Realisierung einer Touch-Bedienung für die Gebäudeautomation auf Basis der Android-Plattform.

BACHELORARBEIT

für die Prüfung zum

Bachelor of Engineering

im Studiengang Informationstechnik

an der Dualen Hochschule Baden-Württemberg Lörrach

von

DAVID EBERLEIN

September 2009

Bearbeitungszeitraum:	Juni 2009 bis September 2009
Kurs:	TIT06A
Ausbildungsfirma:	Fr. Sauter AG Basel
Gutachter der Ausbildungsfirma:	Dipl.-Ing. Stefan Hoene
Gutachter der Studienakademie:	Prof. Dr. Stephan Trahasch

Danksagung

Bei Herrn Prof. Dr. S. Trahasch bedanke ich mich für die Betreuung der Bachelorarbeit und die Unterstützung bei der Erstellung dieser Arbeit. Desweiteren gilt mein besonderer Dank Herrn S. Hoene, der mich durch seine engagierte Betreuung und stete Diskussionsbereitschaft mit vielseitigen Denkanstößen für die entwickelte Software bereicherte. Auch Herrn T. Krieg möchte ich Danken, der mir das Bearbeiten dieser Arbeit überhaupt ermöglichte.

Des weiteren danke ich den Leitern der Sauter Mechanik Lehrlingswerkstatt, Herrn G. Martin und Herrn P. Nebel, die es ermöglichten, dass entworfene 3D-Modell Physisch abzubilden. Ebenso danke ich den Herren E. Michael, F. Brugger, L. Guthnecht, F. Brand sowie R. Santchi, welche am Bau des Modells beteiligt waren.

Mein spezieller Dank geht ebenso an Herrn S. Krauß, der die Steuerplatine für das Modell Designte, Lötete und Programmierte sowie Herrn Dr. M. Bianchi, der mir bei der Erstellung des Regelungsplans stets zur Seite stand.

Auch bei den Herren F. Jost und M. Ziswiler von Noser Engineerig möchte ich mich für die tatkräftige Unterstützung bei der Portierung der Android-Plattform sowie der steten Bereitschaft meine Fragen zu beantworten sehr herzlich bedanken.

Nicht zuletzt will ich mich bei meiner Frau Anna, die das Modell Tapezierte, bei der Entstehung dieser Arbeit half und mich stets unterstützte sehr herzlich bedanken, Ich liebe dich.

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit den Möglichkeiten, die Software-Plattform Android als Grundlage einer intuitiven, touch-basierten Software für die Visualisierung und Bedienung von Gebäudeautomation einzusetzen.

Hierzu wird ein System, bestehend aus drei Komponenten aufgebaut. Die erste besteht aus dem Modell eines Auditoriums, welches mehrere regelbare Datenpunkte enthält. Die zweite Komponente ist eine Sauter Modu525 Automationsstation, die das Modell regelt. Eine Android-Applikation bildet den letzten Teil des Systems und stellt das Visualisierungs- und Bedien-Interface für den Benutzer dar. Die Applikation läuft sowohl auf einem HTC Dream (G1) sowie einem Toradex Evaluierungsboard mit 10,4“ Touchscreen, auf das im Zuge der Arbeit Android portiert wurde. Die Android-Applikation kommuniziert über die Protokolle BACnet bzw. Webservice wahlweise über Ethernet, WLAN oder Mobilfunknetze mit der Automationsstation.

Die Tatsache, dass es sich bei Android um ein neuentwickeltes, für Embedded-Touch-Devices spezialisiertes Framework handelt, hat positive wie auch negative Seiten.

So wird trotz der Verwendung von Java als Programmiersprache für jeden Entwickler eine grundlegende Einführung in die Android-Umgebung benötigt. Zudem sind momentan nur Portierungen auf ARM-Architekturen in Betracht zu ziehen und auch diese erfordern einiges an Aufwand.

Die Ergebnisse der Arbeit zeigen jedoch, dass Android technologisch gesehen durchaus in der Gebäudeautomation einsetzbar ist. Die Reaktionszeiten zwischen Schalten eines Datenpunktes in der Software und der Reaktion des Modells liegen in einem guten Bereich. Auch das Touch Interface ist intuitiv steuerbar und ansprechend, was Usability Tests belegen.

.

Summary

The following paper discusses the possibilities of using the software platform Android as a basis for an intuitive touch based software aiming for the visualization and control of building automation.

For this purpose a system made up of three components is set up. The first one consists of an auditorium model that contains several controllable data points. The second component is a Sauter Modu525 automation station that regulates the model. An Android application forms the last part of the system and constitutes the visualization and operation interface for the user. The application runs on a HTC Dream (G1) as well as on a Toradex evaluation board with a 10.4” touch screen. Android has been ported onto the evaluation board in the course of this paper. The Android application communicates via the protocols BACnet or Webservice optionally over Ethernet, WLAN or mobile networks with the automation station.

The fact that Android is a newly developed framework that is specialized for embedded touch devices has positive as well as negative aspects.

A basic introduction into the Android environment is needed for every developer despite the use of Java as a programming language. Also, only ports onto the ARM architectures can be taken into consideration at the moment and these require a high expenditure.

However, the results of the paper show that technologically Android is definitely applicable in building automation. The reaction times between the changing of a data point in the software and the reaction of the model is satisfactory. Furthermore the touch interface is intuitively assessable which has been verified by usability tests.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ADB	Android Debugger Bridge
AS	Automationsstation
BACnet	Building Automation Control Network
COV	Change of Value
DDMS	Dalvik Debug Monitor Service
GUI	Graphical User Interface
IDE	Integrated Development Environment
JNI	Java Native Interface
OHA	Open Handset Alliance
REST	Representational State Transfer
PWM	Pulsweitenmodulation
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VM	Virtual Machine
WSDL	Web Service Description Language
XML	Extended Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Einführung in die Problematik

In der Gebäudeautomation werden seit Jahren sogenannte Automationsstationen eingesetzt um Heizung, Lüftung, Klima und Licht zu steuern. Diese Stationen sind der zentrale Anlaufpunkt von Sensoren, Antrieben für Klimatechnik, Lichtern usw. Ihrem Regelungsplan entsprechend regeln die Stationen daraufhin Temperatur, Luftfeuchte, Beleuchtung usw.

Die neueste Generation der Sauter Automationsstationen (AS) Modu525 verfügen bereits über ein Webinterface (ModuWeb), in dem der Benutzer aktuelle Zustände und Trendverläufe aller Datenpunkte, sowie aktuelle Alarme von allerorts über das Internet einsehen und quittieren kann. Es können Zeitpläne erstellt, Netzwerkeinstellungen verändert und Benutzer mit verschiedenen Rechten angelegt werden. Insgesamt verfügt das Webinterface also über ein großes Spektrum an Funktionen, die für den Erfahrenen Nutzer, wie beispielsweise den Hausmeister, sehr wertvoll und notwendig sind.

Oft gibt es jedoch Situationen, in denen unerfahrene Benutzer mit dem geregelten System interagieren müssen. Sei dies das Verstellen der Temperatur im Hotelzimmer, das Öffnen und Schließen von Storen oder das Schalten von Lichtern. Traditionell wird dies mit mechanischen Schaltern gelöst, die mit der Automationsstation verbunden sind und die gewünschte Funktion auslösen. Mit dem Fortschritt der Technik wird es jedoch möglich, die einzelnen, überall in einem Gebäude verteilten Schalter durch neue digitale Technologien zu bündeln und das Schalten aller relevanten Datenpunkte zentral auf einen Blick zu erlauben. Mechanische Schalter würden mit einem solchen System koexistieren und beispielweise bei Ausfällen als Ersatz dienen können.

Ziel der vorliegenden Arbeit ist es, ein System zu entwickeln, auf dem alle wichtigen Datenpunkte übersichtlich und auch für den unerfahrenen Benutzer schnell zugänglich und steuerbar gemacht werden. Hierfür eignen sich insbesondere Embedded Wireless Touch Devices. Hierbei handelt es sich um kleine, handgroße Geräte, die über ein Touch-Display gesteuert und über Wireless-Verbindungen Zugriff auf lokale Netze sowie das Internet haben. Solche Geräte

ermöglichen es, gewünschte Datenpunkte von überall in einem Gebäude schnell und intuitiv über die Touch-Oberfläche zu steuern.

Neben diesen portablen Geräten sind ebenfalls stationäre Geräte mit großen Touch-Displays interessant, die an zentralen Stellen, wie beispielsweise neben der Tür im Arbeits- oder Hotelzimmer, platziert sind.

Das Implementieren einer Software, die genau diesen Zweck erfüllt und sowohl auf portablen Geräten mit kleinen Touch-Displays als auch auf stationären Geräten mit größeren Touch-Displays lauffähig ist, soll im Rahmen dieser Arbeit durchgeführt werden.

1.2 Android

Mit Android bietet Google und die Open Handset Alliace (OHA) [1] eine Software-Plattform an, die für Handheld Devices entwickelt wurde. Vom Betriebssystem, über Media-Codecs bis hin zum Entwicklerframework mit einer Entwicklungsumgebung und Emulator bietet Android alles, was ein Handheld-Device softwaretechnisch benötigt. Die gesamte Software-Plattform befindet sich unter der Apache 2 Lizenz [2] und ist somit jedermann kostenfrei zugänglich. Dank des Linux Kernels, der den Kern des Betriebssystems darstellt, soll das Portieren von Android auf verschiedene Hardware relativ leicht möglich sein. In diesem Jahr wurden bereits fünf Handhelds verschiedener Hersteller mit dem Android Betriebssystem vorgestellt und es sollen noch weitere folgen.

Somit bietet Android alle Grundlagen, um eine touchorientierte Visualisierungs- und Bedienungssoftware auf einer Embedded Plattform zu entwickeln. Inwiefern sich Android für den Einsatz in der Gebäudeautomation eignet, soll diese Arbeit zeigen.

1.3 Gegenstand und Ziele der Arbeit

Diese Arbeit soll, anhand einer auf Android entwickelten Beispielsoftware, die Möglichkeiten untersuchen Touch-Applikationen auf der Android-Plattform zu entwickeln und diese in die Gebäudeautomation zu integrieren. Hierfür soll ein System aufgebaut werden, das aus einem mechanischen, automatisierten Modell eines Auditoriums (mehr dazu in Kapitel 4), einer AS Modu525, die für die Regelung verantwortlich ist, sowie der Android-Applikation, welche als Vi-

sualisierungs- und Bedienschnittstelle fungiert, bestehen soll. Im Zuge dessen sollen die Möglichkeiten untersucht werden, die Android-Plattform auf einer Hardware mit großem Touchscreen zu portieren und die entwickelte Software auch dort auszuführen.

Ziel ist es festzustellen, ob Android als Software-Plattform für Visualisierungsapplikationen in der Gebäudeautomation geeignet ist. Hierbei soll der Softwareentwicklungsprozess, Test-, Profiling- und Debugging-Möglichkeiten sowie die Portierbarkeit von Android auf andere Hardware als Kriterien dienen.

Außer Android sollen keine weiteren Technologien näher untersucht werden. Bei der Software soll es sich um einen Prototypen und kein fertiges Produkt handeln. Der Schwerpunkt soll auf der Usability, der Navigation, dem Look and Feel sowie der Netzwerkkommunikation und den Reaktionen von Software und Modell liegen.

Sicherheitsaspekte sollen untersucht und dokumentiert, müssen aber nicht implementiert werden. Auch von einem Rollen- und Rechtekonzept kann in diesem Prototyp abgesehen werden.

2 Grundlagen

2.1 Alternative Embedded Touch Lösungen

2.1.1 Web Applikationen

Um eine Embedded-Touch-Software für die Visualisierung und Bedienung der Gebäudeautomation zu Realisieren, gibt es verschiedene Lösungsansätze. Zum Einen muss die Möglichkeit von Webapplikationen in Betracht gezogen werden, die durch die steigenden Internet-Bandraten im Desktopbereich immer mehr an Popularität gewinnen. Hierbei befände sich die eigentliche Applikation auf der AS und würde von dem entsprechenden Embedded Device heruntergeladen und lokal ausgeführt werden. Frameworks hierfür sind beispielweise Flex, Silverlight oder JavaFX. [3]

Trotz des Vormarschs von Webapplikationen im Desktopbereich, sind diese auf Embedded-Wireless-Systemen aus verschiedenen Gründen noch kaum verbreitet. Grund hierfür ist vor allem die Tatsache, dass die Geräte über Mobilfunkverbindungen, zumeist GPRS oder UMTS, auf das Internet zugreifen. Diese sind zum Einen in der Bandbreite eingeschränkt und zum Anderen werden die Preise für die Nutzung dieser Dienste oft nach dem übertragenen Datenvolumen abgerechnet. So ist eine Applikation, die - je nach Caching-Strategie - vor jeder Benutzung erst heruntergeladen werden muss, eher unpraktisch für den Endanwender. Zusätzlich sind die Geräte meist über einen Akku mit begrenzter Kapazität gespeist, der sich bei intensivem Datenverkehr schnell leert. In diesem Bereich werden stattdessen Applikationen gewünscht, die einen minimalen Datentransfer benötigen.

Aus diesen Gründen gibt es auch kaum Clients für entsprechende Hardware und Plug-ins für Embedded Browser. Auch wenn dies mit der neuesten Generation von Mobile Devices und Mobilfunknetzen mehr und mehr zunimmt, wird es wohl noch 1-2 Jahre dauern, bis Webapplikationen ihren Durchbruch auf mobile Plattformen geschafft haben.

Für die Zukunft sind Webapplikationen eine vielversprechende Lösung, die von Sauter in anderen Projekten genauer untersucht wird. Diese Arbeit soll sich jedoch ausschließlich mit nativen Applikationen beschäftigen.

2.1.2 Mobile Plattformen

Gesucht wird also eine Software-Plattform, die von der Hardwareabstraktion über Betriebssystem und Applikationsframework mit Graphical User Interface (GUI) alle Grundlangen bietet, eine ansprechende Touch-Applikation auf Embedded Hardware zu entwickeln.

Wenn es darum geht, auf Embedded-Touch-Systemen eine Software mit ansprechenden GUIs zu erstellen, sind Mobile-Plattformen wohl die beste Wahl. Es handelt sich hierbei um Software-Plattformen, die speziell für Mobile-Devices entwickelt wurden. Interessant wäre das Entwickeln auf einer Mobile-Plattform auch deswegen, weil es - je nach Kundenwunsch - sogar möglich wäre, ganz auf die Hardware zu verzichten und dem Benutzer lediglich die Software zur Verfügung zu stellen, welche dieser sich dann auf seinen Handheld installieren kann. Der Markt der Mobile-Plattformen ist im Gegensatz zum Desktop OS Markt viel verteilter. Die führenden Plattformen speziell im Touch-Bereich sind iPhone OS, Symbian OS und Windows Mobile[3]. Etwas abgelegen liegt der Neueinsteiger Android[4]. Obwohl all diese Plattformen, insbesondere das iPhone OS, das Implementieren von ansprechenden Touch-Applikationen ermöglichen, sind die Lizenzbestimmungen und das Portieren auf andere Hardware sehr problematisch bzw. rechtlich gar nicht möglich. Das linuxbasierte Android ist das einzige System, das vollständig lizenzfrei ist und theoretisch ohne Probleme auf eine beliebige linuxfähige ARM Architektur portiert werden kann. Wenn man dem Herausgeber Google Glauben schenkt, soll die Plattform den Mobile-Markt revolutionieren und in den nächsten Jahren der Markt mit Android-Handys überschwemmt werden.

Da die Fr. Sauter AG auf ihren Automationsstationen ebenfalls ein Linux basierte Softwarearchitektur mit einer darauf befindlichen Java VM einsetzt, ist Android auch für andere Einsatzgebiete besonders interessant und soll im Laufe der Arbeit genauer untersucht werden.

2.2 Einführung in die Android-Plattform

2.2.1 Überblick

Android ist eine Open Source Software-Plattform für Mobile-Multimedia-Devices. Das Projekt ist von Google initiiert und wird auch unter dessen Namen vermarktet. Entwickelt wird es von der Open Handset Alliance, die sich aus vielen kleinen und größeren Firmen zusammensetzt. Hierzu gehören Mobilfunkbetreiber wie T-Online, Handyhersteller wie Sony-Erricson, Samsung oder HTC sowie Firmen, die sich auf Softwareentwicklung für Embedde Devices spezialisiert haben. Google Android ist ein reines Softwarepaket, das lizenzkostenfrei von jedem beliebigen Hersteller auf seiner Hardware eingesetzt werden kann.

Googles Ziel dahinter ist es, dem Mobilen Markt ein einheitliches, offenes Betriebssystem zu geben, welches auf der Hardware der verschiedensten Handy-Hersteller läuft. Dies soll das Entwickeln von Mobile-Applikationen erleichtern, da mit einer einheitlichen Plattform keine Softwareportierungen mehr nötig sind und ein Softwareprodukt ohne Anpassungen auf jedem Android Handy ausführbar ist.

Bis heute sind unter anderem die PDAs HTC Dream (G1), HTC Magic, HTC Hero und Samsung Galaxy mit der Android-Plattform ausgeliefert worden. Aber auch andere Hersteller wie Archos, Sony Ericsson und Motorola wollen in den kommenden Monaten und Jahren entsprechende Hardware veröffentlichen. Es gibt einige Projekte, in denen die Android-Plattform auf andere Hardware, wie beispielsweise das Nokia N810 [5] oder den Asus EEEPC [6], mehr oder minder funktionsfähig portiert wurde. Offiziell unterstützt werden bis jetzt jedoch nur ARM Plattformen.

Die linuxbasierte Android-Software-Plattform setzt sich, wie auf Abbildung 1 zu sehen ist, aus 4 Schichten zusammen, die in den folgenden Kapiteln vorgestellt werden sollen. Diese Kapitel haben keinen Anspruch auf Vollständigkeit und sollen nur eine grobe Einführung in das Thema geben. Tiefergehende Informationen können in den Büchern [7], [8] und [9] sowie der Android Online Dokumentation [10] gefunden werden.

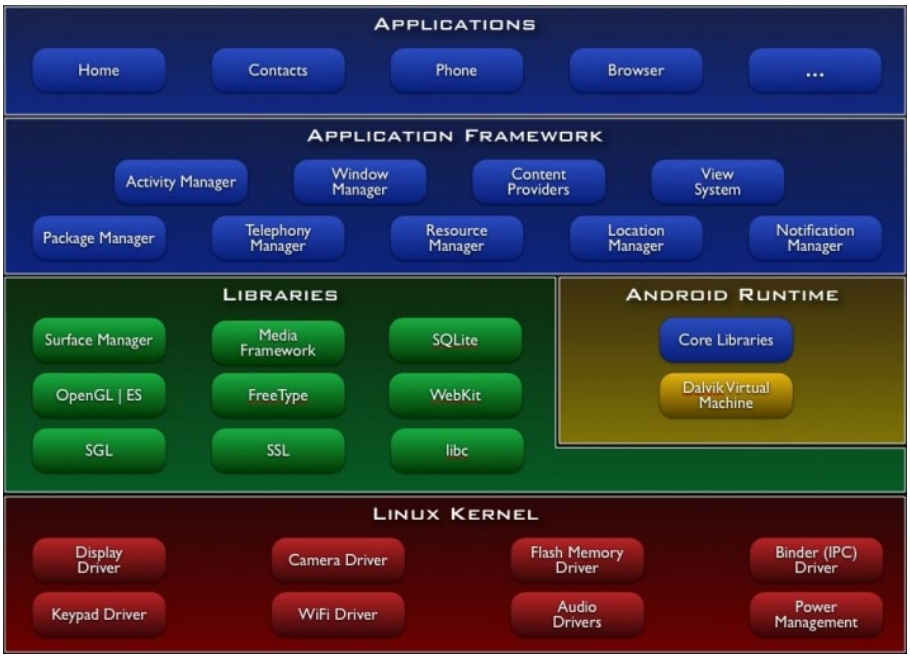


Abbildung 1: Android Framework [10]

2.2.2 Linux Kernel

Auf der untersten Ebene der Plattform arbeitet ein Linux Kernel. Dieser ist für das Threading, die Speicherverwaltung, die Hardwareabstraktion und den Netzwerkstack zuständig. Verwendet wird ein Linux 2.6 Kernel, der für das Android Framework angepasst und erweitert wurde. Die Änderungen beinhalten unter anderem Funktionalitäten, um auf Android Devices live debuggen zu können, die Möglichkeit Android-Applikationen bei niedrigem Speicher zu terminieren sowie ein angepasstes Memory-Management, das besonders für batteriebetriebene Geräte optimiert wurde.

