer dann zum Einsatz, wenn das Layout statisch und vorab bekannt ist sowie auf Benutzereingaben reagiert werden muss. Es soll ebenfalls zur Modularisierung und Wiederverwendbarkeit des Codes beitragen.

Threads

Rechenintensive und langwierige Prozesse oder Methoden sollten in Threads ausgelagert werden. Dies ermöglicht die Trennung vom UI-Thread und sichert damit die Reaktionsfähigkeit des User-Interfaces und verhindert dessen Einfrierung. In der Applikation *AndroHmiS7* sind aufgrund des Zeitbedarfs die Fälle

- Verbindungsherstellung zur SPS,
- zyklische Abfrage und Zugriff auf die Prozessdaten und
- Abfrage des Diagnosepuffers,

jeweils in eigene Threads auszulagern.

6.4 Implementierung

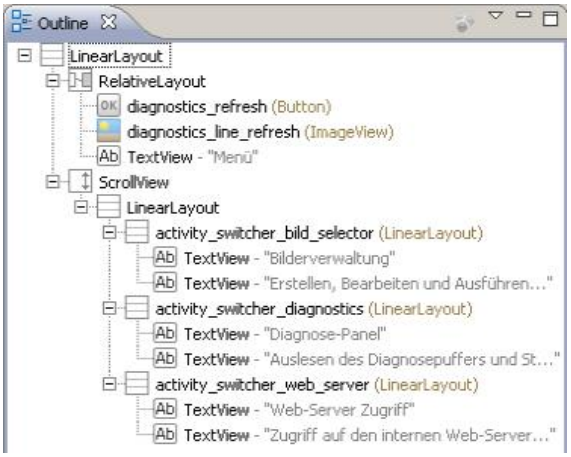
6.4.1 User-Interface

XML Layouts

Entsprechend der Vorgabe des Android-Frameworks wurde bei der Implementierung des User-Interface darauf geachtet, die vorgeschlagene Trennung von Funktion und UI im Rahmen des MVC anzuwenden. Alle statischen, sichtbaren Elemente sind daher in XML-Dateien definiert. Diese nutzen eine verschachtelte Struktur basierend auf Tags. Tags sind Schlüsselwörter, welche im Falle eines XML-Layouts, UI-Elemente mit Eigenschaften darstellen. Grundsätzlich basiert jedes Layout auf ein `LinearLayout`, welches die Elemente entsprechend der in der XML-Datei definierten Reihenfolge auf den Bildschirm platziert. Jegliche von Android vorgegebene Anzeige- und Bedienelemente – wie Buttons, Textausgabefelder und weitere Layouts - können dann verschachtelt in dieses `LinearLayout` eingefügt werden. Als Beispiel ist ein *Outline*

(Strukturübersicht) des zu Beginn startenden Auswahlbildschirms aufgezeigt.

Abbildung 6-12 XML-Outline des Auswahlbildschirms



Jegliches Element wird im Stil `<Elementname />` oder `<Elementname></Elementname>` angefügt. Innerhalb der spitzen Klammern können Eigenschaften dieses Elementes festgelegt werden. Der entstandene Rumpf bietet Platz für weitere Elemente. Am Beispiel des `TextViews` „Bilderverwaltung“ soll dies erläutert werden.