2.2.3 Libraries

Oberhalb des Kernels befinden sich die Low-Level C / C++ Bibliotheken. Teil dieser Bibliotheken sind unter anderem eine Android eigene LibC, Grafik- und Audiobibliotheken sowie die Kryptobibliothek OpenSSL. Diese Bibliotheken sollten im Normalfall nicht von den Android-Applikationen direkt aufgerufen werden, da das Applikation Framework alle nötigen Schnittstellen nach oben bereitstellt.

Wichtig zu wissen ist, dass Google anstatt der Standard GNU LibC, die selbst entwickelte, so genannte Bionic LibC einsetzt. Diese wurde aus verschiedenen Gründen gegenüber der GNU glibc verändert:

- Lizenz: Google wollte keine GPL Lizenz einsetzten. Der Bionic Code ist daher unter der BSD Lizenz.
- Größe: Da die Bibliothek in jeden Prozess geladen wird, muss sie sehr klein sein. Bionic ist etwa 200KB groß, das ist halb so groß wie die GNU Version der glibc.
- Geschwindigkeit: Da Android auf Embedded Systemen eingesetzt wird, muss auch die LibC entsprechend schnell sein.
- Zusätzlich wurden noch einige Android spezifische Dienste z.B. für System Eigenschaften und Logging in die Bionic libc eingebaut.
- Die Bionic LibC ist somit nicht ganz kompatibel mit der GNU glibc. Alle Android Programme müssen daher gegen die Bionic libc kompiliert werden.

2.2.4 Android Runtime

Die Android Runtime besteht aus der Dalvik Virtual Machine (VM), auf der alle Android-Applikationen laufen. Obwohl Java als Programmiersprache für Android-Applikationen verwendet wird, handelt es sich bei dieser VM nicht um eine Java VM, sondern einen eigenen Interpreter. Die Java Programme werden von einem normalen Java Compiler übersetzt und anschließend mit einem Programm in das dex-Format (Dalvik executable) konvertiert, das von der Dalvik VM interpretiert wird. Dieser Bytecode ist kompakter als der, vom Java Compiler erzeugte, *.class-Bytecode und wurde speziell für die ARM Architektur auf Performanz und Speicherverbrauch optimiert.

Auch wenn es sich hier um eine spezielle VM handelt, sind fast alle Java-Framework Bibliotheken eins zu eins umgesetzt. So kann auch die Standard Java Native Interface (JNI) Schnittstelle genutzt werden, um auf die C und C++ Bibliotheken zuzugreifen.

Jeder Prozess im Android-System besitzt eine eigene VM. Dadurch sollen Prozesse voneinander getrennt und somit Sicherheit und Unabhängigkeit der Prozesse gewährleistet werden (jede VM hat einen eigenen Adress- und Speicherbereich). Für den Informationsaustausch zwischen Applikationen wird ein spezielles Datenbereitstellungssystem, der sogenannte ContentProvider, angeboten.

2.2.5 Application Framework

Für den Android-Software-Entwickler ist das Applikation Framework die Schnittstelle zum Android-System. Es handelt sich hierbei um eine umfangreiche API, die der High-Level-Programmierung zur Verfügung steht.

Die API setzt sich aus großen Teilen der Standard Sun Java API, einigen Paketen von dritten wie z.B. Apache, sowie einem großen Teil von Android eigenen Funktionen zusammen.

Hierzu gehört das GUI-Framework, welches eigens für Android entwickelt wurde und wenig mit seinen Geschwistern Swing oder SWT gemein hat. Die GUI Objekte sind für den Gebrauch auf Mobile-Devices, d.h. Touchbedienung auf kleinen Displays, optimiert. Geschrieben werden die GUIs wahlweise mit Java oder aber mit XML, ähnlich wie man es von HTML oder Flex kennt. Ziel der Einführung von XML als GUI-Beschreibungssprache ist es, das GUI von der Applikationslogik zu trennen und so eine leichte und übersichtliche GUI Erstellung und Anpassung zu erlauben.

Nähere Informationen über die Unterschiede des Android-Applikations-Frameworks zum Sun Java Framework können in Kapitel 2.3 gefunden werden.

2.2.6 Applications

Auf dieser Schicht sind Front-End Applikationen, sogenannte „Android Apps“ zu finden. Sie werden aus dem Programmmenü aufgerufen und besitzen im Normalfall ein Userinterface, mit dem der Nutzer interagieren kann.

Hierunter fallen Programme wie der Webbrowser, das Telefonbuch, die Telefonfunktion, der Media Player, Google Maps und andere. Jede Android-Applikation hat die gleichen Berechtigungen, es gibt keine Hierarchie unter diesen Applikationen. Alle von Google mitgelieferten Apps können durch andere ersetzt oder einfach gelöscht werden. Auf dieser Ebene gilt es, eine Applikation für die Gebäudesteuerung umzusetzen.

2.3 Das Android Application Framework

2.3.1 Überblick

Nachdem ein Überblick über alle Komponenten des Android-Systems gegeben wurde, soll der Aufbau des Android Frameworks etwas vertieft werden. Als Grundlage für die zu entwickelnde Software spielt das Application Framework und die richtige Anwendung, der von ihm bereitgestellten Mitteln, eine zentrale Rolle. Da sich vieles von dem Programmieren eines normalen Java-Programms unterscheidet, sollen die wesentlichen Unterschiede hier vorgestellt werden.

Das Android-Framework weicht sowohl, was GUI-Design und Navigation, als auch Applikationslifecyles angeht sehr stark von dem altbekannten Java-Sun Desktopschema ab. Das Framework erinnert an eine lokale Umsetzung von Webapplikationen und Webnavigation. Um dies zu erläutern, sollte zuerst ein Blick auf die von Android bereitgestellten Java Pakete geworfen werden.

Wie bereits erwähnt, wurde ein großer Teil der Java Sun API nachimplementiert. Hierzu zählen Pakete wie java.lang, java.io, java.net, java.math usw. Hinzukommen die org.apache sowie die org.xml Pakete, die für die http-Kommunikation und xml-Verarbeitung zur Verfügung stehen. Die Mehrheit aller Java-Bibliotheken sollten damit ohne Probleme auch auf Android laufen.

Das Lifecyclemanagement der Applikationen, die Datenverwaltung, die Interprozess-Kommunikation sowie die GUI-Darstellung werden von dem eigens dafür entworfenen Android Paketen übernommen. Dies hat sehr große Auswirkung auf die Art und Weise, wie Applikationen für Android implementiert werden.

Warum Android an die Umsetzung eines lokalen Internets erinnert, sollen die nächsten Kapitel erläutern. Die Kapitel haben aufgrund der umfangreichen Komplexität keinen Anspruch darauf, das Android Framework vollständig zu behandeln. Es werden lediglich zentrale Punkte sowie für die umzusetzende Applikation wichtige Themen herausgegriffen. Tiefergehende Informationen können in den Büchern [7], [8] und [9] gefunden werden.

2.3.2 Activities und Intents

Eine Android-Applikation kann aus einer oder mehreren von vier Basiskomponenten bestehen. Im Regelfall besitzt eine Applikation mindestens eine Activity. Activities haben immer eine grafische Oberfläche und bestehen normalerweise jeweils aus einem Fenster. Die Alternative dazu ist ein Service, der, wie eine Activity, Teil eines Programms sein kann, jedoch keine grafische Oberfläche besitzt und in der Regel in einem getrennten Prozess im Hintergrund läuft.

Zusätzlich gibt es noch BroadcastRecievers, die auf systemweite Ereignisse wie Telefonanrufe reagieren können, sowie ContentProviders, die anderen Applikationen Daten bereitstellen können. Diese sind aber für die zu entwerfende Applikation uninteressant und werden nicht weiter vertieft.

Da im Laufe der vorliegenden Arbeit eine visuelle Bedienschnittstelle realisiert werden soll, ist die Activity-Komponente die einzig wichtige der vier beschriebenen Komponenten und soll in den folgenden Abschnitten näher beschrieben werden.

Wie gesagt kann eine Applikation aus einer oder mehreren Activities bestehen, bei der jede Activity eine Ansicht darstellt, die wiederum aus mehreren View-Objekte besteht. View-Objekte werden im nächsten Kapitel näher behandelt. Ein Beispiel für eine Applikation mit mehreren Activities wäre ein Mediaplayer.

Dieser hat vier verschiedene Ansichten. Zum einen den eigentlichen Player, der Informationen über das aktuell abgespielte Lied liefert sowie Play-, Pause-, Vor- und Zurück-Tasten bietet. Desweiteren gibt es einen Mediabrowser, in dem man die lokale Musikdatenbank durchsuchen kann, um Lieder zu der aktuellen Playlist hinzuzufügen. Natürlich gibt es dann noch die Ansicht der aktuellen Playlist und eine Einstellungsansicht. Die Applikation hätte also vier Activities.

Das Revolutionäre an Activities ist, dass sie alle autonome Teile mit einem eigenen Lifecyclemanagement sind, deren Funktionalität in einem sogenannten Intentfilter gespeichert wird. Der Mediabrowser könnte beispielweise „Ich stelle eine Ansicht aller Musikdateien, sortiert nach Autor, Album oder Titel bereit“ in seinem Intentfilter stehen haben.

Zwischen Activities wird nun mit sogenannten „Intents“ navigiert. Will man von dem MediaPlayer zu dem MediaBrowser wechseln, wird beispielweise ein neuer Intent mit der Anfrage nach einer Activity, welche Medien darstellt, aufgegeben. Der ApplicationManager geht nun die IntentFilter aller installierten Activities durch und startet die am besten auf die Anfrage passende Activity. Dies könnte beispielweise auch der MediaBrowser einer ganz anderen Applikation sein.

Will man eine bestimmte Activity aufrufen, kann dies über den Klassennamen getan werden. In diesem Fall wird sofort die entsprechende Activity-Klasse gestartet, ohne nach anderen Alternativen zu suchen.

Aufgerufene Activities kommen immer an die oberste Stelle des Activity-Stacks, in dem sich alle laufenden Activities befinden. Dort bleiben sie bis der Benutzer eine andere Activity in den Vordergrund bringt und die neue Activity an die erste Stelle des Stacks geschoben wird.

Will man von einer Activity in die zuvor benutze Activity zurückkehren, geschieht dies über die physikalische „Zurück“-Taste auf dem Gerät. Dies macht es möglich, eine Nummer aus einer SMS über die „Contacts“-Activity zu speichern und von dort wahlweise mit der Camera-Activity oder der PhotoBrowser-Activity ein Bild für den Kontakt hinzuzufügen und mit einem Klick der Zurücktaste wieder in der SMS-Activity zu sein, um diese zu Ende lesen zu können. Die Art der Navigation erinnert, wie schon im Überblick erwähnt, an das Browsen im Web.

2.3.3 Activity Lifecycles

Auch die Lifecycles von Android-Applikationen haben wenig mit den Lifecycles normaler Java-Applikationen gemeinsam. Da man sich auf Embedded- und damit Ressourcenbeschränkten-Plattformen befindet, können nicht beliebig viele Activities gleichzeitig laufen. Nach einer gewissen Zeit füllt der Activity-Stack den Speicher und eine der gestarteten Activities muss zwangsweise geschlossen werden, um das Öffnen neuer Activities zu ermöglichen.

Dies wird über eine Priorisierung und ein komplexeres Lifecyclemanagent der Activities realisiert. Der Lebenszyklus jeder Activity wird durch Callback-Funktionen gehandhabt. Eine Activity kann sich in verschiedenen Zuständen

befinden, wobei bei Zustandsübergängen jeweils eine Callback-Methode aufgerufen wird.

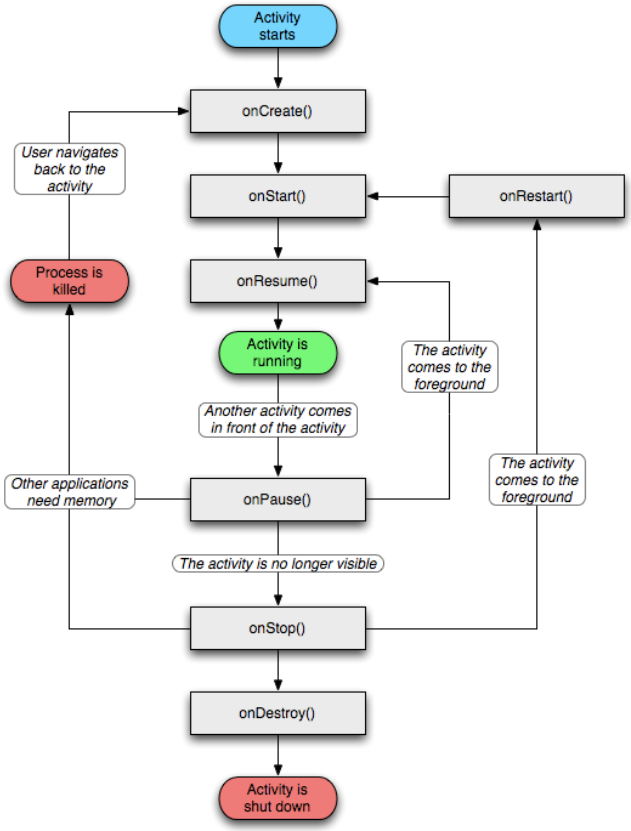


Abbildung 2: Activity Lifecycle [10]

Eine Activity kann sich in drei Zuständen befinden: Der erste Zustand ist „Beendet“. Dies bedeutet, dass die Applikation lokal im Flash liegt, nicht gestartet ist und nicht ausgeführt wird.

Der zweite Zustand ist „Aktiv“. Dies geschieht dann, wenn die Activity durch einen Intent aufgerufen wurde und die Activity visuell in den Vordergrund gebracht wird. Die Activity befindet sich im obersten Platz des Activity-Stacks und empfängt alle Benutzereingaben. Die Callback-Funktionen onCreate(), onStart() und onResume() werden bei dem Übergang von „Beendet“ nach „Aktiv“ aufgerufen.

Der dritte Zustand ist „gestoppt aber weiterhin im RAM“. Dies ist der Fall, wenn eine andere Activity in den Vordergrund und damit an die oberste Stelle des Activity-Stacks gebracht wird. Die vorherige Activity wird pausiert und gestoppt und verliert den Fokus, bleibt aber im RAM erhalten. Die Callback-Funktionen onPause() und onStop() werden aufgerufen.

Im dritten Zustand gibt es zwei mögliche Nächstzustände. Entweder der Benutzer bringt die Activity wieder in den Vordergrund, womit sie wieder in den zweiten Zustand gelangt (onRestart(), onStart() und onResume() werden aufgerufen). Oder aber das System benötigt den belegten Speicher und die Activity wird heruntergefahren und der Speicher freigegeben. Bei letzterem wird keine weitere Funktion mehr aufgerufen, die Applikation wird einfach beendet und befindet sich wieder im ersten Zustand.

Wird die Applikation aus dem „Aktiv“-Modus beendet, werden die Funktionen onPause(), onStop() und onDistroy() aufgerufen.

Es bleibt dem Entwickler überlassen in den entsprechenden Callback-Methoden seine Applikation aufzubauen, GUIs anzuzeigen und Daten zu speichern. So wird in der Regel in der onCreate()-Methode das GUI aufgebaut und zu speichernde Benutzerdaten in der onPause()-Methode persistent gemacht, um Datenverluste zu verhindern.

2.3.4 GUI Framework

Das Android GUI Framework ist zuerst einmal, wie die meisten GUIs, nach dem Composite Design-Pattern [16] aufgebaut. Das bedeutet, dass sowohl die sichtbaren Objekte, als auch die Behälterobjekte, in denen sich diese befinden, von einer Klasse, in diesem Fall der View-Klasse, abgeleitet sind.

Dies ermöglicht komplexe GUIs zu entwickeln, die beliebig tief geschachtelt werden können. Grenzen setzt nur die Größe des Displays.

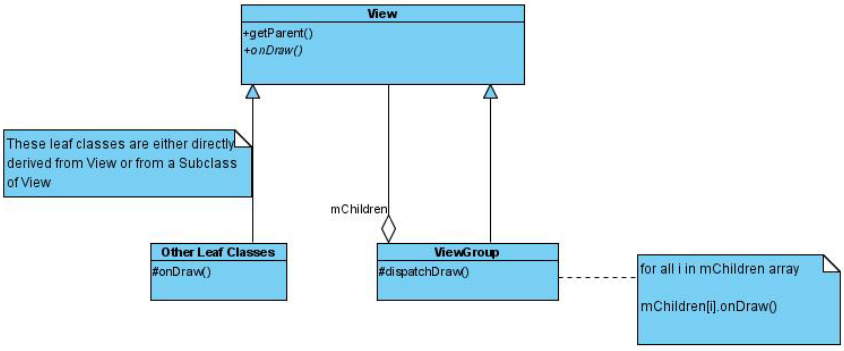


Abbildung 3: Die View Klasse

Während die Blatt-Klassen direkt oder durch Subklassen von View erben, wird zusätzlich die ViewGroup-Klasse eingeführt, die als Behälterklasse für die Blätter dient.

Das Positionieren und Zeichnen von Objekten wird jeweils von dem übergeordneten Objekt durchgeführt. Dem Wurzelknoten, der im Normalfall ein Objekt der Klasse ViewGroup ist, steht der ganze Bildschirm zum Zeichnen zur Verfügung. Dieser löst den Zeichnen-Prozess aus. Er gibt all seinen Kindern mit Hilfe der onMeasure()-Methode die Aufforderung ihre Größe zu bestimmen. Je nachdem, ob das Kind selber eine ViewGroup oder ein Endknoten ist, misst es seine Größe oder gibt den Befehl wiederum an seine Kinder weiter.

Die gemessenen Größen sollen dem Elternobjekt als Referenz dienen, um festzustellen wie viel Platz den Kindern zur Verfügung gestellt werden sollte. Allerdings müssen diese nicht zwingend eingehalten werden. Das Elternobjekt besitzt alle Freiheiten, die Größen nach Bedarf zu ändern. Nachdem das Elternobjekt die Kinder nach seinem Wunsch positioniert und skaliert hat, ruft es für jedes Kind die onLayout()-Methode mit den entsprechenden Maßen und der Position, die das Objekt einnehmen darf, auf. Das Kind passt seine Größe den Vorgaben an und gibt die Maße wiederum seine Kinder weiter. Ist das Layouting abgeschlossen, wird mit dem gleichen Verfahren die onDraw() Methode aufgerufen, in der die Objekte dann schlußendlich in spezifizierter Größe und Position auf das Display gezeichnet werden.

Das Android Application Framework bringt eine Reihe von „Blatt“- und „Behälter“-Klassen, sogenannte Widgets, mit, die für das Bauen der meisten Applikationen ausreichen. Hierzu zählen von ViewGroup abgeleitete Klassen wie TableLayout, RelativeLayout oder AbsoluteLayout, die es ermöglichen Kinder auf verschiedene Weise zu positionieren. Desweiteren stehen direkt oder indirekt von View abgeleitete Klassen wie TextView, Button, ImageButton, ListView oder ProgressBar zur Verfügung. Alle Views sind nach dem Android Standard Look and Feel gestaltet.

2.3.5 GUI-Design

Das Implementieren von GUIs findet über XML statt. Grund hierfür war der Wunsch der Android Entwickler, das Implementieren der grafischen Oberfläche aus dem Java Code zu nehmen. Dies hat vor allem zwei Vorteile. Der erste Vorteil ist die Übersichtlichkeit. Wer bereits SWT oder Swing GUIs von Hand geschrieben hat weiß, dass der Code sehr unübersichtlich und aufgrund dessen schwer anpassbar ist. Zusätzlich gibt es oft enge Verstrickungen zwischen Pro-

gramm-Logik, wie beispielweise die ButtonListener, und der eigentlichen GUI. Das Extrahieren des GUI-Layouting in XML erleichtert die Übersichtlichkeit und hält Darstellung von Logik getrennt.

Der zweite Grund ist das Implementieren selbst. Die Möglichkeit GUIs getrennt von der Software zu erstellen, erleichtert die Arbeitsteilung um ein Vielfaches. Während der Logik-Programmierer ungestört seinen Quellcode ändern und pflegen kann, hat der GUI-Programmierer alle Freiheiten sein Design zu entwickeln ohne in Quellcode-Konflikte mit dem Logik-Programmierer zu kommen. Ob der Entwickler die Vorteile der XML-GUI-Beschreibung nutzen will oder sich doch in Java sicherer fühlt, bleibt alleine ihm überlassen.