Abbildung 6-13 Definition des TextViews „Bilderverwaltung“

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="@dimen/activity_layout_button_height"
    android:drawableLeft="@drawable/ic_folder_78"
    android:drawablePadding="@dimen/activity_header_button_drawable_margin"
    android:gravity="left|center"
    android:padding="@dimen/activity_layout_button_padding"
    android:text="@string/bild_manager"
    android:textColor="@color/almost_black"
    android:textSize="@dimen/activity_layout_button_text_size"
    android:textStyle="bold" />
```

Es wird ein einfaches `TextView` mit den Eigenschaften Größe, Abstände zu benachbarten Elementen, Textformatierung, Textinhalt und ein an der linken Seite angefügtes Bild definiert. Während der Laufzeit erstellt Android auf Basis dieser XML-Datei dann ein Objekt der Klasse `TextView` mit den oben genannten Eigenschaften und fügt es in den Layout-Kontext ein.

Realisierung von Anzeige- und Bedienelementen

AndroHmiS7 verfügt in der aktuellen Version über einen Pool verschiedener Anzeige- und Bedienelemente, welcher um weitere Elemente beliebig erweiterbar sein sollen. Um diesem Anspruch gerecht zu werden, ist jedes Element durch eine eigene Klasse repräsentiert und implementiert das Interface `S7View`. Dieses stellt die Schnittstelle zwischen Runtime bzw. Editor und den Elementen bereit. Sie zwingt die Elemente zur Implementierung bestimmter Methoden, welche von der Runtime und vom Editor verwendet und vorausgesetzt werden.

Neben dem S7View Interface, welches den allgemeinen Status eines Elements verleiht, können diese noch die Interfaces S7Connectable, S7Writeable und S7Readable implementieren, um so die entsprechende Funktionen Verbinden, Schreiben und Lesen bereitzustellen. Implementiert ein Element beispielsweise das Interface S7Readable, so muss die Methode read() realisiert werden. Die assoziierte Klasse erwartet dann eine entsprechende Verhaltensweise zum Lesen von Prozessdaten. Die konkrete Implementierung bleibt dabei aber dem Element überlassen und ermöglicht so die Berücksichtigung elementspezifischer Eigenschaften.

Beispielsweise hat die read() Methode einer S7Led die Aufgabe, dem Status einer LED einen booleschen Wert – also true für „An“ und false für „Aus“ – zuzuweisen. Im Gegensatz dazu kann ein S7TextView sogar einen String als Klartext darstellen und benötigt daher eine andere Implementierung der read() Methode.

Um die hohe Wiederverwendbarkeit von Java und Android spezifischen Klassen auszunutzen, erweitern die S7Elemente durchweg bestehende Anzeige- und Bedienelemente. So ist beispielsweise ein S7Button (Die Form S7<Elementname> ist die einheitliche Bezeichnung für die Elemente) von der Android-Klasse Button abgeleitet und implementiert die entsprechenden Interfaces zum Verbinden und Schreiben.