Die Struktur des XML-GUI-Layouts entspricht dem entstehenden View-Baum und erleichtert dadurch die Navigation durch den Code. Objekte können in beliebiger Verschachtelung miteinander verbunden werden. Um beispielweise ein Formular zu erstellen, in dem man seinen Namen eingeben und diesen mit einem Knopf bestätigen kann, wäre folgendes Layoutfile denkbar:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:id="@+id/mein_behälter"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="horizontal">
            <TextView
            android:id="@+id/tv_name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Name: "/>
            <EditText
            android:id="@+id/et_name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Geben Sie hier Ihren Namen ein"
            android:textSize="12sp"/>
            <Button
            android:id="@+id/btn_ok"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="ok"/>
</LinearLayout>
```

Abbildung 4 zeigt das soeben definierte GUI-Layout auf einem Android Device.

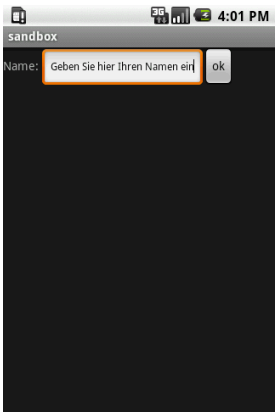


Abbildung 4: Darstellung des in XML definierten Layouts

Über die, für jedes Objekt definierte android:id, kann später aus dem Java Code auf die erstellen Objekte zurückgegriffen werden. Mit weiteren Parametern ist es möglich, die Position des Kindes im Elternelement zu definieren, Eigenschaften des speziellen Objekts zu verändern sowie die Bilder, die für die Objekttexturen verwendet wurden, zu ersetzten. Dies ermöglicht es, dem GUI-Layouter mit wenig Coding-Aufwand eigene Styles von Objekten zu erstellen.

2.3.6 Resources

Android erlaubt es verschiedene externe Dateien in das Programm einzubinden. Hierfür steht im Android-Projekt ein Ressource-Ordner zur Verfügung, in dessen Unterordner die entsprechenden Dateien kopiert werden müssen, um aus dem Java-Quellcode darauf zugreifen zu können.

Diese Dateien werden Resources genannt und teilen sich in verschiedene Kategorien ein. Für jede Kategorie gibt es in dem Resource Ordner einen eigenen Unterordner. Die Layout Resources, in denen die XML-GUI-Layouts gespeichert werden, wurden im vorherigen Kapitel bereits vorgestellt.

Desweiteren gibt es Drawable Resources, zu denen PNG Grafiken gehören, die später in der Software, beispielweise als Hintergründe oder Buttons, dienen können.

Zudem gibt es Value Resources. Hier werden wiederum XML-Dateien gespeichert, die beliebige String-Werte, Int-Werte oder Arrays, jeweils versehen mit einer ID, beinhalten können. Diese Values stehen dann jeder Klasse im Programm zur Verfügung und sind nützlich, um globale Angaben wie beispielweise den Programmnamen oder die Version zu speichern.

Die letze Sorte von Resources sind Binary Resources. Hier können beliebige binäre Dateien, wie Audio-Dateien oder Videos gespeichert werden, auf die später aus der Software zugegriffen werden kann.

Aus dem Java-Code wird auf die Resources immer über ihre spezifizierte ID, bzw. bei binären Dateien und Bildern über ihre Namen zugegriffen. Android bietet für jede Resource-Kategorie eigene, sogenannte Resourceloader an.

2.3.7 Testing

Für das automatisierte Testen von Programmen stellt Android das gesamte Unit-Framework zur Verfügung. Zusätzlich gibt es spezielle TestUnit-Klassen, die es ermöglichen, Tests auf der GUI-Ebene zu schreiben. Somit ist ein vollwertiges Test-Driven-Development möglich.

2.4 Android Entwicklungsumgebung

Die Android Entwicklungsumgebung setzt sich aus verschiedenen Tools zusammen und soll insgesamt alles anbieten, um eine Applikation implementieren, testen, debuggen, profilen, auf Geräte spielen und in den Android Market publizieren zu können. Die wichtigsten Tools sollen hier kurz vorgestellt werden.

Eclipse

Als Programmierumgebung dient die, vor allem von Java-Entwicklern bevorzuge, Integrated Development Environment (IDE) Eclipse. Google bietet zwei Plugins für die IDE, die einige Konfigurationsfeatures sowie eine neue Eclipse-Ansicht, genannt DDMS, zur Verfügung stellen.

Entwickelt wird zunächst einmal in der üblichen Java-Ansicht. Diese wird um spezielle Editoren für Android spezifischen XML-Dateien sowie einige Android spezifische Konfigurationsmenüs erweitert. Desweiteren können aus Eclipse heraus Android Emulatoren erstellt und verwaltet werden. Hierbei handelt es sich um vollwertige ARM-Emulationen. Es wird die komplette Andoird-Plattform von Linux Kernel über C-Bibliotheken bis zum Android Framework emuliert.

Über einen Run-Konfigurator kann bestimmt werden auf welchem Emulator oder auf welchem angeschlossenen Hardware-Device die Android-Software

ausgeführt werden soll. Debugging kann ebenfalls wahlweise auf einem Emulator oder einem Hardware-Device stattfinden.

Die Dalvik Debug Monitor Service Ansicht (DDMS) gibt dem Entwickler einen tiefen Einblick in das laufende System eines Gerätes oder Emulators. Hier können alle laufenden Programme, deren Threads sowie Speicherverbrauch und Logs eingesehen werden. Zusätzlich erlaubt diese Ansicht es, durch sämtliche Dateien des Gerätes zu browsen sowie Anrufe und SMS-Nachrichten auf einem entsprechenden Emulator oder angeschlossene Hardware-Device zu simulieren.

Android Debug Bridge (ADB)

Dies alles wird über die Android Debug Bridge ermöglicht. Es handelt sich hierbei um eine TCP Server-Client Applikation, die auf jedem Android Device läuft und es dem Host ermöglicht auf das Device zuzugreifen und auf verschiedene, allgemeine und Android spezifische Funktionen zurückzugreifen.

Die Android Debug Bride kann ebenfalls über die Konsole benutzt werden. Es stehen hier Funktionen wie eine primitive Linux Shell, das Installieren von Applikationen oder das Lesen und Schreiben von Dateien zur Verfügung. Hardware Devices können entweder über eine USB-Kabelverbindung oder aber über das Netzwerk angesprochen werden.

Weitere Tools

Außer diesen essentiellen Tools, stellt die OHA eine ganze Reihe von weiteren Tools zur Verfügung, die für GUI-Erstellung, Profiling, GUI-Layout-Analyse und weitere Zwecke einsetzbar sind.

2.5 Hardware Einführung

2.5.1 Android Devices

Wie bereits erwähnt, soll die Visualisierungs- und Bediensoftware auf einem Handheld sowie einem stationären Device mit größerem Display ausgeführt werden. Da Android ein offenes Projekt ist und auf Linux basiert, sollte eine Portierung auf eine dritte Hardware durchaus machbar sein.

Als Handheld soll das HTC Dream [11], auch bekannt als T-Mobile G1, in einer speziellen Entwicklerversion dienen, welche Android Development Phone genannt wird. Das Gerät kann direkt bei Google ohne Vertragsbindung bezogen werden und verfügt über Root-Rechte, welche die Entwicklung auf dem Gerät erleichtern.

Das Gerät verfügt über ein kapazitives 3,2“ Touch-Display mit einer Auflösung von 320x480 Pixeln. Die Rechenleistung stellt ein 32Bit ARMv6 Prozessor in RISC Architektur, der mit 528Mhz getaktet ist. Das Gerät verfügt über 256MB RAM.

Als zweite Plattform soll entweder ein X86 Industrie Touch-PC oder ein ARM-Device dienen, das mindestens über ein 10“ Touch-Display verfügt und im Laufe der Arbeit noch evaluiert werden soll.

2.5.2 Sauter Modu525

Die Sauter Modu525 ist eine Gebäudeautomationsstation der neusten Generation. Sie ist eine PowerPC Architektur, taktet mit 500 MHz und verfügt über eine 10/100Mbit Ethernet Schnittstelle.

Nativ unterstützt sie das BACnet Protokoll sowie prototypenweise Webservices, um über das IP-Netz die gesteuerten Datenpunkte auslesen und setzen zu können. Die Datenpunkte selbst können über verschiedene Arten von Steuerleitungen an die Automationsstation angeschlossen werden. Die Station kann über Module erweitert werden, um entsprechend mehr Datenpunkte auszulesen oder regeln zu können. Weitere Details über die Anschlussmöglichkeiten der Datenpunkt können Kapitel 4.3 entnommen werden.

3 Anforderungen

3.1 Allgemeine Anforderungen

Es soll ein verteiltes System aufgebaut werden, dass aus drei Hauptkomponenten besteht:

- Mechanisches Modell mit Datenpunkten, die gesteuert werden.
- Automationsstation Modu525, welche die Regelung und Ansteuerung der Datenpunkte übernimmt.
- Visualisierung und Bedienung durch eine Android-Applikation, die auf unterschiedlicher Hardware läuft.

Anhand dieses Systems soll die Nutzbarkeit der Android-Applikation als Visualisierungs- und Bedienungsschnittstelle evaluiert und beurteilt werden. Hierbei sollen folgende Punkte genauer untersucht werden:

- Diskussion / Analyse: Eignet sich Android als SW-Plattform für Visualisierung und Bedienung der Gebäudeautomation?
- Ist ein Port auf x86 möglich?
- Wie ist das „Look and Feel“ der Beispielsoftware, wie aufwändig ist es zu erstellen?
- Wie schnell reagiert die AS bzw. das Modell auf Bedieneingaben der Beispielsoftware?
- Wie effektiv ist der Workflow für die Erstellung einer Android-Applikation?
- Wie gut sind Debugging und Profiling möglich?

3.2 Modell Anforderungen

- Es soll ein mechanisches Modell konstruiert werden, das ein für Sauter typisches Umfeld aufweist, in dem verschiedene Datenpunkte, die physikalisch reagieren, angesteuert werden können.
- Hierbei müssen verschiedene Arten von Datenpunkte existieren, die analog, binär und als Multistate angesteuert werden können.
- Es müssen genügend Datenpunkte vorhanden sein, um sinnvolle Szenarien definieren zu können.

3.3 Anforderungen an die Regelung

- Die Regelung soll über einen Regelungsplan mit der Sauter Software Case Engine erstellt werden und sämtliche Datenpunkte im Modell ansteuern.
- Der Plan soll es ermöglichen, die Datenpunkte entweder einzeln oder übergreifend mit Szenarien zu steuern.
- Die gesamte Regelungslogik des Systems soll in dem Regelungsplan liegen. Es ist nicht gewünscht, dass die Bedienungs- und Visualisierungs-Software Teile der Regelung übernimmt.

3.4 Softwareanforderungen

3.4.1 Überblick

Es soll eine Android-Applikation entwickelt werden, die eine Visualisierung und Bedienung des Modells erlaubt. Das Programm soll sowohl auf einem HTC Dream sowie einer x86 oder ARM Plattform mit großem (min. 10“) Display ausgeführt und getestet werden.

Die entsprechende Hardware soll mit einem, noch zu evaluierenden Netzwerkprotokoll über LAN, WLAN und Mobilfunknetze mit der AS Modu525 kommunizieren können. Die Android-Applikation soll eine reine Visualisierungs- und Bedienungsschnittstelle zwischen den von der AS geregelten Datenpunkten und dem Anwender sein und selbst nichts regeln. Die Software soll folgende Anforderungen erfüllen:

3.4.2 Funktional

- Die Software soll die Möglichkeit bieten sämtliche Datenpunkte des Modells einzeln zu bedienen sowie die vordefinierten Szenarien zu benutzen.
- Die Intelligenz für die Szenarien soll im Regelungsplan und nicht in der Android-Software liegen.
- Der aktuelle Zustand eines Datenpunktes sollte erkennbar sein, auch wenn dieser von Dritten geändert wurde.
- Die Position der Datenpunkte im Modell soll in der Software erkennbar gemacht werden.

3.4.3 Netzwerk

- Die Software soll über eine drahtlose Schnittstelle mit der Automationsstatioin AS525 kommunizieren und von dieser aktuelle Werte abholen und neue Werte setzen können.
- Das Netzwerkprotokoll muss schnell und robust sein. Protokolle, die die AS525 bereits unterstützt, sollten bevorzugt werden.
- Das Protokoll muss, je nach Einsatzbereich (LAN / WAN), ausreichend gesichert sein bzw. gesichert werden können.

3.4.4 GUI

Die Software soll die Möglichkeit bieten folgende Arten der Regelung zu bedienen:

- Knöpfe (Ein / Aus)
- Multistates (Auswahl von einer definierte Anzahl an Zuständen)
- Zieh oder Drehregler für analoge Werte

Die Oberfläche und Navigation durch das Programm soll intuitiv, zeitgemäß und ansprechend sein. Eine Visualisierung des Modells ist gewünscht, sodass eine Person, welche die Software zum ersten Mal benutzt, sich darin schnell zurechtfinden und ihre Position, bzw. die Position der Datenpunkte im Modell schnell ausmachen kann.

Die Bedienelemente sollen die Möglichkeiten des Touchs möglichst gut umsetzen und Dreh- und/ oder Ziehmechanismen einsetzten. Ziel ist es, weg von der steifen Steuerung einer HTML Seite, hin zu einer fließenden, intuitiven Touch-Steuerung zu kommen.

Die GUI soll sowohl auf einem 3,2“ als auch auf einem großen Display von min. 10“ ohne Anpassungen einsetzbar sein, was eine Skalierbarkeit der Darstellungen fordert.

3.4.5 Nicht Funktional

- Zwischen der Betätigung eines Bedienelements und der physikalischen Reaktion darf höchstens eine Sekunde vergehen.

- Das Programm soll für Mobile Devices optimiert, d.h. mit möglichst wenig Prozessorlast und langen Leerlaufzeiten, umgesetzt werden.
- Die Software soll so implementiert werden, dass der Einsatz eines anderen Netzwerkprotokolls minimalen Aufwand in der Anpassung des Codes bedeutet.
- Die Software soll modular erweiterbar und auf einer generischen Basis gebaut werden, um die Möglichkeit offen zu lassen aus dem Prototyp ein Produkt zu entwickeln.
- Für den Prototyp sollen kein Rollen und Rechtesystem implementiert werden. Es muss darauf geachtet werden, dass sowohl eine Systemfremder, eine Putzfrau, als auch ein Hausmeister die Software gleichermaßen bedienen können. Es sollen jedoch keine Rechte für Benutzer vergeben werden.
- Auch eine Mehrsprachigkeit ist für den Prototyp nicht erwünscht. Eine deutsche Sprachausführung ist ausreichend.
- Es ist wichtig, dass die Software schnell auf Benutzereingaben reagiert. Auch wenn es zu längeren Reaktionszeiten durch die Netzwerkübertragungen kommt, sollte der Benutzer trotzdem das GUI benutzen können, ohne dass dieses einfriert und auf das Netzwerk wartet.
- Da auf der 10“ Hardware im Gegensatz zum HTC Dream keine zusätzlichen physikalischen Navigations-Knöpfe vorgesehen sind, muss die Software ausschließlich über das Touch-Display steuerbar sein.

3.5 Hardwareanforderungen für das zweite Android Device

Mit dem HTC Dream steht die erste Hardwareplattform für die Android-Applikation fest. Für die zweite Plattform kommen verschiedene Lösungen in Frage. Die Beste soll im Rahmen dieser Arbeit gefunden und Android darauf portiert werden. Folgende Anforderungen sind essentiell:

- LCD Display von min. 10“ mit einem reaktiven Touch-Display.
- ARM oder X86 Architektur mit min. 400MHz
- Min 64MB RAM
- 100MBit Ethernet Netzwerkschnittstelle
- Realistische Chancen Android innerhalb von 2 Wochen zu Portieren.

3.6 Vorgehensweise

Ziel der nächsten Schritte ist es, die einzelnen Komponenten des Systems zu designen und aufzubauen. Um dies zu ermöglichen, ist eine Gesamtübersicht mit den Rollen der Komponenten wichtig, wie es auf Abbildung 5 zu sehen ist.

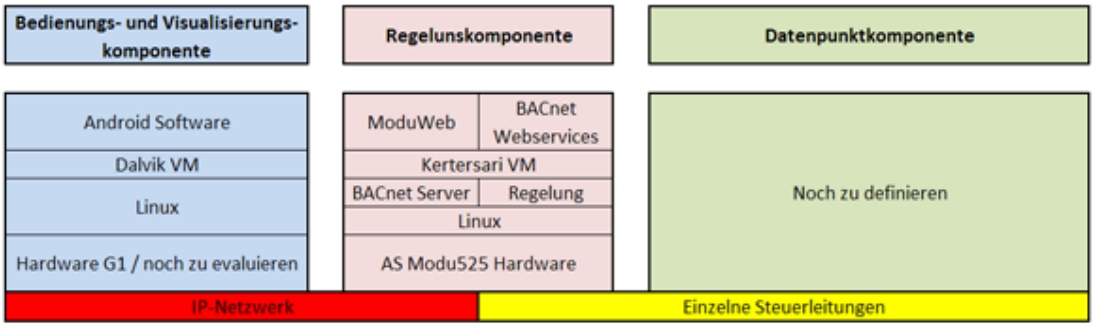


Abbildung 5: Systemkomponenten

Das System soll wie definiert in drei Komponenten geteilt werden. Die Grundliegende Komponente sind die physikalischen Datenpunkte, die über Steuerleitungen von der Regelungskomponente, einer AS Modu525, gesteuert werden. In der Datenpunktkomptonente ist kaum Logik vorhanden, es geht hier hauptsächlich um die richtige Verkabelung und Ansteuerung der Hardware. Diese gilt es als Erstes zu definieren und umzusetzen.

Auf der Datenpunktkomponente aufbauend soll anschließend der Regelungsplan erstellt werden, der das Herz der Regelungskomponente ist. Hier soll sich sämtliche Regelungslogik befinden.

Zuletzt soll die Visualisierungs- und Bedienkomponente in Form einer Android-Applikation designt und umgesetzt werden. Sie wird über das Netzwerk (Wahlweise UMTS / WLAN / Ethernet) mit der Automationsstation kommunizieren und entsprechende Regelungsprozesse auslösen und stoppen. Dem Benutzer soll die Bedienungs- und Visualisierungskomponente als transparente Schnittstelle zu den Datenpunkten dienen.

4 Modell

4.1 Allgemein

Um ein praxisbezogenes Modell zu erstellen, wird ein Treffen mit dem Sauter Produktmanager Herr Cristobal Fernandez arrangiert, dem viele praktische Anwendungsszenarien für die AS Modu525 bekannt sind. Es sollen echte Anwendungsfällte mit dem Bedarf nach einer Lösung, wie sie Android bieten könnte, zusammengetragen und ein entsprechendes Modell daraus definiert werden.

In der Besprechung mit Herr Cristobal zeigt sich, dass ein Konferenzzimmer, das verschiedene Licht und Klimaszenarien besitzt und durch den Redner gesteuert werden kann, eine aktuelle Kundenanforderung ist.

Um die nötige Vielfalt an Datenpunkten und ein in sich geschlossen Model zu erhalten, wird die Entscheidung getroffen ein Auditorium zu modellieren. Das Modell soll Beleuchtung für Bühne, Tribüne und Foyer enthalten. Zusätzlich soll es eine Klimaregelung in Form von Heizung, Lüftung und einem Fenster eingebaut werden.

Das Modell soll als 3D-Objekt mit dem kostenfreien Google SketchUp erstellt und anschließend in der Sauter Mechanik Lehrwerkstatt gebaut werden.

4.2 Modell Design

Das 3D-Modell wurde aufgrundlage von Uly Ramirez‘ „Modern Theather“ gebaut, dass im Google 3D-Warehouse [12] lizenzfrei bezogen werden kann und auf Abbildung 6 zu sehen ist.

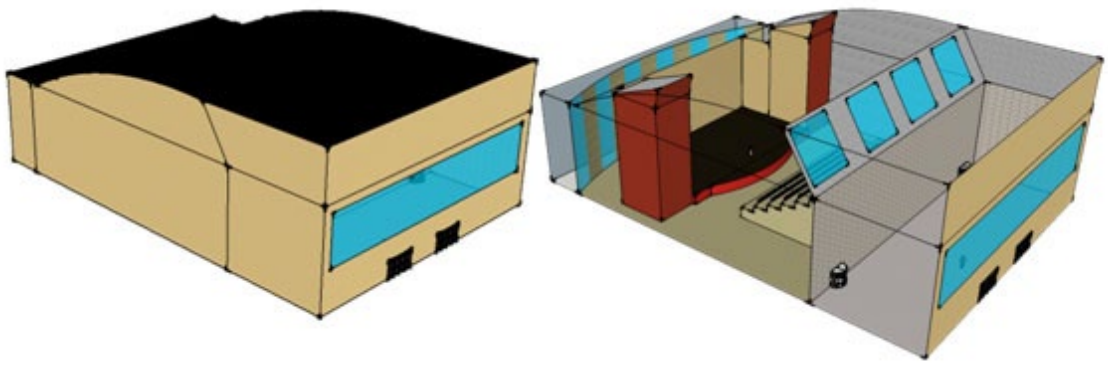


Abbildung 6: Originalmodell von Uly Ramirez

Das Modell wurde umfassend angepasst und erweitert, um den Anforderungen zu entsprechen. So wurde unter anderem ein Fenster in das obere Drittel der Wand hinter der Tribüne eingelassen, eine Lüftung in die Seite des Gebäudes gelegt, sowie eine Leinwand und Projektor für Präsentationen hinzugefügt. Die Tribüne wurde verbreitert und die Stuhlreihen angepasst. Die hohen Reihenunterschiede und wenigen Stufen der Tribüne wurden zu Gunsten des Modellbaus gewählt. Die Texturwahl fiel auf eine Kombination aus Parkett, Betonboden und weißen Raufaserwänden. Um die Bühnenkonstruktion hervorzuheben, wurde diese mit einem hellen Blau versehen.

Insgesamt wurden die Farben hell und blass gewählt, um den später entstehenden Screenshots Neutralität zu verleihen und das Bild nicht zu überladen. Das Modell wurde in zwei Räume aufgeteilt (die ab jetzt als Auditorium und Foyer bezeichnet werden), um in der Software beispielhaft eine Raumnavigation zeigen zu können.

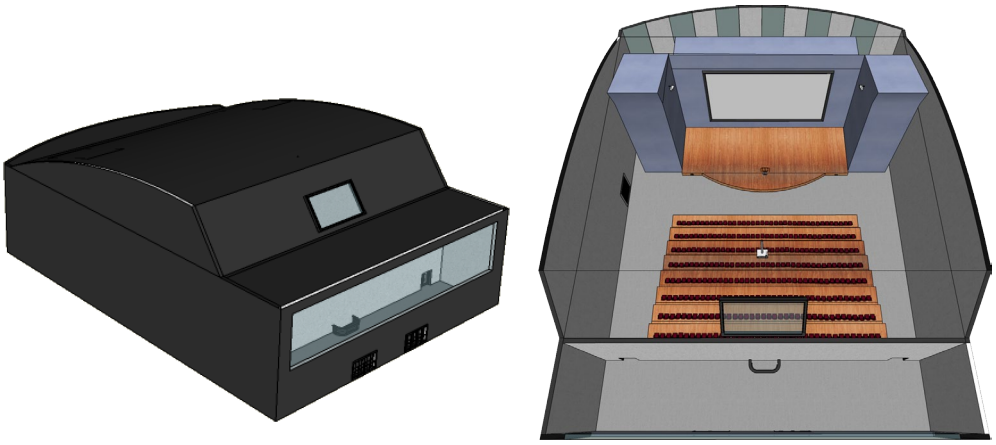


Abbildung 7: Angepasstes Modell für die Automation

4.3 Automation im Modell

4.3.1 Datenpunkte

Um eine repräsentative Regelung zu ermöglichen und verschiedene Szenarien zeigen zu können, wird das Modell mit Datenpunkten ausgestattet, die über die AS Modu525 geregelt werde sollen. Hierbei ist es wichtig verschiedene Datenpunkt-Typen und verschieden zu steuernde Verbraucher zu integrieren, um in der zu entwickelnden Software verschiedene Möglichkeiten der Visualisierung zeigen zu können. Die in das Modell zu integrierenden Datenpunkte werden in Tabelle 1 und Tabelle 2 aufgelistet.

Auditorium			
Anzahl	**Datenpunkt Typ**	**Datenpunkt**	**Werte**
2	Analog	Licht Bühne, dimmbar	0-100
1	Analog	Spotlight Bühne, dimmbar	0-100
2	Analog	Licht Tribüne, dimmbar	0-100
1	Binär	Projektor	an, aus
1	Multistate	Lüftung	0, 1, 2, 3
1	Multistate	Fenster	zu, gekippt, offen
1	Multistate	Heizung	0, 1, 2, 3
1	Analog	Temperatursensor	15 - 35
4	Binär	Notbeleuchtung	an, aus

Tabelle 1: Datenpunkte im Auditorium

Foyer			
Anzahl	**Datenpunkt Typ**	**Datenpunkt**	**Werte**
1	Binär	Licht Foyer	an, aus
1	Binär	Notbeleuchtung	an, aus

Tabelle 2: Datenpunkte im Foyer

Die Notbeleuchtung soll zentral für das ganze Gebäude geregelt werden. Das bedeutet, dass sie im ganzen Gebäude entweder an oder aus ist. Die Notbeleuchtung soll aus 5 kleinen grünen LEDs bestehen, die jeweils über den Türen, und links und rechts an der Bühne angebracht sind, um Besuchern die Fluchtwege zu weisen. Alle anderen Datenpunkte sollen explizit in dem entsprechenden Raum vorhanden sein.

4.3.2 Szenarien

Die zentrale Steuerung der Datenpunkte erlaubt es dem Benutzer neben der einzelnen Bedienung von Datenpunkte sogenannte Szenarien anzubieten. Hierbei handelt es sich um Voreinstellungen einer bestimmten Auswahl von Datenpunkten, die häufig vorkommen und dem Benutzer Zeit und Mühe bei der einzelnen Einstellung der verschiedenen Werte sparen sollen. Da es keinen Sinn macht, für die zwei Datenpunkte im Foyer eigene Szenarien zu definieren, sollen alle Szenarien gebäudeübergreifend sein. Um dem Benutzer die Einstellung so leicht wie möglich zu machen, werden zwei hierarchische Ebenen von Szenarien umgesetzt.

Die untere Ebene teilt sich wiederum in zwei Kategorien auf. Zum einen Szenarien für Licht und zum anderen Szenarien für Klima. Wäre außer dem

Projektor noch eine Audioquelle vorhanden, wäre eine weitere Kategorie „Multimedia“ denkbar. Da dies aber nicht der Fall ist, wird der Projektor in die Licht-Szenarien integriert.

Da mit den klimabetreffenden Datenpunkten im Modell lediglich eine Temperatursteuerung möglich ist, sollen die Klimaszenarien sich auf eine Soll-Temperatur-Regelung beschränken, die durch das automatische Schalten von Lüftung, Fenster und Heizung erreicht werden soll.

Eine Ebene darüber soll es kategorieübergreifende Szenarien geben, die sowohl Licht als auch Klima steuern.

„Licht & Video“ Szenarien						
Szenario	**Licht Bühne**	**Spot Bühne**	**Projektor**	**Licht Tribüne**	**Licht Foyer**	**Notbeleuchtung**
Einlass	0%	0%	aus	100%	ein	ein
Präsentation	80%	100%	aus	0%	alter Wert	ein
Präsentation mit Beamer	0%	60%	ein	0%	alter Wert	ein
Filmvorführung	0%	0%	ein	0%	alter Wert	ein
Reinigung	100%	100%	aus	100%	ein	ein
Notbeleuchtung	0%	0%	aus	0%	aus	ein
Manuell	manuell	manuell	manuell	manuell	manuell	manuell

Tabelle 3: Datenpunktwerte für „Licht & Video“ Szenarien

Die in Tabelle 3 dargestellten Szenarien sollen beispielhaft zeigen, wie eine solche Szenariensteuerung aussehen könnte. Durch das Schalten eines der Szenarien sollen alle betroffen Datenpunkte auf den vordefinierten Wert gesetzt werden. Wird ein Datenpunkt, der in die „Licht & Video“ Kategorie gehört manuell geändert, soll das aktive Szenario auf „Manuell“ springen, da es sich dann nicht mehr in dem vordefinierten Wert befindet. Wird das Szenario von Hand auf „Manuell“ geändert, sollen die momentanen Datenpunktwerte beibehalten werden.

Das Klimaszenario soll sich auf die Soll-Temperatur beschränken. Hier soll dem Benutzer, ähnlich wie bei einer Klimaanlage, erlaubt werden eine Temperatur zwischen 15°C und 35°C zu wählen, die die Automationsstation dann durch die Regelung von Fenster, Lüftung und Heizung annähert.

Um dies umzusetzen, wird allen drei Datenpunkten der Zustand „auto“ hinzugefügt, in dem die AS dann die Regelung übernimmt. Sobald der Benutzer

die Solltemperatur verändert, sollen die drei Datenpunkte automatisch in den „auto“-Zustand gehen, sofern sie das noch nicht sind.

Die Gebäudeszenarien sollen sich im Modell auf die Zustände „Belegt“ und „Nicht Belegt“ beschränken. Ziel dabei ist es, das Ausschalten aller Lichter und das Zurücksetzen der Solltemperatur beim Verlassen des Gebäudes zu erleichtern.

Gebäude Szenarien		
Szenario	**„Licht & Video“**	**Soll-Temperatur**
Nicht Belegt	Notbeleuchtung	22°C
Belegt	Manuell	manuell

Tabelle 4: Datenpunktwerte für Gebäude Szenarien

Wird das Szenario auf „Nicht Belegt“ gestellt, werden alle Lichter außer der Notbeleuchtung ausgeschaltet und die Solltemperatur auf 22°C gesetzt. Sobald der Benutzer die Solltemperatur oder ein „Licht & Video“-Datenpunkt verändert, soll das Szenario automatisch auf „Belegt“ springen. Bei einem manuellen Wechsel auf „Belegt“ sollen die letzten Werte einfach übernommen werden.

4.3.3 Bedienterminals

Bedienterminals wären an verschiedenen Stellen denkbar. Zum einen sollten neben der Eingangstür sowie den zwei Türen, die ins Auditorium führen, Bedienterminals angebracht sein. Diese sollten zugangsgeschützt sein und wären für das Personal vorgesehen.

Zusätzlich sollte am Rednerpult ein weiteres Terminal zur Verfügung stehen, welches für den Vortragenden gedacht ist. Für den Hausmeister käme ein Handheld in Frage, sodass er von überall den aktuellen Status abprüfen kann. Auch für den Redner wäre ein Handheld eine Alternative, falls kein Rednerpult benutzt wird.

4.4 Modell Aufbau

Für das mechanische Modell müssen zum einen die Maße und zum anderen die Materialien bestimmt werden. Die Maße wurden so gewählt, dass das Modell ausreichend groß ist, um verschiedene Bereiche innerhalb des Auditoriums zu beleuchten, ohne dass das gesamte Modell erhellt wird. Auch die Transportabilität des Modells spielte eine Rolle bei der Wahl der Größe.

Um eine gute Einsicht in das Modell zu erlauben, aber trotzdem nicht zu viel Licht von außen in das Modell eindringen zu lassen, sollen die Seiten aus durchsichtigem und das Dach aus lichtdichten Material hergestellt werden. Dies erlaubt es, das Modell in einem Raum mit Deckenbeleuchtung zu platzieren und dennoch die Lichtszenarien erkennen zu können. Auf Abbildung 8 sind die, der Mechanik Werkstatt überreichten, Größenangaben des Modells zu sehen.

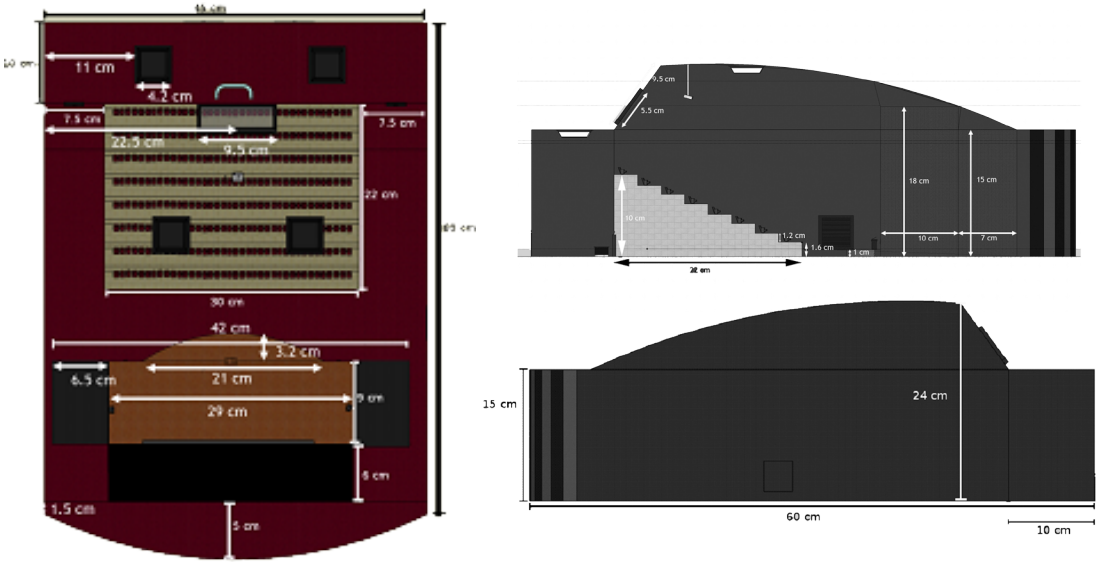


Abbildung 8: Modell Maße

Nach der Definition der Maße und Materialien wurde die Umsetzung des Modells an die Sauter Lehrwerkstatt übergeben, wo das Modell akkurat und zufriedenstellend aufgebaut wurde. Die Seiten, Fenster und Türen wurden aus Plexiglas gefertigt. Für die Notbeleuchtungen wurden spezielle, kleine, grüne Plexiglas-Würfel ausgefräst, in welche die LEDs eingelassen wurden. Alle anderen Teile wurden aus Aluminium gefräst. Alle Teile wurden mit Gewinden versehen, sodass das Modell leicht zusammen und auseinander geschraubt werden kann.

In den Würfeln neben der Bühne, dem Raum hinter der Bühne, sowie unter der Tribüne wurden Löcher in die Aluminiumbodenplatte gefräst, um dort Kabel verlegen zu können. Das Ganze wurde auf eine Holzplatte geschraubt, die unter dem Modell ausgehöhlt wurde, um das Durchführen von Kabeln zu ermöglichen. Abbildung 9 zeigt das fertige Modell.

Abbildung 9: Modell nach der Fertigstellung in der Mechanik Werkstatt

4.5 Elektronik

4.5.1 Allgemein

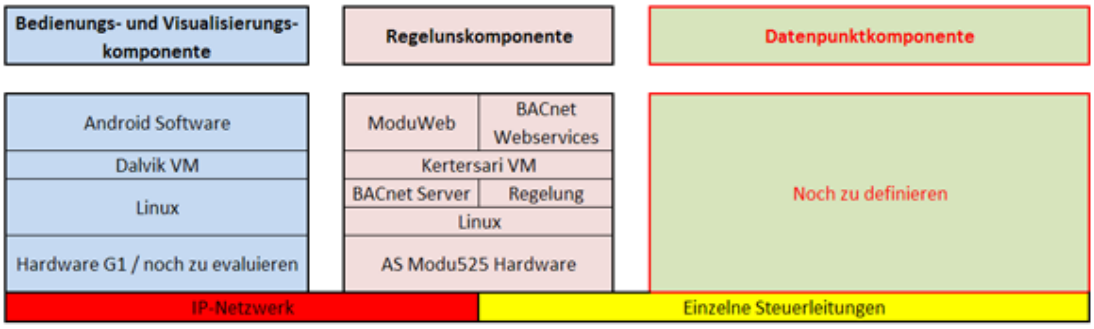


Abbildung 10: Zu definierender Komponente: Datenpunkte

Für die Umsetzung der Datenpunktkomponente sollen verschiedene elektronische Bauteile in das mechanische Modell integriert werden. Sämtliche Raum-Beleuchtung soll mit Power LEDs gelöst werden, während für die Notbeleuchtung normale, kleine, grüne LEDs ausreichen.

Als Lüftung soll ein PC-Gehäuse-Lüfter dienen, der in die Seite des Modells eingelassen wird und Luft in das Modell blasen soll. Damit kein Überdruck entsteht, soll das Fenster über einen kleinen Motor geöffnet werden. Hierfür soll ein Mikro Servo aus dem Modellbau eingesetzt werden, der einen Drehwinkel von 90° besitzt und somit das Fenster über eine Hebelmechanik auf und zu ziehen kann. Für die Heizung sollen Heizwiderstände eingesetzt werden, die sich unter Strom erhitzen. Der Wärmegrad kann dann über ein PWM Signal oder die Regulierung des Stroms bestimmt werden. Für das Auslesen der Temperatur im Modell soll ein Nickel 1000 Sensor verwendet werden, wie er auch in den Sauter Klima Messelementen zum Einsatz kommt.

Da die Heizwiderstände nicht genügend Wärme abgeben, um das ganze Modell zu beheizen, sollen sie unmittelbar vor den Lüfter platziert werden. Der Temperatursensor soll sich wiederum davor befinden. Somit wird der Temperatursensor beim Einschalten der Heizwiderstände eine schnelle Reaktion zeigen und das Einschalten des Lüfters wird die Temperatur wieder schnell senken. Die Temperatur kann daher aufgrund der niedrigen Reaktionszeiten für den Benutzer sichtbar geregelt werden. Tabelle 5 zeigt alle elektronischen Bauteile, die für das Modell verwendetet wurden.

Elektronik		
Funktion	**Anzahl**	**Bauteil**
Beleuchtung	6	Luxeon Star 350mA Lumiled weiß
Notbeleuchtung	5	grüne LED
Projektor	1	Aiptek Miro II 7“ Digitaler Bilderrahmen
Lüftung	1	4.8 cm 12V Titan Gehäuse Lüfter
Fenster	1	Blue Bird BMS-371 Mikro Servo
Heizung	2	Heizwiderstand
Temperatursensor	1	Nickel 1000 Sensor

Tabelle 5: Elektronische Bauteile im Modell

4.5.2 Zusatzplatine

Einige elektronische Bauteile im Modell können aus verschiedenen Gründen nicht direkt von der AS gesteuert werden und erfordern eine erweiterte Lösung. Um die Lumileds dimmen zu können, müssen sie mit einem Pulsweiten modulierten (PWM) Signal von ca. 100Hz angesprochen werden. Die AS kann allerdings maximal 10Hz Frequenzen erzeugen. Zusätzlich wäre eine externe Stromspeisung nötig, da die AS nur Steuersignale oder Relays (siehe Kapitel 5.2.2) schalten kann. Auch die Position des Servos wird absolut mit einem PWM-Signal bestimmt.

Es macht daher Sinn, eine zusätzliche Platine zu entwickeln, an die sämtliche Elektronik angeschlossen wird. Die Platine soll für die richtige Stromversorgung und PWM-Signale für die einzelnen Elektronikbauteile sorgen, während die AS die Regelung durch Steuersignale, die an die Platine gesendet werden, übernimmt. Die Platine soll für den Benutzer unsichtbar in den Hohlraum unter der Tribüne eingebraut werden. Die Stromversorgung bezieht sie von der Automationsstation, die einen dafür vorgesehenen AUX-Power Anschluss mit einer Spannung von 13,4V besitzt.

Das Platinen-Layout, das Löten der Bauteile auf die Platine sowie das Programmieren des Mikrocontrollers wird von Sven Krauß Übernommen. Auf der Platine befinden sich 6 Anschlüsse mit 3,5V Spannungsversorgung für Power LEDs, 6 Anschlüsse für normale LEDs sowie 3 Ausgänge für Servos und ein Ausgang für den Ventilator. Gesteuert werden können diese über 8 analoge sowie 8 binäre Eingänge, die frei programmierbar auf die Ausgänge gelegt werden können.

4.5.3 Projektor Simulation

Um einen Projektor zu simulieren soll ein 7“ digitaler Bilderrahmen eingesetzt werden. Anforderungen sind Außenmaße von maximal 15cm x 20cm und, dass das Gerät beim Schließen des Stromkreises automatisch hochfährt und sofort auf die Bilderwiedergabe wechselt. Grund hierfür ist, dass das Gerät über ein Relay gesteuert werden soll, das den Stromkreis öffnet oder schließt.

Ein Test von 9 Geräten im Expert Lörrach zeigte, dass der Apitek Miro II Bilderrahmen der einzige war, der diesen Anforderungen genügen konnte. Alle anderen Geräte waren zu groß, verfügten über einen Soft-Power-On-Button oder gingen nach dem Start in ein Optionsmenü. Auch das Apitek Gerät verfügt zwar über einen Soft-Power-On/Off-Button, wird dieser aber ständig gedrückt gehalten, erfüllt das Gerät die gewünschten Anforderungen.

Abbildung 11 zeigt das Modell mit sämtlicher Elektronik sowie Tapezierarbeiten, die vorgenommen wurden, um dem entworfenen 3D-Modell möglichst gut zu entsprechen und um das Licht diffus abzustrahlen.

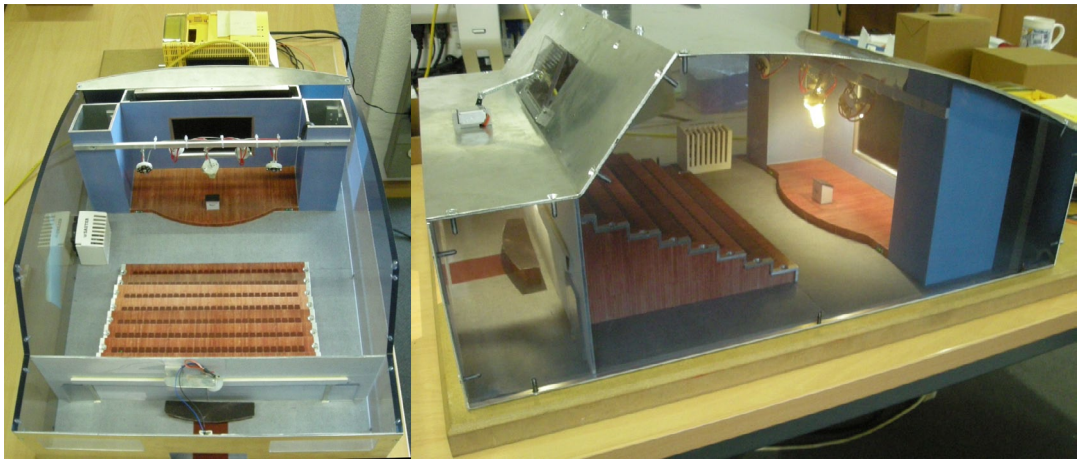


Abbildung 11: Das fertiggestellte Modell

5 AS Regelung

5.1 Allgemein

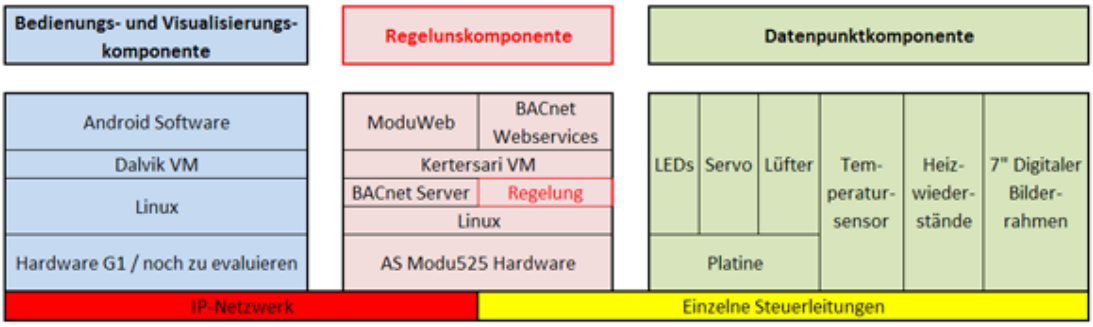


Abbildung 12: Zu definierender Komponente: Regelung

Nachdem der erste Teil des verteilten Systems fertig ist, muss das Modell mit der AS Modu525 verbunden und ein Regelungsplan für diese erstellt werden. Hierfür wird die Sauter Case Engine Software verwendet, die das Erstellen des Regelungsplans mit einer Funktionsbausteinsprache ermöglicht.

Die Software verfügt über verschiedene Arten von Bausteinen die miteinander verbunden werden können. So gibt es Bausteine für Ein- und Ausgänge, wie beispielweise Binary-, Analog-, und Multistate In- und Outputs, die mit den Klemmen der Automationsstation verbunden werden können. Zudem gibt es Bausteine für Signalformatierung, logische Operationen, mathematische Operationen, Zähler, Vergleiche und andere.

Der Aufgebaute Plan wird nach Fertigstellung auf die Station geladen und dort in 100ms-Abständen immer wieder mit den aktuell anliegenden Werten ausgeführt.

5.2 Datenpunktanbindung

5.2.1 Analog

Die Datenpunkte sollen, wie bereits in Kapitel 4.3.1 definiert, mit der Automationsstation verbunden werden. Die analogen Datenpunkte (Licht Bühne, Spot Bühne und Licht Tribüne) werden alle mit sogenannten 0-10V Ausgängen verbunden. Diese legen eine Spannung von 0-10V auf die Klemme, je nachdem welcher Float-Wert am damit verbundenen Analog-Output Baustein anliegt. Um

den Wert richtig skalieren zu können, muss in dem Baustein ein minimaler und ein maximaler Wert angegeben werden. Die 0-10V Werte werden mit den analogen Eingängen der Zusatzplatine verbunden und dort, je nach Datenpunkt, richtig auf die Ausgänge geschaltet.

5.2.2 Binär

Die Binary-Output-Bausteine können unterschiedliche Arten von Klemmen ansteuern: zum einen die sogenannten Open Collector (OC) Ausgänge, die eine logische 1 oder 0 auf die Leitung geben, und zum anderen Relays, bei denen es sich um Schalter handelt. Ein Relay besitzt zwei Klemmen die bei einer logischen 1 verbunden werden.

Wie bereits erwähnt, soll der digitale Bilderrahmen über ein Relay gesteuert werden. Hierzu wird das Netzteil lediglich in die Steckdose gesteckt, eine Ader aufgetrennt und deren Enden mit den Klemmen eines Relays verbunden.

Die LEDs der Notbeleuchtung werden zentral über ein OC gesteuert. Auch das Licht im Foyer wird mit einem OC an- und ausgeschaltet. Da die AS Modu525 selber jedoch über keine OC-Ausgänge verfügt, muss die Station um das Modul Modu551 erweitert werden, das die gewünschten Ausgänge besitzt. Das Modul wird über eine Steckverbindung mit der AS verbunden und ist dann sofort betriebsbereit. Verbunden werden die Ausgänge mit den binären Eingängen der Zusatzplatine, die entsprechend programmiert wird.

5.2.3 Multistate

Mit den Multistate-Datenpunkten Heizung, Lüftung und Fenster verhält es sich etwas anders. Die Position des Servos wird von 0° - 90° über ein PWM Signal bestimmt. Obwohl die Öffnung des Fensters mit einem Multistate angegeben werden soll, ist es sinnvoll, das Fenster an einen Analog-Output anzuschließen. Die verschiedenen Multistate-Zustände sollen dann das Fenster um vordefinierte Winkel öffnen. Dies gibt einem die Möglichkeit, zu einem späteren Zeitpunkt eine stufenlose Öffnung mit geringem Aufwand umsetzen zu können.

Der Lüfter soll über eine klassische Multistate-Steuerung laufen. Für die 4 Zustände 0,1,2 und 3 werden zwei Bit benötigt, die auf zwei OC-Ausgänge gelegt werden.