```
1. public class S7Button extends Button implements IS7View, IS7Connectable, I
   S7Writeable
```

Anzeigeelemente

Anzeigeelemente implementieren typischerweise die Interfaces S7View, S7Connectable sowie S7Readable zum lesenden Zugriff auf die Prozessdaten. Dabei nutzen die read() Methoden die asynchronen Lese-Methoden unter Nutzung von Multiple Requests (vgl. Kapitel 4.3.2).

Tabelle 6-5 Verfügbare AndroHmiS7 Anzeigeelemente

Anzeigeelement	Beschreibung
S7Led	Binäre Anzeige im Stil einer LED, welche bei einem booleschen true Wert ein Leuchtsignal anzeigt
S7ProgressBar	Analoge Fortschrittanzeige zur Darstellung einer Variable wie beispielsweise einen Füllstand
S7TextView	Alphanummerische Ausgabe einer Variable

Bedienelemente

Bedienelemente implementieren typischerweise die Interfaces S7View, S7Connectable sowie S7Writeable zum schreibenden Zugriff auf die Prozessdaten. Analog zu den Anzeigeelementen, schreiben auch die Elemente S7EditText und S7SeekBar die Daten asynchron unter Nutzung von Multiple Requests in die SPS. Die Elemente S7Button und S7CheckBox hingegen schreiben den repräsentierten booleschen Wert ereignisgesteuert und daher synchron beim Klicken bzw. bei Eintritt des OnClick Events (vgl. Kapitel 4.3.2).

Tabelle 6-6 Verfügbare AndroidHmiS7 Bedienelemente

Bedienelement	Beschreibung
S7Button	Binäre Taster zum Schreiben des booleschen Wert `true` bei Aktivierung, `false` bei Deaktivierung
S7CheckBox	Binärer Schalter zum Schreiben des booleschen Wert `true` beim Status „Checked“, `false` bei „Unchecked“
S7EditText	Alphanummerische Eingabe einer Variable
S7SeekBar	Analoger Schieber zur Eingabe eines analogen Wertes

Daneben können statische Elemente wie das `S7Surface` verwendet werden, welche in keiner Abhängigkeit der Prozessdaten stehen und daher kein `S7Connectable` Interface implementieren.

Kontextmenüs

Kontextmenüs spielen eine wesentliche Rolle zur dialoggeführten Interaktion mit dem Benutzer der Applikation. Entsprechend dem Android Look&Feel verursacht ein langer Klick auf ein Element das Aufrufen eines Kontextmenüs mit dafür zugeschnittenen Optionen und Funktionen. Als Beispiele sind die Kontextmenüs der Anzeige- und Bedienelemente sowie die Verbindungsparametrierung aufgeführt.

Abbildung 6-14 Kontextmenü für Elemente

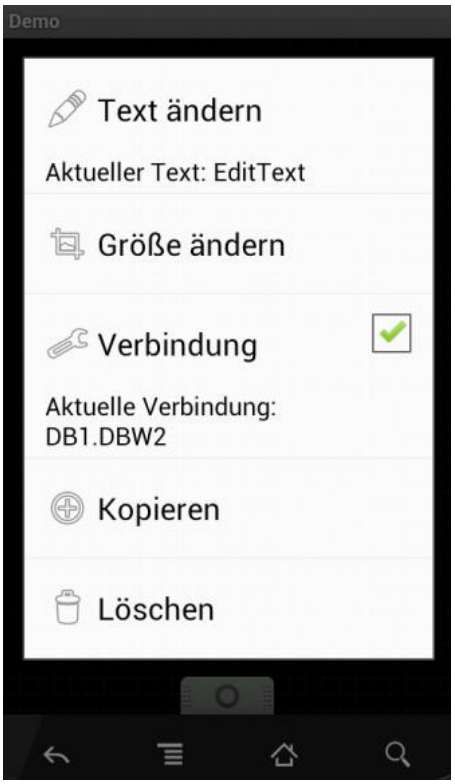


Abbildung 6-15 Kontextmenü zur Verbindungsparametrierung

6.4.2 Datenhaltung und –Speicherung

Repräsentation eines Bildes zur Laufzeit

Zur Laufzeit ist ein Bild durch eine Instanz der Klasse `Bild` zum Bearbeiten im Editor oder zur Ausführung in der Runtime repräsentiert. Neben bildspezifischen Eigenschaften wie *IP-Adresse*, *Polling Interval* und *Bildname*, werden die Anzeige- und Bedienelemente (also Instanzen des Typs `S7View`) in einer für Objekte des Typs `View` ausgelegten, generischen `ArrayList<View>` gespeichert. Dies ermöglicht die komfortable Verwaltung durch dynamisches Hinzufügen und Entfernen einzelner Elemente sowie die Suche bestimmter Objekte. Jede dort abgelegte `S7View` Instanz enthält wiederrum Werte zu den Eigenschaften Größe, Position, Inhalt sowie die über ein `S7Data` Objekt repräsentierte Verbindung zu einer bestimmten Variable auf der SPS.

Speicherung und Lesens eines Bildes mittels XML

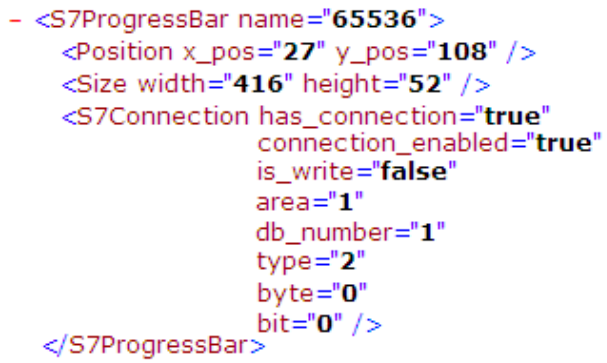
Die Speicherung eines Bildes in das XML-Format erfolgt unter Nutzung des SAX-Writers. Das Lesen erfolgt analog dazu mit dem SAX-Parser. Die XML-Dateien müssen das Bild vollständig mit allen Elementen und deren Parametrierung repräsentieren, da das Laden eines Bildes vollständig auf Basis dieser XML-Dateien geschieht. Folgende Abbildung zeigt die Struktur, in welcher die Bilder im Speicher abgelegt werden.

Abbildung 6-16 XML-Struktur eines Bildes

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <Bild>
  - <Preferences>
      <Common bild_name="Demo" />
      <Connection ip_adress="192.168.0.33" polling_intervall="0" />
    </Preferences>
  - <S7Views>
    + <S7ProgressBar name="65536">
    + <S7TextView name="Tank A">
    + <S7TextView name="Tank B">
    + <S7EditText name="EditText">
    + <S7EditText name="EditText">
    + <S7Button name="Leeren">
    + <S7CheckBox name="Start">
    + <S7Surface name="">
    + <S7Led name="Fehler">
    + <S7Led name="Tor auf">
    + <S7Led name="Anlage bereit">
    + <S7ProgressBar name="65536">
    + <S7SeekBar name="100">
    + <S7TextView name="TextView">
    + <S7SeekBar name="100">
    + <S7TextView name="TextView">
    + <S7Led name="Leeren aktiv">
    </S7Views>
</Bild>
```

Jedes Tag verfügt über einen Satz von Attributen, wie z.B. beim Erweitern der Ansicht der ersten *S7ProgressBar* ersichtlich.