Die Heizwiderstände müssen über ein Relay gesteuert werden, da das Netzteil der Zusatzplatine nicht ausreichend Strom für sie liefern kann. Die Heizwiderstände werden also in Reihe geschaltet und wie die Zusatzplatine über den AUX-Power-Anschluss der AS gespeist. Der Stromkreis wird durch ein Relay geführt, mit dem die Heizung dann nach Bedarf an- oder ausgeschaltet werden kann. Da die Heizstufe somit nicht über den Strom geregelt werden kann, muss ein PWM-Signal benutzt werden, das mit dem entsprechenden PWM-Baustein erzeugt wird. Der Multistate-Output regelt somit die verschiedenen Weiten der Pulse in der PWM. Da die Heizwiderstände sehr heiß werden und durchbrennen können, wird eine maximale Belastung von 20% vorgesehen. Die Zykluszeit wird auf 10 Sekunden gesetzt. Die Maximale Pulsbreite beträgt demnach 2 Sekunden. Die Ausgangsbelegungen sämtlicher Datenpunkte können in Tabelle 6 eingesehen werden.

Ausgangsbelegung				
Datenpunkt	**Datenpunkt Typ**	**Ausgangs Typ**	**Modul**	**Klemme**
Licht Bühne	Analog	0V-10V	Modu525	2
Spot Bühne	Analog	0V-10V	Modu525	4
Licht Tribüne	Analog	0V-10V	Modu525	6
Licht Foyer	Binär	Open Collector	Modu551	1
Notbeleuchtung	Binär	Open Collector	Modu551	2
Projektor	Binär	Relay	Modu525	49/50
Lüfter	Multistate	2x Open Collector	Modu551	4/6
Heizung	Multistate	Relay	Modu525	41/42
Fenster	Multistate	0V-10V	Modu525	8

Tabelle 6: Ausgangbelegung der Automationsstation

5.3 Licht & Video Steuerung

5.3.1 Allgemein

Für die Regelung der „Licht & Video“-Datenpunkte, wie sie in Kapitel 4.3.2 genannt werden, ist vor allem die Umsetzung der Szenariensteuerung schwierig. Wie sich herausstellt, sind die Regelungspläne bzw. die Case Engine Bausteine für Szenariensteuerungen nicht vorgesehen, auch wenn dies durchaus ein realistischer Kundenwunsch ist. Es treten bei dem Umsetzen des Plans verschiedene Probleme auf, die hier im Näheren erläutert werden sollen.

Zuerst sollte man sich darüber im Klaren sein, dass jeder „Licht & Video“-Datenpunkt grundsätzlich von zwei Quellen verändert werden kann. Zum einen kann dies über die manuelle Schaltung des Datenpunktes geschehen, zum anderen über das Schalten eines Szenarios, das den Datenpunkt auf einen vordefinierten Wert setzt.

Dieses Schalten eines Datenpunktes muss wiederum unterschiedliche Folgezustände auslösen. Man nehme an, das Szenario befände sich im Reinigungsmodus. Der Projektor wäre in diesem Fall ausgeschaltet. Würde der Benutzer den Projektor nun von Hand einstellen, müsste das „Licht & Video“-Szenario auf „Manuell“ springen, da es sich nicht mehr in dem vordefinierten Reinigungsmodus befände. Würde der Benutzer aus dem Reinigungsmodus jedoch in den Filmvorführungsmodus schalten, so würde auch hier der Projektor angestellt werden, das Szenario müsste aber auf dem Filmvorführungsmodus bleiben und dürfte nicht auf „Manuell“ wechseln. Abbildung 13 soll das Problem visuell darstellen.

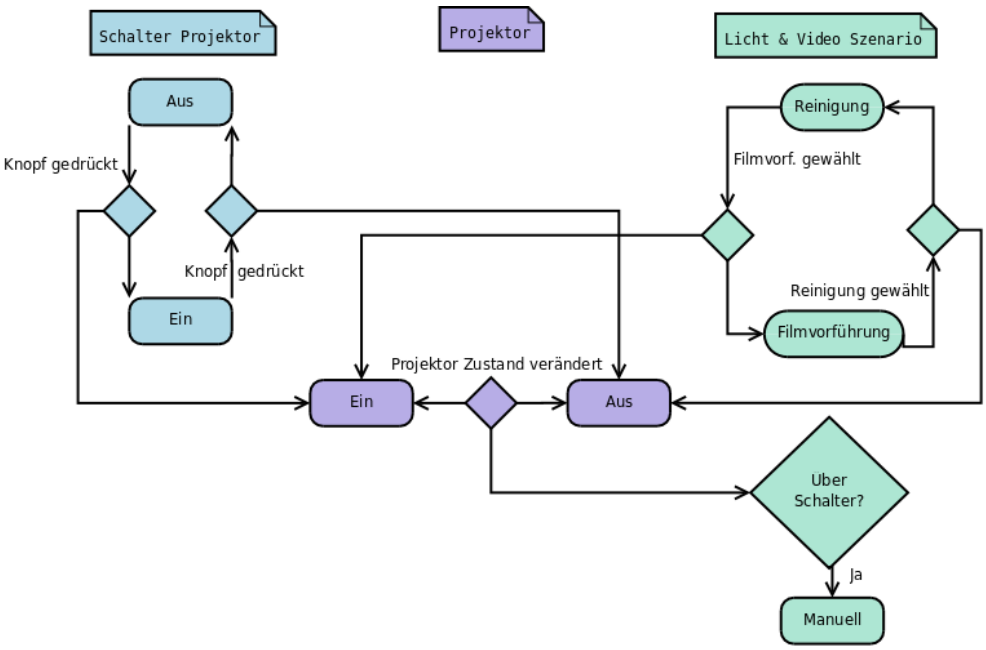


Abbildung 13: Licht & Video Zustände

5.3.2 Feststellen von Werteänderungen

Das Beispiel zeigt, dass bei jeder Werteänderung des Projektors, und somit auch bei jedem anderen „Licht & Video“-Datenpunkt, festgestellt werden muss, ob die Änderung über ein Szenario oder den individuellen Schalter für den Da-

tenpunkt veranlasst wurde. Ansonsten wäre es nicht möglich, das aktuelle „Licht & Video“-Szenario bei manuellen Änderungen auf „Manuell“ zu schalten, bei Änderungen durch ein Szenario aber dieses beizubehalten. Es muss also festgestellt werden können, ob ein Datenpunkt manuell oder über ein Szenario verändert wurde. Dies kann durch eine simple aber effiziente Schaltung realisiert werden, die in Abbildung 14 zu sehen ist.

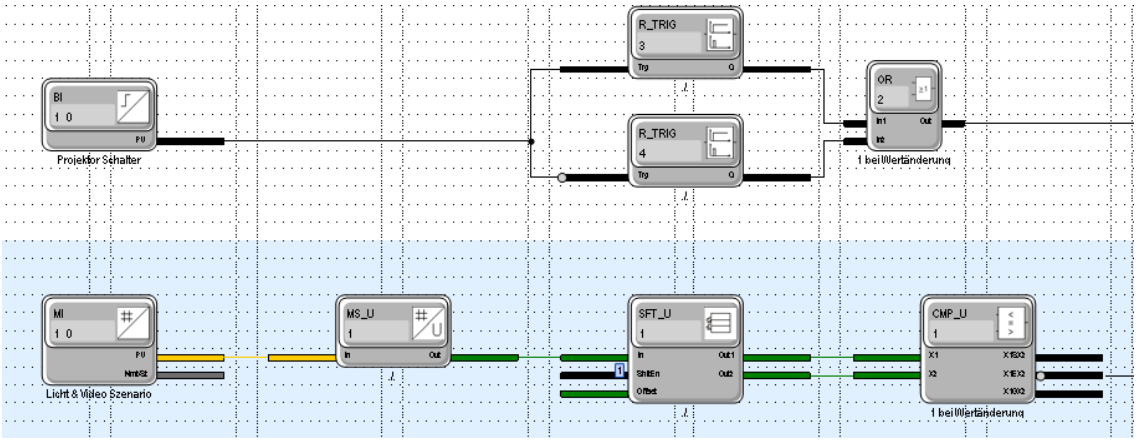


Abbildung 14: Schaltung um Wertänderungen festzustellen

Für den binären Projektor-Schalter werden zwei Rise-Trigger eingesetzt, von denen einer negiert ist. Die Ausgänge werden wieder über einen OR-Baustein verbunden. Sobald der Wert des Projektor-Schalters sich ändert, wird eine Flanke gesetzt.

Der „Licht & Video“-Szenario-Multistate-Input muss zunächst in einen Unsigned Integer Wert konvertiert werden. Dieser wird durch einen Shifter geführt, der auf Out1 den aktuellen Wert und auf Out2 den Wert des letzen Zyklus ausgibt. Dieser Wert wird an einen Vergleicher weitergegeben. Stimmen die zwei Werte nicht überein, hat sich das „Licht & Video“-Szenario verändert und eine positive Flanke wird erzeugt.

Ist der OR-Baustein 1, so wurde der Wert vom Projektor-Schalter gesetzt. Iist der CMP_U Baustein 1, so wurde der Wert durch das „Licht & Video“-Szenario gesetzt. Sind beide 0 hat sich nichts geändert.

Mit dieser Schaltung kann zwar festgestellt werden, durch welche Quelle sich der Projektor-Wert verändert hat, es gibt jedoch noch weitere Probleme, mit dem in Abbildung 13 gezeigten Sachverhalt.

Das erste Problem ist, dass ein Output immer nur genau mit einem anderen Baustein verbunden werden kann. Es ist demnach nicht möglich sowohl den Projektor-Schalter als auch das „Licht & Video“-Szenario mit dem Projektor zu

verbinden. Es muss also schon vor dem Schreiben auf den Projektor eine Wahl getroffen werden.

Aus dem gleichen Grund kann das „Licht & Video“-Szenario nicht gleichzeitig von der zu programmierenden Android-Software und von dem Regelungsplan verändert werden, wie es auf Abbildung 13 dargestellt ist. Die Szenarioänderung auf „Manuell“, die durch den Regelungsplan ausgelöst wird, ist also so nicht möglich.

5.3.3 Einführen von Schalter- und Ausgangsdatenpunkten

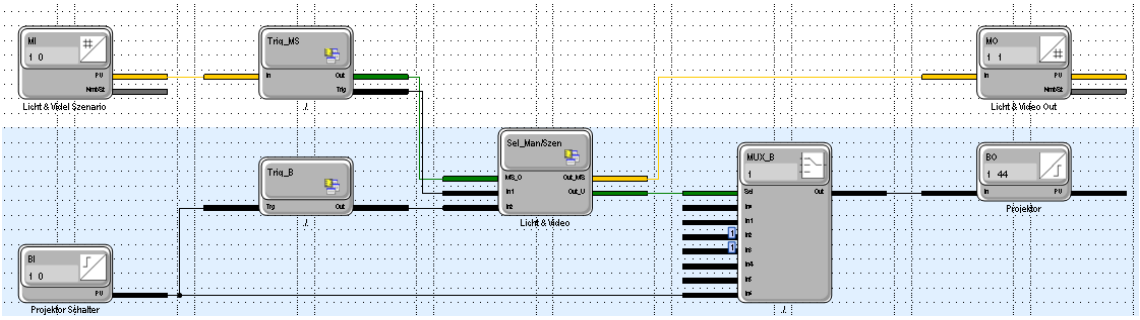


Abbildung 15: Licht & Szenario Schaltung

Die in Abbildung 15 gezeigte Schaltung löst beide oben genannte Probleme. In Trig_MS und Trig_B befindet sich die in Abbildung 14 gezeigte Schaltung. Die Ausgänge werden in den Sel_Man/Szen Baustein geleitet. Diesem wird außerdem der aktuelle „Licht & Video“-Szenario-Wert übergeben.

Der Baustein leitet, je nachdem ob die Wertänderung von dem Szenario oder dem Projektor-Schalter kommt, den aktuellen Szenario-Wert oder die Zahl 6 auf Out_U weiter. Auf den inneren Aufbau des Bausteins soll hier nicht näher eingegangen werden. Er kann in der Anlage C unter /Regelungsplan eingesehen werden.

Der von Sel_Man/Szen auf Out_U gelegte Wert wird auf den Eingang Sel des MUX_B Bausteins gelegt. MUX_B hat auf der linken Seite 7 Eingänge: In0 bis In6. Jeder Eingang steht für eines der 7 „Licht & Video“-Szenarien. Die Szenarien haben dieselbe Reihenfolge wie sie in Tabelle 3Tabelle 1. Der Eingang In6, der für das Szenario „Manuell“ steht, ist mit dem Projektor-Schalter verbunden. Alle anderen Eingänge sind mit vordefinierten Werten belegt. Der MUX_B-Baustein leitet immer den Input an den Projektor-Ausgang weiter, der an Sel anliegt. Aus diesem Grund wird bei einem Trigger des Projektor-Schalters der

Ausgang von Sel_Man/Szen auf 6 gesetzt, da dann direkt der Wert des Projektor-Schalters an den Projektor-Ausgang weitergeleitet wird. Kommt der Trigger von dem „Licht & Video“-Szenario, wird der entsprechende vordefinierte Wert weitergeleitet. Somit umgeht man das Problem, dass mehrere Quellen auf den Projektor schreiben.

Um das zweite Problem zu lösen, muss ein zweiter „Licht & Video“-Baustein eingeführt werden. Ähnlich wie bei dem Projektor, bei dem ein Projektor-Schalter und ein Projektor-Ausgang vorhanden sind, werden nun auch ein Szenario-Schalter und ein Szenario-Ausgang eingeführt. Der Szenario-Ausgang wird auch durch den Sel_Man/Szen Baustein gesetzt und zeigt daher immer das aktuelle Szenario an. In dem umgesetzten Plan befinden sich logischerweise für jeden „Licht & Video“-Datenpunkt ein Trigger-Baustein sowie ein MUX Baustein, der den entsprechenden Wert für das jeweilige Szenario weiterleitet.

5.3.4 Synchronisierung

Ein letztes Problem in diesem Plan muss noch gelöst werden. Wird der Plan wie bisher beschrieben eingesetzt, sind jeweils die Schalter und Ausgänge asynchron. Ist der Projektor manuell eingeschaltet, wird jedoch durch das Reinigungsszenario wieder ausgeschaltet, bleibt der Projektor-Schalter trotzdem noch auf „an“. Will der Benutzer den Projektor nun von Hand wieder einstellen, kann er dies nicht, da der Projektor-Schalter ja bereits auf „an“ steht. Er muss Ihn also erst ausschalten und dann wieder einschalten, was umständlich und verwirrend ist. Es muss also eine Synchronisierung zwischen dem jeweiligen Datenpunkt-Ausgang und dem Datenpunkt-Schalter geben.

Da ein Datenpunkt nur durch eine Quelle beschrieben werden kann und die Datenpunkteschalter durch die Android-Software gesetzt werden sollen, muss auch die Synchronisierung zwischen Ausgang und Schalter über die Software geschehen. Dies ist allerdings mit den gegeben Mitteln nicht möglich, da der Projektor-Schalter bei einer Synchronisierung immer einen Trigger abgeben würde, was zur Folge hätte, dass das Szenario auf „Manuell“ geschaltet werden würde.

Es ist also nötig, dass die Software dem Regelungsplan mitteilen kann, ob ein Wert aktiv vom Benutzer gesetzt wird oder ob der Wert synchronisiert wird, und damit das Szenario nicht auf „Manuell“ geändert werden soll.

Um dies zu bewerkstelligen, muss ein weiterer Datenpunkt eingeführt werden. Es handelt sich um einen binären Input mit dem Namen Sync/Manuell.

Ist dieser Datenpunkt auf 1 gesetzt, wird jede Datenpunktänderung als manuelle Änderung behandelt und das Szenario auf „Manuell“ geschaltet. Ist der Datenpunkt 0, werden Datenpunktänderungen als Wertesynchronisation behandelt und nichts Weiteres unternommen. Der Sync/Manuell-Datenpunkt wird wie in Abbildung 16 zu sehen in die Trigger-Schaltung eingebaut.

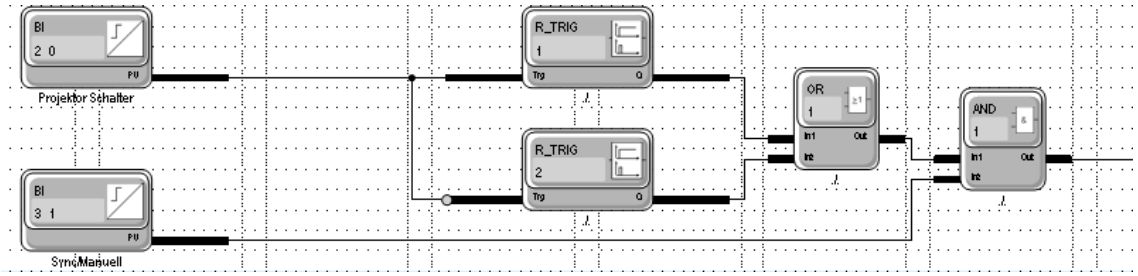


Abbildung 16: Erweiterung um einen Sync-Baustein

Es handelt sich hierbei um eine Erweiterung der in Abbildung 14 gezeigten Schaltung. Der Trigger wird nun nur noch auf 1 gesetzt, wenn es sowohl eine Werteänderung im Projektor-Schalter gab, als auch der Sync/Manuell-Datenpunkt auf 1 ist. Dies setzt natürlich voraus, dass die entsprechende Software vor jeder Wertesynchronisierung den Sync/Manuell-Baustein auf 0 und vor jeder echten Werteänderung den Baustein auf 1 setzt.

5.4 Temperaturregelung

Die Temperaturregelung wird mit einem PID-Baustein geregelt, wobei nur der P-Anteil genutzt wird. Ist die aktuelle Temperatur unter der Soll-Temperatur, werden je nach Temperaturunterschied die Heizwiderstände wenig bis sehr stark geheizt, das Fenster geschlossen und die Lüftung ausgeschaltet. Liegt die Solltemperatur unter der aktuellen Temperatur, wird das Fenster zwischen 0° und 90° geöffnet und die Lüfterstufe auf 0 bis 3 gesetzt.

Alle Klima-Bausteine, die im Zustand „auto“ sind, werden von der Regelung gesteuert. Wenn einer der Bausteine von Hand auf einen anderen Wert gesetzt wird, werden trotzdem alle Bausteine im „auto“-Zustand weiterhin über die Regelung gesteuert. Wird die Solltemperatur verstellt, werden alle Klima-Bausteine mit der gleichen Schaltung wie die „Licht & Video“-Szenarien automatisch auf „auto“ gesetzt.

6 Software Anforderungsanalyse

6.1 Allgemein

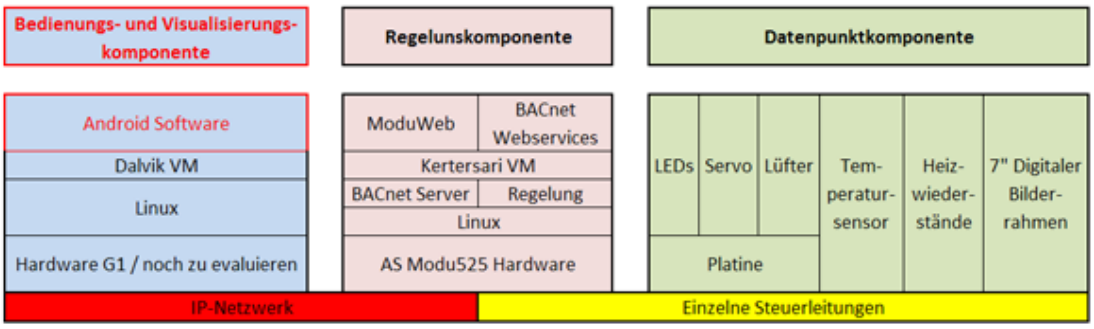


Abbildung 17: Zu definierender Komponente: B&V

Bevor mit dem aktiven Design der Software und des grafischen Interfaces begonnen werden kann, muss zunächst eine ausführliche Anforderungsanalyse durchgeführt werden. Hierbei müssen, neben den in Kapitel 3.4 spezifizierten Anforderungen, vor allem Anforderungen, die durch die Android-Plattform, die zugrunde liegenden Hardware sowie durch das Modell und den Regelungsplan gegeben sind, berücksichtigt werden.

Zunächst soll die Systemumgebung der Software, das Android-Framework, näher betrachtet werden. Der Fakt, dass es sich bei der zu erstellenden Software um eine Embedded Applikation und kein Desktop-Programm handelt, hat trotz der bekannten Java-Umgebung große Auswirkungen auf das Implementieren der Software. So sollen die Folgen der in Kapitel 2.2 vorgestellten Unterschiede von Android zu Sun Java für die zu entwickelnde Applikation untersucht werden.

Neben der Softwareumgebung muss auch die physische Umgebung der Komponente betrachtet werden. Hierzu gehören regelungsplanbedingte Anforderungen und ihre Konsequenzen, sowie die Wahl der Netzwerkprotokolle für den Informationsaustausch zwischen Android und der Automationsstation.

Anschließend wird aufgrund der durchgeführten Analysen und der gegeben Anforderungen Rollen und Akteure definiert, sowie Use-Cases für die Grund-Funktionalitäten der Software erstellt. Abschließend werden die Anforderungen für das grafische User Interface analysiert und ein Konzept dafür entworfen.

6.2 Analyse des Android-Frameworks

Activities und Intents

Für das Design der zu erstellenden Software hat die Activity-Intent-Struktur von Android wenig Auswirkung. Da es sich um eine sehr spezielle, in sich geschlossene Anwendung handelt, macht der Einsatz von Intent-Filtern kaum Sinn. Auch das Verwenden verschiedener Activities für Ansichten ist fraglich, da auf eine für Android untypische Navigation umgesetzt werden muss.

Android Navigation

Android-Applikationen verlassen sich im Normalfall auf die physikalischen „Menu“- und „Zurück“-Tasten der Geräte für die Navigation durch die Activities. Da die zu evaluierende Hardware jedoch ausschließlich über das Touchscreen gesteuert werden soll, muss eine Navigation in die grafische Oberfläche integriert werden, die in Form einer Navigationsleiste auf sämtlichen Ansichten sichtbar ist. Das gleiche Navigationsmenü für jede Activity aufzubauen macht kaum Sinn, daher scheint es in diesem Fall besser zu sein, eine Activity zu benutzen und innerhalb dieser eine feste Navigationsleiste zu haben. Der Rest des Bildes kann mit einer ViewGroup belegt werden, die verschiedene Ansichten zeigen kann.

Look and Feel

Das Erstellen eines eigenen „Look and Feels“ für die Software dürfte mit Hilfe der XML-Layouts kein Problem sein. Android stellt eine große Palette an Widget-Klassen zur Verfügung, die für die meisten grafischen Umsetzungen ausreichen dürften. An einigen Stellen werden die Klassen wahrscheinlich noch angepasst werden müssen.

Netzwerk Protokolle

Der Einsatz von Netzwerk-Protokollen, die nicht nativ von Android unterstützt werden, sollte aufgrund der Standard-Konformität des Android-Frameworks zu Sun kein Problem darstellen, solange entsprechende Java-Bibliotheken existieren. Es sollte jedoch auf die Komplexität der Protokolle geachtet werden, da die Zielplattformen Embedded Systeme mit eingeschränkter Rechenleistung sind.

6.3 Netzwerkschnittstelle

6.3.1 Umgebung

Bevor verschiedene Protokolle evaluiert werden können, wird zunächst die Umgebung, d.h. die zwei Komponenten, die über das Netzwerk verbunden werden, betrachtet.

Auf der einen Seite steht die Android-Software, die auf Java basiert und wahlweise auf einem Embedded-Ethernet-Device oder aber einem Embedded WLAN- / Mobilfunk-Device ausgeführt wird. Es kann also Verbindungen mit geringer und großer Bandbreite, sowie stabiler und instabiler Konnektivität geben. Wichtig ist, dass bei beiden Anwendungsfällen annehmbare Reaktionszeiten der Datenpunkte, wie in Kapitel 3.4.3 spezifiziert, garantiert sind. Mit Java als Grundlage sollte grundsätzlich eine Implementierung fast jedes Protokolls möglich sein.

Auf der anderen Seite steht die über Ethernet verbundene AS Modu525 und der Regelungsplan. Dieser befindet sich in einer C++ Umgebung und publiziert seine Datenpunkte über das Building Automation Control Network Protokoll (BACnet) [13]. Zusätzlich verfügt die AS über eine Java VM, welche über eine Java Native Interface (JNI) Brücke und entsprechenden BACnet-Klassen Zugriff auf sämtliche Datenpunkte der Station erlaubt. Auf dieser Ebene befindet sich ein SOAP[14] Webservice Server, der das Auslesen und Schreiben von einem oder mehreren Datenpunkten erlaubt.

6.3.2 Protokollkriterien

Da beide Seiten über eine Java-Umgebung verfügen, läge das Verwenden des Java Remote Method Invocation Protokolls (RMI) [15] zunächst auf der Hand. Ein zweiter Blick zeigt jedoch, dass das Protokoll von Android nativ nicht unterstützt wird und das aus einem guten Grund. Handheld Devices, für die Android entwickelt wurden, verfügen zumeist, wie bereits erwähnt, über unstabile, langsame Netzwerkverbindungen. Es sind demnach Protokolle erwünscht, die kurz eine Verbindung zum Server aufbauen und diese anschließend wieder fallen lassen. Dies spart auf der einen Seite Ressourcen und auf der anderen Seite ist keine permanente Verbindung für eine solche Kommunikation nötig.

RMI dagegen verlässt sich auf eine initiale TCP-Verbindung, die über die gesamte Applikationslaufzeit aufgebaut bleibt. Für ein fest verkabeltes Netzwerk bedeutet dies eine maximale Performanz, für unstabile mobile Netzwerke ist eine solche Art der Verbindung jedoch denkbar schlecht.