Abbildung 6-17 Detailansicht eines XML-Tags mit Attributen



In Java wird die Speicherung einer XML-Datei zuerst über einen `FileOutputStream` und einen `OutputStreamWriter` vorbereitet.

```
1.  FileOutputStream fOut = mContext.openFileOutput(bild.getName()
2.          + ".xml", Context.MODE_PRIVATE);
3.  OutputStreamWriter osw = new OutputStreamWriter(fOut);
4.  osw.write(transformBildToXml());
5.  osw.close();
```

Die Methode `transformBildToXml()` liest die Bilddaten ein, speichert die gewonnenen Informationen in XML-Tags ab und gibt die zu schreibende XML-Datei als String an den `OutputSTreamWriter` zurück. Zu speichernde Informationen über das Bild sind zum einen globale Eigenschaften wie *Bildname*, *Polling-Intervall* und *IP-Adresse* und zum anderen jegliche Anzeige- und Bedienelemente mit vollständiger Parametrierung.

Tags bilden das strukturelle Grundgerüst, welche mit Attributen gefüllt werden.

```
1.  ser.startTag("", Tag.PREFERENCES);
2.  ser.attribute("", Tag.BILD_NAME, mBild.getName());
3.  ser.endTag("", Tag.PREFERENCES);
```

Das Objekt `ser` ist eine Instanz der Klasse `XmlSerializer`, welche die Aufgaben der Tag- und Attributzuweisung übernimmt.

Der lesende Zugriff auf XML-Dateien erfolgt ereignisgesteuert. Hierfür wird eine verschachtelte Struktur von Parser- und Reader-Objekten aufgebaut und mit einem Handler versehen, welcher auf Ereignisse wie z.B. das Erkennen eines Start- oder Endtags und dessen Attribute reagieren kann.

```
1.  SAXParserFactory spf = SAXParserFactory.newInstance();
2.  SAXParser sp = spf.newSAXParser();
3.  XMLReader xr = sp.getXMLReader();
4.  BildDataHandler dataHandler = new BildDataHandler(mContext);
5.  xr.setContentHandler(dataHandler);
6.  xr.parse(new InputSource(new FileInputStream(mContext.getFilesDir()
7.          + "/" + bildName + ".xml")));
8.  mBild = dataHandler.getBild();
```

Der Handler gibt eine Instanz der Klasse `Bild` mit den empfangenen Bilddaten aus der XML-Datei zurück. Diese Daten gewinnt er durch Empfangen von Events wie beispielsweise dem `startElement` Event.

```
1. public void startElement(String namespaceURI, String localName,
2.         String qName, Attributes atts) throws SAXException {
3.     if (localName.equals(Tag.CONNECTION)) {
4.         mBild.setIpAdress(atts.getValue(Tag.IP_ADRESS));
5.     }
6. }
```

6.4.3 Verbindung zur SPS

S7Connection

Für das Herstellen, Trennen und Überwachen einer Verbindung zum SIMATIC Controller dient die Klasse *S7Connection*. Sie überprüft vor der Verbindungsherstellung zur SPS, ob eine Verbindung zu einem WLAN Access Point hergestellt wurde. Innerhalb eines eigenen Threads ruft sie dann die Verbindungsmethode des Interfacemoduls auf. Der verknüpfte `Handler` empfängt über eine `Message` das von *Libnodave* zurückgegebene Ergebnis des Verbindungsaufbaus (Der Wert 0 steht hier für eine erfolgreiche Verbindung) und kann hierauf entsprechend mit `setConnected(true)` reagieren.

```
1. public void connectToPlc() {
2.     if (Util.isWifiActivated(mActivity)) {
3.         if (!mConnectionAlive) {
4.             mProgressDialog.show();
5.             Thread thread = new Thread(this);
6.             thread.start();
7.         }
8.     }
9. }
10.
11. public void run() {
12.     Message msg = new Message();
13.     res = connect(((IConnectableActivity) mActivity).getIpAddress());

14.     msg.arg1 = res;
15.     handler.sendMessage(msg);
16. }
17.
18. private Handler handler = new Handler() {
19.     public void handleMessage(Message msg) {
20.         mProgressDialog.dismiss();
21.         if (msg.arg1 == 0) {
22.             setConnected(true);
23.         }
24.     }
25. };
```