Da die Netzwerkanbindungen der zwei Android-Devices von Grund auf verschieden sind, macht es Sinn die Software mit mehr als nur einem Kommunikationsprotokoll auszustatten. So wäre ein Protokoll, das schnell und stabil mit beliebig viel Traffic im lokalen Ethernet eingesetzt werden kann, für die fest installierten großen Touch-Devices geeignet. Da man sich im lokalen, drahtgebundenen Netz befindet, ist der Sicherheitsaspekt hier zu vernachlässigen.

Zusätzlich soll ein verbindungsloses Protokoll, das wenig Traffic verursacht, bereitgestellt werden. Dieses kann bei externen Zugriffen über Wireless Verbindungen benutzt werden. Hier spielt der Sicherheitsaspekt durchaus eine Rolle, da der entsprechende Server frei aus dem Internet erreichbar sein muss.

6.3.3 Lokales Netzwerkprotokoll

Für das lokale Netzwerkprotokoll macht das Implementieren eines BACnet Clients in der Android-Software aus verschiedenen Gründen Sinn.

- Der BACnet Server ist auf Seiten der Automationsstation direkt in C++ Implementiert und ist daher um einiges performanter als eine Java Implementierung eines anderen Protokolls, die sich zuerst durch die JNI Schicht auf die C++ Ebene herunter arbeiten muss, um auf Datenpunkte zuzugreifen.
- Zusätzlich ist die Automationsstation für die Verwendung von BACnet gebaut worden und damit für das Protokoll optimiert.
- BACnet bietet die Funktionalität mit einem Befehl mehrere Datenpunkt auszulesen oder darauf zu schreiben. Dies stellt beispielsweise beim Abrufen aller Datenpunkte ein Vorteil dar.
- BACnet bietet eine sogenannte Change of Value (COV) Informierung. Auf ein Abonnement hin, werden alle Änderungen eines Datenpunktes dem abonnierenden Client mitgeteilt.
- Ein weiterer Grund ist die Möglichkeit, die lizenzfreie BACnet4Java Bibliothek von Serotonin Software [23] für den BACnet Client in der Android-

Software zu verwenden, welche einen einfachen Zugriff auf BACnet Server erlaubt.

- Hinzu kommt, dass ein natives Protokoll der Automationsstation benutzt wird und somit keine neue Protokollunterstützung implementiert werden muss.

Aufgrund der überzeugenden Argumente, die für BACnet sprechen, werden für das lokale Protokoll keine weiteren Alternativen in Betracht gezogen.

6.3.4 Mobiles, externes Netzwerkprotokoll

Die Wahl des zweiten Protokolls gestaltet sich um einiges schwieriger. Das Protokoll sollte über einen offenen Port, wie beispielweise den HTTP-Port, da sonst bei der Kommunikation über das Internet Probleme mit Firewalls auftreten können. Wie bereits erwähnt, verfügt die Station über SOAP Webservices, die auf HTTP aufsetzen. Diese unterstützen unverschlüsselten sowie über SSL verschlüsselten (HTTPS) Zugriff und stellen einen Teil der BACnet Methoden als Webservices zur Verfügung. Hierunter zählen die Methoden zum Lesen und Schreiben von einzelnen oder mehreren Datenpunktwerten. Eine COV-ähnliche Funktion gibt es allerdings nicht.

Obwohl Webservices grundsätzlich eine typische Lösung für verbindungslosen Informationsaustausch über das Internet auf Embedded Devices darstellen, verhält sich dies bei SOAP Webservices etwas anders. SOAP Webservices sind meist durch das Verwenden von publizierten Webservice Description Language (WSDL) Dateien als Client sehr einfach nutzbar. Es gibt für die meisten Programmiersprachen Clientgeneratoren, die aus einer eingelesenen WSDL-Datei eine fertigen Stub generieren, aus dem die Webservice Methoden lokal im Code aufgerufen werden können.

Dies spart zwar viel Implementierungsaufwand, die generierten Stubs besitzen jedoch immer einen großen Overhead, der durch die komplexe XML-Definition der Services entsteht. Aufgrund des komplexen Aufbaus der SOAP-Nachrichten wird verhältnismäßig viel Rechenleistung für das Zusammensetzten und Auslesen der einzelnen Nachrichten benötigt. Zudem kommt hinzu, dass, je nach Komplexität des Service, die SOAP-Nachrichten mit viel Overhead aufgebläht werden. Dieser wird mit über das Netzwerk geschickt und verursacht zusätzlichen Traffic.

Aus diesen Gründen wird von der Verwendung von SOAP Webservices für Android grundsätzlich abgeraten. Besser hingegen sind RESTful Webservices, siehe [9], Kapitel 6.5. Diese benutzen das HTTP-Protokoll, ohne darauf ein eigenes Protokoll wie SOAP zu definieren. Für jede Ressource gibt es eine eigene URI. Mit HTTP-Befehlen wie GET, POST, SET oder DELTE können die entsprechenden Ressourcen dann ausgelesen, verändert oder gelöscht werden. Hierdurch nimmt die Komplexität und damit auch die Größe der gesendeten Nachrichten stark ab. Dies wiederum bedeutet einen geringeren Generierungs- bzw. Ausleseaufwand für einzelne Nachrichten. Tabelle 7 zeigt einen Vergleich zwischen SOAP und RESTful-Webservies auf.

	SOAP Webservices	**RESTful Webservices**
Verbindungslos	ja	ja
Protokoll	HTTP / SOAP	HTTP
Komplexität	hoch	niedrig
Nachrichtengröße	groß	klein
Implementierungs-aufwand Server	bereits implementiert	groß
Implementierungs-aufwand Client	mittel	gering

Tabelle 7: Vergleich zwischen REST und SOAP Webservices

Trotz den Vorteilen, welche die REST-Webservices aufweisen, fällt die Entscheidung für die Prototypenimplementierung auf das Umsetzen der SOAP-Webservices. Dies soll anhand der folgenden Punkte kurz erläutert werden.

- Die SOAP Webservices sind auf der AS bereits BACnet konform implementiert und erlauben alle benötigten Funktionen für das Auslesen und Schreiben von Datenpunkten, wohingegen für REST Webservices zuerst ein neues Konzept geschaffen werden müsste. Ein solcher Aufwand würde den Rahmen dieser Arbeit sprengen.
- Die SOAP Webservices bauen bereits auf dem Rollensystem der Station auf und werden über Benutzername und Passwort authentifiziert. Im Zusammenhang mit einer HTTPS Verbindung sollte somit ausreichend Sicherheit, auch für eine Publikation im Internet, geboten sein.
- Da die Serviceaufrufe für das Auslesen und das Schreiben eines Datenpunktwertes sehr einfach gehalten sind, können diese ohne Hilfe eines zusätzlichen SOAP-Generators erzeugt und ausgelesen werden. Auch das Erzeugen von Nachrichten, die mehrere Datenpunkte Auslesen oder Schrei-

ben, ist ohne Hilfe eines SOAP Generators möglich. Lediglich für das Auslesen dieser Methoden wird der Einsatz eines SOAP-Interpreters benötigt.

6.4 Analyse des Regelungsplans

Der Regelungsplan bzw. die Datenpunkte im Regelungsplan sind die Schnittstelle zwischen Android-Applikation und den darzustellenden physischen Datenpunkten. An dieser Stelle besteht eine enge Kopplung, deren Management und Implementierung einen zentralen Punkt in der Software darstellt.

Sehr entscheidend für das Umsetzten der Software ist, dass alle anzusteuernden, physikalischen Datenpunkte in der Regelung zwei Datenpunkte, nämlich einen Schalter-Datenpunkt und einen Ausgangs-Datenpunkt, besitzen. Ein GUI-Element, das einen Datenpunkt repräsentiert, muss also immer auf den Schalter-Datenpunkt schreiben, dem Benutzer aber den Ausgangs-Datenpunkt als aktuellen Zustand anzeigen. Dies wird in Abbildung 18 veranschaulicht. Wie in Kapitel 5.3.4 beschrieben, müssen diese zwei Datenpunkte zusätzlich von der Software synchron gehalten werden.

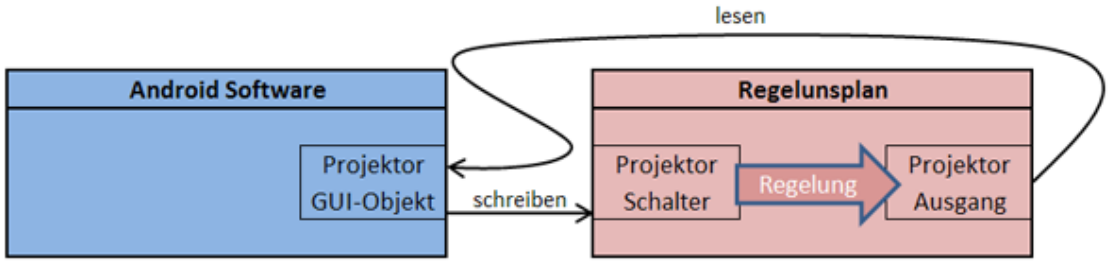


Abbildung 18: Lesen und Schreiben von Datenpunkten

Sowohl über BACnet als auch über WebServices werden Datenpunkte immer auf die gleiche Weise angesprochen, ohne den Datenpunkt-Typ zu beachten. Da die Protokolle hier konsistent sind, wird es für die Applikation Sinn machen, die Datenpunkte ebenfalls auf diese Weise zu hinterlegen.

6.5 Rollen und Akteure

6.5.1 Zielgruppen

Unter Rollen werden in diesem Fall die Benutzergruppen verstanden, die mit der Software interagieren sollen. In unserem Anwendungsfall gibt es zwei

Zielgruppen. Die erste ist das Personal des Gebäudes. Hierunter zählen Menschen wie die Empfangsdame, der Hausmeister oder die Putzfrau. Diese Personen haben regelmäßigen Umgang mit dem System und können darin eingelernt werden.

Die zweite Zielgruppe sind Redner. Für sie ist das System völlig fremd und trotzdem müssen sie innerhalb kürzester Zeit dazu im Stande sein Licht, Temperatur und Projektor nach Ihrem Wunsch einstellen zu können.

Diese Tatsache zeigt deutlich, dass die Software selbst für absolut unerfahrene Benutzer, aus jeder beliebigen gesellschaftlichen Gruppe mit der Software ohne Probleme umgehen können müssen.

6.5.2 Akteure

Zu den Akteuren des Systems gehört, wie bereits definiert, der Benutzer. Zusätzlich kommen die Komponenten des verteilten Automationssystems dazu, wie sie in Abbildung 17 zu sehen sind.

Für die Use-Cases wird die Datenpunkte Komponente des Systems jedoch ausgenommen, da an dieser Stelle fast nur noch Hardware geschaltet wird, die die Befehle der AS Modu525 direkt umsetzt. Akteure für die Use-Cases sind also der Benutzer, die Bedien- und Visualisierungskomponente sowie die Regelunskomponente, die in folgender Beziehung zueinander stehen:

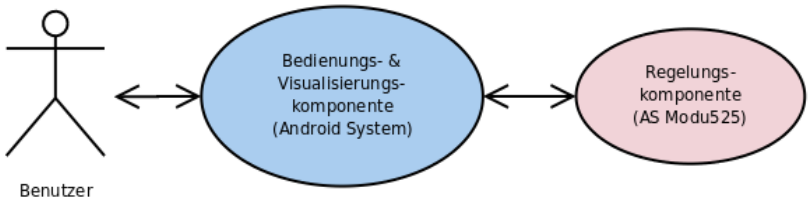


Abbildung 19: Use-Case Akteure

6.6 Use-Cases

Es werden lediglich die Use-Cases mit der Andoird Software als Hauptakteur behandelt, da die Use-Cases mit der Automationsstation als Hauptakteur inhaltlich bereits in Kapitel 5 behandelt wurden. Benutzer-Use-Cases können ohne grundlegende Design-Entscheidungen über Navigation, Ansichten und Darstellungen von Datenpunkten schwer definiert werden.

Use-Case Nummer: Bezeichnung	**ASW1: Auslesen der Datenpunkte und Szenarien**
Ziel:	Die Android SW ruft alle Ausgangs-Datenpunkte von der AS ab und stellt diese korrekt dar.
Hauptakteur:	Android SW
Nebenakteure:	Benutzer, AS Modu525
Vorbedingung:	Dem Benutzer steht eine Android-Plattform zur Verfügung, die sich mit der AS Modu525 verbinden kann.
Auslöser:	Der Benutzer startet die Android-Software und navigiert zu der/ den GUI Darstellung/en, die die Datenpunktwerte zeigt/ zeigen.
Garantie falls erfolg-reich:	Alle Datenpunkt-GUI-Elemente sind auf den entsprechenden, aktuellen Datenpunktwert gesetzt.
Garantie falls Fehl-schlag:	Fehlermeldung, die auf die Ursache des Fehlers hinweist. Das GUI wird normal angezeigt. Die Datenpunkt-GUI-Elemente zeigen keinen Wert bzw. einen Standardwert an.
Hauptszenario:	1. Die Android SW startet korrekt und initialisiert sich, das GUI wird aufgebaut. 2. Die Android SW versucht eine Verbindung zur AS aufzubau-en und ruft alle Ausgangs-Datenpunktwerte ab. 3. Die Android SW setzt die GUI-Elemente der Datenpunkte auf die abgeholten Werte und synchronisiert die Schalter-Datenpunkte (Siehe ASW3)

Tabelle 8: Use-Case 1, Auslesen von Datenpunkten und Szenarien

Use-Case Nummer: Bezeichnung:	**ASW2: Ändern eines Datenpunktes**
Ziel:	Der Benutzer verändert den Wert eines Datenpunktes
Hauptakteur:	Android-Software
Nebenakteure:	Benutzer, AS Modu525
Vorbedingung:	Die Android-Plattform hat die Verbindung zur Automationsstation aufgebaut
Auslöser:	Der Benutzer hat das GUI-Element für das Ändern des Daten-punktes betätigt
Garantie falls erfolg-reich:	Das GUI Element zeigt den neuen Wert/ Zustand an, der entspre-chende physikalische Datenpunkt übernimmt den Wert
Garantie falls Fehl-schlag:	1. Fehlermeldung, die auf die Ursache des Fehlers hinweist. Das GUI-Element geht in den alten Zustand zurück oder bleibt dort. Der physikalische Datenpunkt verändert sich nicht.
Hauptszenario:	2. Die Android SW fordert die AS über das Netzwerk auf, den Sync/Manuell Datenpunkt auf 1 zu setzten. 3. Die Android SW sendet den zu ändernden Schalter-Datenpunkt und den neuen Wert über das Netzwerk an die AS Modu525 4. Die AS bestätigt die Werteänderung 5. Die Android SW ruft den Wert des entsprechenden Aus-gangs-Datenpunktes so lange ab, bis dieser den gesetzten Wert hat bzw. wartet sie bei einem "push"-fähigen Protokoll, bis der entsprechende Ausgangs-Datenpunkt die Wertände-rung meldet. 6. Die Android SW ändert das GUI-Element des Datenpunktes sodass dieses den neuen Datenpunkt-Wert darstellt.

	7. Die Android SW ruft alle Datenpunktwerte von der Station ab und aktualisiert falls nötig die entsprechenden GUI Elemente und synchronisiert die Schalter-Datenpunkte (siehe ASW3)

Tabelle 9: Use-Case 2, Ändern eines Datenpunktes

Use-Case Nummer: Bezeichnung:	**ASW3: Synchronisierung zwischen Schalter- und Ausgangs-Datenpunkt**
Ziel:	Der Schalter-Datenpunkt hat den gleichen Wert wie der zuvor geänderte Ausgangs-Datenpunkt
Hauptakteur:	Android SW
Nebenakteure:	AS Modu525
Vorbedingung:	Der Ausgangs-Datenpunkt wurde durch ein Szenario auf einen Zustand geändert, der von dem Schalterdatenpunkt abweicht.
Auslöser:	Wechseln eines Wertes durch ein Szenario.
Garantie falls erfolg-reich:	Schalter- und Ausgangs-Datenpunkt haben den gleichen Wert
Garantie falls Fehl-schlag:	Fehlermeldung mit Hinweis auf die Ursache
Hauptszenario:	1. Die Android SW fordert die AS über das Netzwerk auf, den Sync/Manuell Datenpunkt auf 0 zu setzten. 2. Die Android SW sendet den zu ändernden Schalter-Datenpunkt und den neuen Wert über das Netzwerk an die AS Modu525 3. Die AS bestätigt die Werteänderung

Tabelle 10: Use-Case 3, Synchronisierung zwischen Datenpunkten

Use-Case 1 und 2 beschreiben die Hauptaufgabe der Software, das Visualisieren und das Bedienen von Datenpunkten. Die Benutzerseite dieses Szenarios und wie diese ausgelöst werden können, soll im Screen Design definiert werden.

6.7 GUI-Anforderungen

Neben den Funktionalitäten der Software spielt die Navigation, Präsentation und Bedienung der zu regelnden Datenpunkte eine zentrale Rolle. Aus den vorherigen Kapiteln heraus definieren sich für das Screendesign verschiedene Anforderungen, die hier unterteilt in Kategorien nochmals zusammengefasst werden sollen.

Visualisierung und Bedienung:

- Darstellungen der Räume Auditorium und Foyer.
- Darstellung der aktuellen Szenarien sowie Bedienelement für eine Anpassung

- Darstellung aktueller Werte von Datenpunkten sowie Bedienmöglichkeiten diese Werte zu verändern
- „Wiedererkennungseffekt“: Der Benutzer soll anhand seiner Position im Raum Datenpunkte um herum sich erkennen.

Navigation

- Übersichtliche Navigation durch die Software: Mit möglichst wenig „Klicks“ zum Ziel
- Der Benutzer soll sich schnell in der Darstellung zurechtfinden können und gezielt Datenpunkte finden, einsehen und verändern können.
- Schnelles Finden der gesuchten Funktionalitäten.

Feedback

- Klare Kommunikation von Fehlern mit Hinweisen auf deren Ursache,
- keine Überladung von Fehlermeldungen

Einstellungen

- Wahl zwischen verschiedenen Protokollen
- Angabe der anzusprechenden Automationsstation
- Evtl. Angabe von Sicherheitsinformationen wie Benutzername und Passwort

Hardwarebedingt:

- Gut handhabbare, schöne UI-Darstellungen sowohl auf einem 420x380px als auch auf einem 800x600px Device.
- Navigation allein über das Touch

6.8 GUI Konzept

6.8.1 Ansichten

Faktisch müssen 10 Datenpunkte und 3 Szenarien dargestellt werden. Dies ist eine Menge, die auf dem G1 nicht übersichtlich auf einer Ansicht dargestellt werden kann. Es wird also eine Aufteilung der Datenpunkte benötigt.

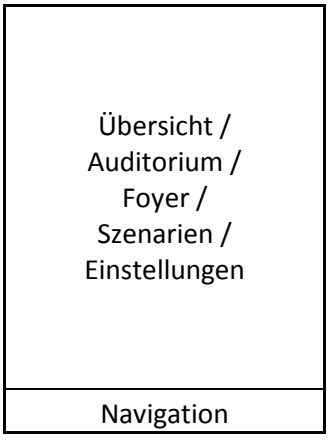
Diese kann entweder nach Kategorien, wie beispielweise „Licht & Video“ und Temperatur, oder aber nach Lokation eingeteilt werden. Da sich die betreffenden Personen in den meisten Fällen im Gebäude befinden werden und ihr

Fokus auf dem Raum, in dem sie sich befinden, liegen wird, macht es Sinn die Datenpunkte der Lokalität nach anzuordnen. Da die Szenarien Raumübergreifend sind, wird für diese mindestens eine weitere Ansicht benötigt. Die Visualisierung der Einstellungen soll ebenfalls in eine getrennte Ansicht ausgelagert werden. Zusätzlich ist eine Übersicht denkbar, auf der alle wichtigen Werte zu sehen sind.

6.8.2 Navigation

Da sich die Navigation innerhalb der grafischen Oberfläche befindet, muss auch hierfür eine geeignete Position für die Navigationstasten gefunden werden. Auch wenn dies für Desktop Programme untypisch ist, bietet sich für Handheld Devices hierfür die untere Kante des Displays an. Grund dafür ist, dass das Gerät oft in einer Hand gehalten und mit dem Daumen gesteuert wird. Die zweite Hand bleibt für andere Dinge frei.

Um dieser Art der Navigation entgegenzukommen, ist die untere Kante zumeist am besten geeignet, da es für den Benutzer je nach Hand-Held-Größe oft schwierig ist an die obere Kante zu kommen. Da die Navigationstasten eine der häufigsten benutzen Tasten sind, sollten diese entsprechend gut erreichbar sein.



6.8.3 Anwendungsgebiet

Bevor das GUI designt wird, muss zuerst das Anwendungsgebiet der Software genau definiert sein. Für dieses Szenario sind die Raumanzahl und das Anwendungsgebiet klar definiert. Das User Interface soll jedoch aus generi-

schen Teilen bestehen, die genauso für andere Gebäude und Räume verwendet werden können. Es stellt sich also die Frage, inwiefern andere Szenarien von dem gebenden Modell abweichen können und an welcher Stelle man auf Gemeinsamkeiten aufbauen kann.

Zimmeranzahl

Wie in den Anforderungen definiert, geht es in der Software darum, beliebigen Benutzern die Möglichkeit zu bieten auf eine direkte Weise mit der Gebäudeautomation zu interagieren, um Licht und/ oder Temperatur nach ihrer Zufriedenheit anzupassen. Es wird dem Benutzer also eine Teilansicht aus dem ganzen System gezeigt, die ihm die für sich relevanten Datenpunkte in einer intuitiven Weise zugänglich macht.

Eine solche Art der Benutzerinteraktion ist im Normalfall auf eine sehr limitierte Anzahl an Zimmern beschränkt. Nur die Datenpunkte, die das Befinden des Benutzers unmittelbar beeinflussen, sollen gezeigt und steuerbar gemacht werden.

Sollte eine entsprechende Software beispielweise in einem Hotel eingesetzt werden, würde jedes Zimmer bzw. jede Suite über ein eigenes Bedienteil verfügen. Dieses Beispiel kann ebenso auf Büros, Klassenzimmer und beliebig andere Anwendungsfelder übertragen werden. Es macht daher Sinn von einer limitierten Zimmeranzahl für die Navigation auszugehen.

Szenarien Ansichten

Eine ähnliche Frage stellt sich bei den Szenarien. Ist es sinnvoll für jedes Zimmer eine eigene Szenarien-Ansicht zu erstellen oder macht es mehr Sinn eine zentrale Szenarien-Ansicht einzusetzen?

Zwei Gründe sprechen dafür, eine zentrale Szenarien-Ansicht zu favorisieren. Da es, wie bereits erwähnt, um die Steuerung der direkten Umgebung des Benutzers geht, werden Szenarien in den meisten Fällen Datenpunkte aus mehr als einem Zimmer steuern. Zusätzlich wird es, wie auch in dem hier realisierten Modell, immer globale Szenarien geben, die beispielweise die Lichter in allen Zimmern abstellen. Eine zentrale Szenarien-Ansicht wird zusätzlich für weniger Ansichten insgesamt sorgen und damit auch dem Benutzer eine einfachere Navigation erlauben.

7 Software Design und Umsetzung

7.1 Allgemein

Das Softwaredesign teilt sich in zwei Bereiche ein. Zum einen das GUI-Layout, welches das Designen der einzelnen Ansichten und Datenpunktvisualisierungen sowie das Look and Feel der Applikation beinhaltet. Der zweite Teil besteht aus dem Design der Software selbst, die sich wiederum in mehrere Komponenten unterteilt.

7.2 GUI Layout

7.2.1 Look and Feel

Da das Look and Feel der Standard Android Widgets zumeist mit seinem grau-schwarzen Stil etwas starr und nicht sehr innovativ wirkt, soll für die Grafische Oberfläche ein neues Look and Feel geschaffen werden. Hierbei sollen sich die Farben an einem Design mit dunkeln Buttons und hellen Schriftzügen orientieren, ähnlich wie es bei Windows Vista, Windows Mobile 6, KDE4.2 oder bei Gira zu finden ist. Grund hierfür ist, dass das Design momentan offensichtlich sehr beliebt ist und dem Benutzer außerdem vertraut erscheinen und damit seinen Einstieg in die Applikation erleichtern wird.

Abbildung 20: Win 6, KDE 4.2, Gira, Win Vista Bedienintefaces

7.2.2 Navigation

Um dem Benutzer eine einfache Handhabung mit der Software zu garantieren ist es wichtig, eine in sich stimmige, einfache Navigation bereitzustellen. Hierfür muss ein Konzept für das Navigieren zwischen den in Kapitel 6.8.1 definierten Ansichten geschaffen werden. Bei den vorhandenen fünf Ansichten wäre es denkbar, jede Ansicht einfach als direkten Link in die Navigationsleiste zu nehmen. Da aber davon ausgegangen werden muss, dass die Software andere Gebäude (mit mehr wie zwei Räumen) visualisieren muss, ist dieses Konzept nicht ausreichend skalierbar.

Es muss daher eine Kategorisierung stattfinden, um die Anzahl der Buttons in der Navigationsleiste zu limitieren und eine Skalierbarkeit zu ermöglichen. Hierfür bieten sich die Räume an. Es wird daher auf eine Subnavigationsebene für die Räume zurückgegriffen, die zur Folge hat, dass in der Metanavigation die drei Elemente Räume, Szenarien und Einstellungen übrig bleiben, die übersichtlich dargestellt werden können. Bei einem Klick auf den Räume-Button wird dem Benutzer über eine neue Navigationsebene die Auswahl zwischen den vorhandenen Räumen gegeben.

Da es für Benutzer, welche die zu steuernden Räumlichkeiten das erste Mal sehen, oft nicht leicht ist, Räume nur über den Namen zu identifizieren, wird auf der Raumnavigationsebene eine visuelle Darstellung der Räume realisiert. Hierfür wird eine Ansicht des gesamten Gebäudes gewählt, in welcher der Benutzer die einzelnen, beschrifteten Räume direkt wählen kann.

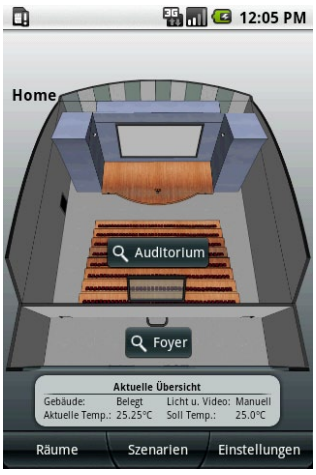


Abbildung 21: Navigationsleiste mit Raumwahl

7.2.3 Symbolsprache

Da das Display zu klein ist, um die Bezeichnung jedes Datenpunktes anzuzeigen und um gewünschte Datenpunkte auch rein visuell erkennen zu können, werden für die verschiedenen Datenpunkttypen sprechende Symbole benutzt. In diesem Fall reicht es nicht, die Datenpunkte in die drei Hauptkategorien Binär, Multistate und Analog zu unterteilen. Viel mehr müssen die verschiedene gesteuerte Elektronik bzw. die Geräte visualisiert werden. Für die im Modell vorkommenden Datenpunkte wurde die in Tabelle 11 gezeigte Symbolsprache eingeführt.

Funktion	Symbol
Licht	
Spotlight	
Notbeleuchtung	
Projektor	
Temperatur	
Heizung	
Lüftung	
Fenster	
Raum ansehen	
Gebäude	

Tabelle 11: Verwendete Symbole

7.2.4 Räume

Die jeweiligen Raumansichten folgen dem Navigationsstiel der Raumauswahl und stellen den jeweils gewählten Raum auf voller Bildgröße dar. Auf diesem Bild sind an den physikalischen Stellen der Datenpunkte jeweils GUI-Objekte platziert, die den aktuellen Wert des Datenpunktes anzeigen und die Möglichkeit bieten diesen zu verändern.

Die Design-Strategie basiert, wie die der Navigation, darauf den Benutzer so schnell wie möglich mit dem System vertraut zu machen. Mithilfe von Raumgrafiken wird es dem Benutzer erleichtert sich zurechtzufinden und gesuchte

Datenpunkte zu finden, da er stets den Vergleich zu seiner realen Umgebung herstellen und sich daran orientieren kann.

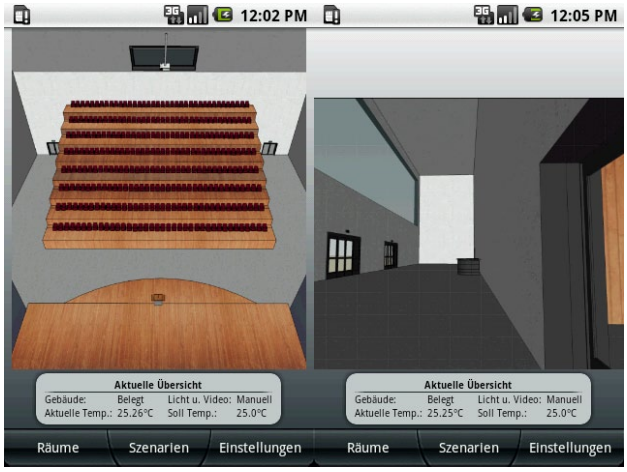


Abbildung 22: Auditorium- und Foyer-Ansicht

Um dem Benutzer einen höheren Wiedererkennungsfaktor zu geben, wurden 3D-Ansichten der Räume gewählt, die mit den echten Texturen belegt wurden. Während die Raum-Navigation eine Übersicht über das ganze Gebäude gibt, zeigen die Auditorium- und Foyer-Ansichten lediglich das gewählte Zimmer.

Für das Auditorium wurde ein Blick von einem Standpunkt hinter der Bühne gewählt, um für den Präsentierenden den Wiedererkennungsfaktor so hoch wie möglich zu gestalten. Da das Foyer sehr schmal und lang ist, war es hier schwierig einen geeigneten Sichtwinkel zu finden, der den Bildschirm gut füllt. Der erstellte Blick erlaubt es alle wichtigen Punkte im Bild zu sehen.

Um dem Benutzer noch mehr Übersicht in der Applikation zu schaffen und unnötiges Hin- und Hernavigieren zu vermeiden, wird in den Raumansichten unten eine kleine Tabelle eingeblendet, welche die wichtigsten globalen Datenpunkte anzeigt. Für die umzusetzende Software werden dort das aktuelle Gebäude- und „Licht & Video“-Szenario sowie die aktuelle Temperatur und die Soll-Temperatur angezeigt. Dies erlaubt es dem Benutzer stets mit einem Blick die wichtigsten Daten einsehen zu können, ohne ständig zwischen Szenario und Raumansichten hin- und herwechseln zu müssen.

7.2.5 Szenarien

Um die Fähigkeiten des Touchscreens und des Android Frameworks zu zeigen, sollen die Szenarien nicht als herkömmliche Ansicht sondern als

hereinziehbare, halbtransparente Ansicht realisiert werden. Außer den genannten Gründen gibt dies dem Benutzer zusätzlich die Möglichkeit, in einer Raum-Ansicht kurz die Szenarien in das Bild zu ziehen, diese zu verändern und wieder aus dem Bildschirm zu ziehen. Er befindet sich danach immer noch in der zuvor gewählten Raum-Ansicht und muss diese nicht über die Raumnavigation neu öffnen.

Um dies zu realisieren, löst sich der Szenario-Navigations-Button aus der Navigationsleiste und fährt durch eine Animation nach oben, die Szenarien-Ansicht hinter sich herziehend. Oben angekommen bleibt er stehen und ermöglicht es dem Benutzer mit der Ansicht zu interagieren. Aus diesem Grund wurde dem Szenario-Button eine abgerundete Form gegeben, die sich passgenau in das Navigationsmenü einbettet und ein Gefühl von Leichtigkeit beim Öffnen und Schließen vermittelt.

Alle Szenarien werden in der hineingezogenen Ansicht in einer Tabelle dargestellt. Dies erlaubt es, die verschiedenen Szenario-Ebenen über Einrückung verständlicher zu machen und würde auch eine Kategorisierung von Szenarien erlauben. Die Tabelle kann nach Bedarf eine beliebige Länge annehmen.

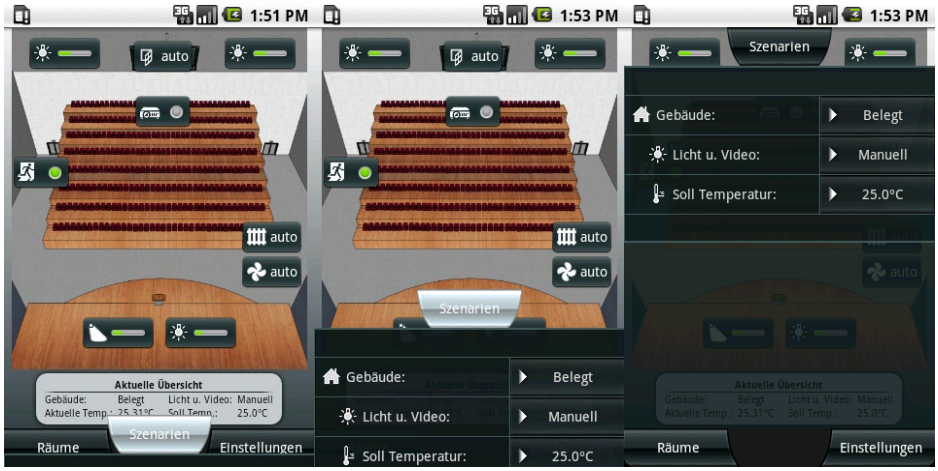


Abbildung 23: Szenarien Ansicht

7.2.6 Einstellungen

Da die Kommunikation mit der AS über verschiedene Protokolle und auch verschiedene Datenverbindungen stattfinden kann, muss dem Benutzer die Möglichkeit gegeben werden, verschiedene Netzwerkparameter einstellen zu können. So wird dem Benutzer in der Einstellungs-Ansicht die Möglichkeit gegeben, zwischen den Protokollen BACnet und Webservices zu wechseln. Zu-

sätzlich kann die IP der Station angegeben werden, da sich diese ändern kann, je nach dem ob lokal oder aus dem Internet auf die Station zugegriffen wird. Benutzername und Passwort, die für WebServices benötigt werden, werden ebenfalls einstellbar gemacht. Zuletzt wird dem Benutzer noch die Möglichkeit gegeben, ein ständiges Pollen der Datenpunktwerde zu aktivieren oder deaktivieren.

Für die Visualisierung dieser Einstellungen wird die SettingsActivity aus dem Android Framework benutzt werden, die es mit wenig Implementierungsaufwand erlaubt, Einstellungen zu visualisieren sowie der restlichen Applikation Zugriffe darauf zu ermöglichen. Die Visualisierung kann in Abbildung 24 gesehen werden.

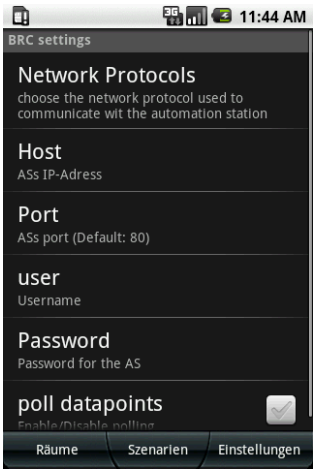


Abbildung 24: Einstellung Ansicht

7.2.7 Datenpunkt Visualisierung

Die Datenpunkte werden alle als Buttons dargestellt, die den aktuellen Status des Datenpunktes anzeigen und mit denen der Benutzer auf verschiedene Weise interagieren kann. Grundsätzlich gibt es drei Sorten von Datenpunkten: Analoge, Binäre und Multistates.

Binäre Datenpunkte können über einen Togglebutton visualisiert werden. Durch das Drücken des Buttons wird der Zustand an/aus gewechselt. Visualisiert wird dies durch ein aufleuchtendes bzw. dunkles Statuslicht.

Analoge und Multistate-Datenpunkte benötigen eine etwas komplexere Steuerung. Dargestellt werden sie zunächst auch als Buttons. Bei Multistate-Datenpunkten wird der aktuelle Zustand auf dem Button angezeigt, bei Analogen Buttons wird der aktuelle Wert durch eine grafische Leiste visualisiert.

Abbildung 25: Binary-, Multistate- und Analog-Button

Bei einem Drücken des entsprechenden Buttons öffnet sich ein Dialogfenster, in dem die Gewünschte Einstellung vorgenommen werden kann. Für analoge Datenpunkte wie z.B. die Lichtintensität wird eine Ziehleiste implementiert, die links bei 0% anfängt und rechts bei 100% endet. Um dem Benutzer ein schnelles An- und Ausschalten zu gewähren, werden zusätzlich noch ein „An“- und ein „Ab“-Button mit in den Dialog genommen, die den Wert sofort auf 0% bzw. 100% setzen.

Da ein solcher Dialog aber nicht für jeden analogen Datenpunkt ideal ist, wird noch ein zweiter Dialog entworfen, der für Analoge Datenpunkt mit stark eingeschränkten Wertebereichen geschaffen ist. Ein gutes Beispiel hierfür ist die Soll-Temperatur. Der Wertebereich erstreckt sich hier von 15°C – 35°C. Von der Klimaanlage im Auto oder Haus ist man es hier gewohnt den Wert über „+“- und „-“-Knöpfe zu regeln, was mit dem entsprechenden Dialog auch den Benutzern dieser Software ermöglicht wird.

Der Dialog der Multistate Datenpunkte listet alle Zustände des Datenpunktes auf und markiert den momentan aktiven Zustand. Das Look and Feel orientiert sich an „Radio“-Buttons von Hifi-Anlagen oder DVD-Playern. Diese besitzen oft bei der Auswahl zwischen CD / DVD / AUX und Tape häufig Knöpfe mit eingelassenen Status-LEDs, die bei Aktivierung der entsprechenden Funktion leuchten. Die Szenarienauswahl wird ebenfalls mit diesem Dialog dargestellt.

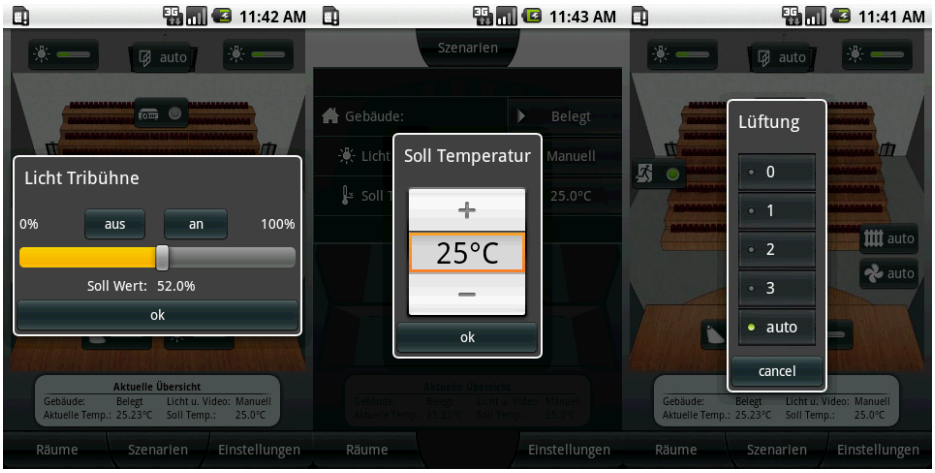


Abbildung 26: für Analog und Multistate Datenpunkte

7.2.8 Darstellung in 800x600 Auflösung

Sämtliche GUI-Darstellungen wurden bis jetzt mit der Android Standard Auflösung 320x480 gezeigt. Die Softwareanforderungen verlangen jedoch, dass eine Darstellung auf einem 800x600 Bildschirm ebenfalls möglich ist.

Das Android Framework bietet die Möglichkeit verschiedene Layout-Files für eine Hochkant und Breitbild-Darstellungen zu definieren. Einzige Voraussetzung ist, dass alle Android IDs in beiden Layout Dateien vorhanden sind. Da das große Device in der Breitbild-Darstellung genutzt wird, werden angepasste Breitbild-Layouts erstellt, um den Bildschirm optimal zu nutzen. Angepasst wurden für die Breitbilddarstellung folgende Dinge:

- Hintergrundbilder für das Auditorium und das Foyer
- Anpassen der Positionen der Datenpunkte in den Bildern
- Die Werte der Übersichtstabelle wurden alle nebeneinander gelegt.

Neben diesen offensichtlichen Änderungen wurde beim Implementieren darauf geachtet, dass alle anderen verwendeten Layouts sich richtig auf die neue Bildschirmgrößen skalieren.

7.3 Software Architektur

7.3.1 Übersicht

Die Software wird als Model-View-Controller-Architektur aufgebaut. Die Datenpunkte bilden die klassische Model-Komponente. Die entsprechenden Visualisierungen der Datenpunkte sowie die restliche grafische Oberfläche bilden die View-Komponente. Für das Aufbauen der Verbindung zur Automationsstation sowie das Lesen und Schreiben der Werte ist der Controller verantwortlich. Da die Aufgaben des Controllers hauptsächlich in der Kommunikation zwischen der Applikation und der AS liegen, wird das Paket Communication genannt.

Abbildung 27 zeigt die einzelnen Komponenten der Software in einer Architektur-Ansicht. Die folgenden Kapitel sollen die einzelnen Komponenten näher erklären. Nähere Informationen über die Software-Architektur sowie die verwendeten Design-Patterns können in [16] und [17] gefunden werden.

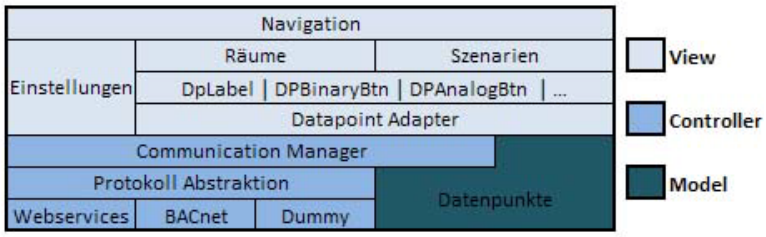


Abbildung 27: Architektonische Ansicht der Software

7.3.2 Model

Wie bereits erwähnt bildet der Datenpunkt an sich das Modell und damit die Daten-Basis der Software. Es wird auf der Klassenhierarchie nicht zwischen Binary, Analog oder Multistate unterschieden. Stattdessen wird der Datentyp als Attribut des Datenpunktes gespeichert. Grund hierfür ist, dass dies das Durchreichen von Datenpunkten durch die Systemschichten erleichtert, da nicht für jeden Datenpunkttyp eigene Methoden bereitgestellt werden müssen.

Der Datenpunkt soll so aufgebaut werden, dass er alle relevanten, finalen Informationen für seine Darstellung beinhaltet. Damit wird der sich ändernde, aktuelle Wert des Datenpunktes ausgeschlossen, der erst in einer abgeleiteten Klasse hinzukommen soll. Grund hierfür ist, dass die Option bestehen soll, das Datenpunkt-Objekt in einer Datenbank abzulegen und bei jedem Neustart auszulesen. Der Datenpunktwert sowie Hilfsvariablen für das Management der Datenpunkte sollen in diesem Fall nicht mitgespeichert werden und können in dem Communication-Paket in abgeleiteter Form hinzugefügt werden.

Zusätzlich erbt der Datenpunkt von der abstrakten Klasse DataPointObservable, welche beliebigen Objekten erlaubt, sich als Beobachter des Datenpunktes zu registrieren.

Weiterhin befindet sich die Value-Klasse im Model-Paket. Diese Klasse ist für das Speichern von Datenpunktwerten verantwortlich und kann entweder über einen float, String oder boolean Konstruktor erstellt werden, die den Datenpunkttypen entsprechen.

7.3.3 View

Das Layout und der Style des GUIs wurden bereits in Kapitel 7.2 behandelt und sollen hier nicht weiter vertieft werden. Dieses Kapitel soll sich mit dem

GUI-Framework sowie der Schnittstelle zum Commnunication- und Model-Paket beschäftigen.

GUI-Framework

Um die Applikation modular, austauschbar und erweiterbar zu machen, wird ein modular zusammengesetztes GUI, das auf ein selbst erstelltes GUI-Framework aufbaut implementiert, welches das bestehende Android GUI erweitert. Die Möglichkeit GUIs mit XML zu definieren kommt dieser Architektur sehr entgegen. Die Softwarebasis wurde so gebaut, dass die gesamte Software mit wenigen Eingriffen in den Code über die XML-Layouts definiert wird. So wäre es innerhalb kürzester Zeit möglich, eine Visualisierung für ein anderes Gebäude umzusetzen, sofern Bilder für die Hintergründe und eine funktionierende Regelung vorhanden wären. Abbildung 28 zeigt die Hierarchie des implementierten GUI Frameworks.

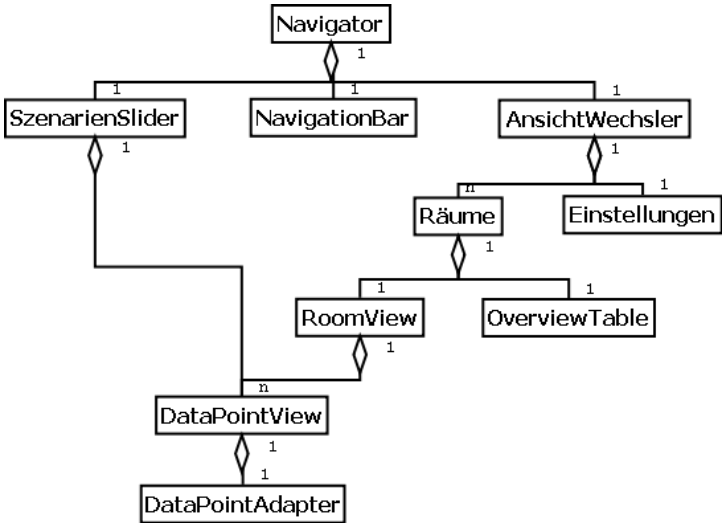


Abbildung 28: View Hierarchie

Die in Kapitel 7.2.7 vorgestellten Datenpunkt-Visualisierungen wurden jeweils als eigene View-Klassen (DataPointView) implementiert, die von einer entsprechenden Klasse, wie beispielsweise Button, erben. Diese erstellten Views müssen lediglich in einer XML-Datei implementiert und mit den richtigen Datenpukt-IDs versehen werden. Die anderen Komponenten der Applikation übernehmen den Rest. Wird die Applikation ausgeführt, wird das View-Objekt automatisch mit dem entsprechenden Datenpunkt verbunden und zeigt dessen aktuellen Wert an.