S7Data

Die S7Data Klasse enthält alle Verbindungsinformationen, welche *Libnodave* für das Lesen oder Schreiben von Variablenwerten aus bzw. in die SPS benötigt. Zudem realisiert sie abhängig vom Datentyp die `read()` und `write()` Methoden.

```
1. public Object read() {
```

```
2.          switch (mType) {
3.          case Type.BOOL:
4.              boolean value = (mBuffer.get(0) > 0) ? true : false;
5.              return new Boolean(value);
6.          case Type.BYTE:
7.              return new String("B#16#"
8.                      + Integer.toHexString(mBuffer.get(0) & 0xFF));
9.          case Type.WORD:
10.             return new String("W#16#" + Integer.toHexString(
11.                                 mBuffer.getChar(0)));
12.         case Type.INT:
13.             return new String(Integer.toString(mBuffer.getShort(0)));

14.         }
15.     }
16.     return null;
17. }
```

Abhängig vom Speicherbedarf des Datentyps werden hier entsprechend viele Bytes aus dem Buffer entnommen, gegebenenfalls konvertiert und allgemein als `Object` zurückgegeben. Da jegliche Klasse von der Superklasse `Object` abgeleitet ist, kann der Rückgabewert mit einer Instanz des Typs `Boolean` für den S7-Datentyp `BOOL` oder `String` für alle anderen Typen besetzt werden. Der Typ `Object` bietet sich an, da so in den nächsten Inkrementen des Entwicklungsprozesses weitere Anzeigeelemente mit bisher unbekannter Struktur und daher unbekannten Rückgabetyp unterstützt werden können.

```
1.  public void write(Object object) {
2.          switch (mType) {
3.          case Type.BOOL:
4.              byte value = (((Boolean) object == true) ?
5.                                  (byte) 1 : (byte) 0);
6.              mBuffer.put(0, value);
7.              break;
8.          case Type.BYTE:
9.              mBuffer.put(0,
10.                     (byte) Integer.parseInt(object.toString(),
11.                      BASE_HEXADECIMAL));
12.             break;
13.         case Type.WORD:
14.             mBuffer.putChar(0,
15.                     (char) Integer.parseInt(object.toString(),
16.                     BASE_HEXADECIMAL));
17.             break;
18.         case Type.INT:
19.             mBuffer.putShort(0,
20.                     (short) Integer.parseInt(object.toString(),
21.                     BASE_DECIMAL));
22.             break;
23.         }
24. }
```

Analog zur `read()` Methode, füllt die `write()` Methode den Buffer mit entsprechend vielen Bytes. Den Wert empfängt sie dabei über den Parameter `object` des Typs `Object`. Dies ermöglicht auch hier die Erweiterung durch neue Bedienelemente, welche die Übergabe eines beliebigen Objekttypen an die Schreibmethode erlaubt.

Polling Thread

Der *Polling Thread* übernimmt die Aufgabe, eine Anfrage an die SPS anzustoßen. Die von der Superklasse geerbte Methode `run()`schickt nach Ablauf des *Polling Intervalls* (Zeit bis zum nächsten Zyklus) eine Nachricht an den korrelierenden `Handler` zum Starten des Aktualisierungsprozesses (vgl. Sequenzdiagramm im Kapitel 6.2.3).

```
1.  public void run() {
2.           while (mRun) {
3.               if (mReady) {
4.                   handler.sendEmptyMessage(0);
5.               }
6.               sleep(mBildExecuter.getBildPollingInterval());
7.           }
8.    }
9.
10.  private Handler handler = new Handler() {
11.     public void handleMessage(Message msg) {
12.         mReady = false;
13.             mBildExecuter.refresh();
14.         mReady = true;
15.     }
16.  };
```

6.4.4 Event-Handling

Listeners

Wie im Detailentwurf im Rahmen des MVC beschrieben, erfolgt die Implementierung des *Control* über Listenerobjekte. Diese Listener werden UI-Elementen zugewiesen und dadurch mit einer Ereignisverarbeitung versehen. Zum Beispiel kann einem Button ein solches Listenerobjekt zugewiesen werden, um eine bestimmte Reaktion auf einen Klick anzubieten. Eine Übersicht aller verwendeten Listenerobjekte und deren Verwendung gibt folgende Tabelle.

Tabelle 6-7 Listener zum Event-Handling

Listener	Verwendung
BildLongClickListener	Langer Klick auf ein Bild in der Bilderverwaltung zum Öffnen eines Kontextmenüs
BildShortClickListener	Kurzer Klick auf dieses Bild zur Ausführung in der Runtime
ToolTouchListener	Touch Event auf ein Element in der Toolbox des Editors führt zum Hinzufügen des Elements auf den Bildschirm
ViewLongClickListener	Langer Klick auf ein Anzeige- oder Bedienelement zum Öffnen des Kontextmenüs für die Elementparametrierung
ViewLongClickListenerSmall	Äquivalent zum oben genannten Listener ohne Eintrag zur Verbindung
ViewShortClickListener	Kurzer Klick auf diese Elemente zum Fokussieren und Hervorheben
ViewTouchDefaultListener	Touch Event auf ein Element im Editor zum Verschieben und Anordnen auf dem Bildschirm
ViewTouchSizeLIstener	Touch Event auf ein Element im Editor im Modus „Resizing“ zum Ändern der Größe

6.5 Anschließende Phasen

Validierung, Inbetriebnahme und Wartung, Weiterentwicklung und Einstellung

Der begrenzte Zeitrahmen zur Entwicklung der Applikation und der Anfertigung der Bachelor-Thesis erlaubte die Durchführung der Phasen Anforderungsermittlung, Analyse, Design und schloss mit der Phase Implementierung ab. Dennoch wurden während der Implementierungsphase Unit-Tests und Funktionstests einzelner Module durchgeführt, die eine stabile Ausführung in der aktuellen Version ermöglichten. Weitere Schritte im Rahmen der Validierung sind Tests auf Android-Geräten aller unterstützten Versionen und unterschiedlichen SIMATIC Controllern der Reihe S7-200/300/400 und der WinAC. Bezüglich der Weiterentwicklung des Prototyps sind Tests zum Usability-Engineering, also der Benutzerfreundlichkeit der Applikation, Langzeitstabilitätstests und Test zur Fehlerbehandlung durchzuführen.

Aktueller Status

- Lauffähige Applikation mit den Funktionen Bilderverwaltung, Editor, Runtime, Diagnose-Panel und Web-Server Zugriff
- Performantes Schreiben und Lesen der Datentypen INT, WORD, BYTE, BIT in den Speicherbereichen Merke und Datenbaustein
- Anzeigeelemente: Textfeld, Eingabefeld, LED, Fortschritts- bzw. Füllanzeige
- Bedienelemente: Button, Checkbox und analoger Schieber
- Einlesen des Diagnosepuffers der SPS sowie Fernsteuerung der SPS

Offene Punkte

- Sicherheit der Verbindung über WLAN
- Unverschlüsselte Übertragung der Daten (abgesehen von der Verschlüsselung des WLAN-Netzwerks)
- Authentifizierungsmöglichkeiten
- Trennung von Runtime und Engineering zwecks Einschränkung im Zugriff auf die Prozessdaten
- Unterstützung weiterer Datentypen wie REAL, TIME, TIME_OF_DAY, S5TIME

7 Fazit

Erreichung der Ziele

Das angestrebte Ziel, eine Android-Applikation zur Interaktion mit Prozessdaten auf einer Siemens SIMATIC SPS zu entwickeln, wurde erreicht.

Erfahrungen und Bewertung der Entwicklung der Applikation

Die Entwicklung einer komplexen Software erfordert von Beginn an eine strukturierte Vorgehensweise. Mithilfe der Einteilung des Entwicklungsprozesses in einzelne Phasen – beginnend beim Sammeln der Anforderungen bis zur Implementierung – war es mir möglich, den roten Faden zu verfolgen und die Phasen strukturiert abzuarbeiten. Am Ende jeder Phase stand dabei ein Ergebnis, auf welches die nächste Phase aufbaute. So konnte ein ständiger Abgleich mit dem Zeitplan und eine Abschätzung der Realisierbarkeit erfolgen.

Ich gewann Einblicke in die Herausforderungen der mobilen Applikationsentwicklung auf der Android-Plattform. Anders als bei der Softwareentwicklung für klassische Desktopanwendungen, spielen bei mobilen Plattformen die Punkte Speicherressourcen-, Leistungs- und Energiemanagement eine übergeordnete Rolle. Unterstützend durch die zugeschnittenen Android-Systemarchitektur konnten wesentliche Anforderungen an solche Applikationen implementieren werden.

Das Zusammenführen eines geschlossenen, auf Sicherheit bedachten Automatisierungssystems mit dem offenen, auf Benutzerfreundlichkeit aufbauenden mobilen Android-System stieß bei der Bearbeitung der Thesis häufig auf berechtigte Einwände. Gerade die Bedenken in Richtung Sicherheit der WLAN-Schnittstelle und des Android-Frameworks sowie die Nutzung der Open-Source-Bibliothek *Libnodave* beschränken die Applikation vorerst auf einen experimentellen Status.

Die Entwicklung von Software im Team, der Zeitdruck aufgrund der parallelen Durchführung von Recherche, Entwicklung und Verfassen dieser Thesis sowie die Anwendung neu erlernter Technologien waren sehr herausfordernde aber motivierende Punkte. Durch die bereits zuvor gesammelten Erfahrungen im Bereich der Softwareentwicklung und die Unterstützung meiner erfahrenen Betreuer, konnte ein zufriedenstellendes Ergebnis erreicht werden.

Ausblick

Als experimentelle Anwendung zum Bedienen kleinerer Automatisierungssysteme, Messemodelle, Haussteuerungen oder als Hilfe bei der Inbetriebnahme ist die Applikation verwendbar. Für kritische Prozesse fehlt die Durchgängigkeit sicherheitsrelevanter Aspekte, insbesondere bei der Kommunikation über die WLAN-Schnittstelle. Die Entwicklung des IWLAN (Industrial WLAN) könnte diese Lücke schließen und einer mobilen HMI-Applikation ein breiteres Anwendungsfeld geben. Die Unterstützung weiterer Datentypen und SPS-Typen steht bei der weiteren Entwicklung an erster Stelle.

8 Stichwortverzeichnis

Verwendete Stich- und Fachwörter

Tabelle 8-1 Stich- und Fachwörter

Stichwort	Bedeutung / Erläuterung
ADT (Android Developer Tools)	Sammlung von Tools zum Entwickeln von Android-Applikationen
API (Application Programming Interface)	Programmierschnittstelle zwischen zwei Softwaresystemen
APK (Android Application Package)	Gepackte Archiv zur Installation von Android-Applikationen
Bild	Oberfläche mit Anzeige- und Bedienelementen
Compiler	Wandelt Quellcode in Maschinencode um
Drag&Drop	Bedienmethode zum Platzieren grafischer Elemente
DVM (Dalvik Virtual Machine)	Laufzeitumgebung für Android-Applikationen
HMI (Human-Machine-Interface)	Mensch-Maschine-Schnittstelle zwischen Anwender und Software - zumeist über Bildschirme
IDE (Integrated Development Environment)	Entwicklungsumgebung für das Entwickeln von Software
IP-Address (Internet Protocol)	Adresse eines Teilnehmers über ein Ethernet-Netz
JDK (Java Development Kit)	Sammlung von Bibliotheken und Tools zum Erstellen von Java-Anwendungen
JRE (Java Runtime Environment)	Laufzeitumgebung für Java-Anwendungen
LogCat	Ausgabefenster für Android-Meldungen
Look&Feel	Standarisierte Design-Aspekte einer Software
MVC (Model-View-Control)	Modell zur Trennung von UI (View), Information (Model) und Funktion (Control)
PLC (Programmable Logic Control)	Englische Bezeichnung einer SPS
Polling	Zyklische Abfrage von Daten
Runtime	Laufzeit oder ausführender Programmteil
S7	Allgemeine Bezeichnung des SIMATIC System
S7Views	Anzeige- und Bedienelemente in *AndroHmiS7*
SAX (Simple API for XML)	Standard-Schnittstelle zum Zugriff auf XML-Dateien
SDK (Software Development Kit)	Sammlung von Bibliotheken und Tools zum Erstellen eigener Software
SPS (Speicherprogrammierbare Steuerung)	Gerät zur Steuerung einer Maschine oder Anlage
Step7	Projektierungssoftware für SIMATIC Komponenten
UI (User Interface)	Grafische Benutzerschnittstelle
UML (Unified Modeling Language)	Modellierungssprache zur Spezifikation, Konstruktion und Dokumentation von Software
WLAN (Wireless Local Area Network)	Lokales Funknetzwerk
XML (Extensible Markup Language)	Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten

9 Literaturverzeichnis

Verwendete Literatur und Quellen

[1] Berger Hans Automatisieren mit SIMATIC [Buch]. - Erlangen : Publics MCD, 2000.

[2] Berger Hans Automatisieren mit STEP7 in AWL und SCL [Buch]. - Erlangen : Publics Corporate Publishing, 2002.

[3] Ehrenstein Constantin Apps entwickeln mit Android 4. - 2012.

[4] Gevatter H.J. Grünhaupt U. Handbuch der Mess und Automatisierungstechnik in der Produktion, [Buch]. - Berlin, Heidelberg : [s.n.], 2006.

[5] GNU GNU Library General Public License, Version 2.0 [Online]. - 1991. - 31. Juli 2012. - http://www.gnu.org/licenses/old-licenses/lgpl-2.0.html#SEC4.

[6] Google Android Developers - Ressourcen [Online]. - 2012. - 20. Juli 2012. - http://developer.android.com/guide/topics/resources/providing-resources.html.

[7] Google Development Guide [Online]. - 2012. - 24. April 2012. - http://developer.android.com.

[8] Hashimi Sayed Y. Pro Android 2 [Buch]. - [s.l.] : APress, 2010. - 978-1430226598.

[9] heise mobile Die Architektur von Android [Online]. - 2012. - 30. 07 2012. - http://www.heise.de/mobil/artikel/Innenansichten-eines-Smartphone-Betriebssystems-1203743.html.

[10] Hergenhahn Thomas Libnodave 0.8.6 Dokumentation. - London : [s.n.], 30. Juni 2011.

[11] Irmer Thomas Integration eines G-Code Interpreters in die Soft-SPS WinAC RTX [PDF]. - Köln : [s.n.], 2012.

[12] Langmann Reinhard Prozeßlenkung: Grundlagen zur Automatisierung technischer Prozesse [Buch]. - Leipzig : Vieweg+Teubner Verlag, 1996.

[13] Langmann Reinhard Taschenbuch der Automatisierung [Buch]. - München : Carl Hanser Verlag GmbH, 2010.

[14] Liang Sheng The JavaTM Native [Buch]. - USA : Sun Microsystems, 2002.

[15] Lux Wolfgang Vorlesungsskript Software Engineering I WS07/08. - Düsseldorf : [s.n.], 7. November 2008.

[16] Lux Wolfgang Vorlesungsskript Software Engineering II SS11. - Düsseldorf : [s.n.], 15. März 2011.

[17] Michael Richter Markus Flückiger Usability Engineering kompakt [Buch]. - München : Elsevier, 2007.

[18] Moecker Sascha Plattformunabhängige Applikationsentwicklung für mobile Geräte unter Nutzung von Web Technologien [Buch]. - Köln : [s.n.], 2012.

[19] Murphy Mark L. Beginning Android 2 : [begin the journey toward your own successful Android 2 applications] [Buch]. - [s.l.] : APress, 2010. - 978-1430226291 .

[20] Siemens AG SIMATIC NET - Industrielle Kommunikation mit PG/PC [PDF]. - Erlangen : [s.n.], 2008.

[21] Zroud Issam Entwurf einer Scope-Applikation zur Visualisierung von [PDF]. - Köln : [s.n.], 2012.

10 Abbildungsverzeichnis

Verwendete Abbildungen

11 Tabellenverzeichnis

Verwendete Tabellen

12 Anlagen

Angefügte Dokumente

- Zeitlichen Projektplanung des Entwicklungsprozesses der Applikation *AndroHmiS7* als Gant-Diagramm
- Formular zur Anmeldung dieser Bachelor-Thesis

Dokumente und Medien auf beiliegender Projekt-CD

- Diese Bachelor-Thesis als PDF-Ausdruck
- Applikation *AndroHmiS7*
  - Quellcode als gepacktes Android-Eclipse-Projekt
  - Quellcode als PDF-Ausdruck
  - Javadoc HTML Dokumentation
  - Screenshots und Videos
- Interfacemodul
  - Quellcode als gepacktes Archiv
  - Quellcode als PDF-Ausdruck
- Bibliothek *Libnodave*
  - Quellcode als gepacktes Archiv
  - Fehlercodes
  - HTML Dokumentation