```
<ch.sauter.android.buildingrc.view.DPToggleButton
      sauter:read_datapoint="16777218"
      sauter:write_datapoint="20971522"
      android:id = "@+id/projektor_auditorium"
      android:drawableLeft="@drawable/ic_projector"
      android:background="@drawable/btn_toggle"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_y="25dip" android:layout_x="51dip"
/>
```

Tabelle 12: Implementieren eines Binären Datenpunktes

Tabelle 12 zeigt wie beispielsweise der Code für das Anzeigen des Projektor-Datenpunktes im Auditorium aussieht. Die ersten zwei Einträge definieren, welcher Datenpunkt ausgelesen und auf welchen Datenpunkt geschrieben wird. Dies geschieht analog zu dem in Kapitel 6.4 definierten Auslesen und Schreiben auf Datenpunkten. Mit android:drawableLeft wird das Projektor-Symbol links auf den Button gesetzt und android:background ändert das Aussehen des Buttons in das selbsterstelle Look and Feel. Es wird hierbei auf eine weitere XML-Datei referenziert, die je nach Zustand des Buttons (gedrückt, markiert, an, aus) die richtige PNG-Grafik anzeigt.

Layout_y und layout_x sind für die Positionierung des Buttons im Hintergrundbild verantwortlich. Es handelt sich hierbei um prozentuale Angaben. Die Werte stehen also für 25% nach unten und 51% nach rechts, relativ zum Hintergrundbild. Um dies zu ermöglichen musste eine eigene Layout-Klasse entworfen werden, da das Android-Framework keine Klasse mit dieser Funktionalität bietet. Android bietet zwar ein Layout an, in dem man absolute Pixel-Positionen angeben kann, dies würde jedoch nur auf dem kleinen Display korrekt dargestellt werden und keine Skalierungen erlauben. Auf dem großen Touch-Device würden die Symbole dann alle links oben im Eck dargestellt werden, während das Hintergrundbild auf die Größe des Displays skaliert werden würde. Durch die relativen Angaben der Positionen, werden die GUI-Elemente nun unabhängig von der Skalierung immer an der richtigen Stelle dargestellt.

View Paket Schnittstellen

Zwischen dem View und dem Modell Paket wird der Behavioural-Design-Pattern Observer eingesetzt. Jedes GUI-Objekt, das einen Datenpunkt visualisiert implementiert das DataPointView Interface, das eine Update Methode beinhaltet. Zusätzlich hat jeder DataPointView ein Attribut des Typs

DataPointAdapter. Dieser dient dazu, die Schnittstelle zwischen den Paketen auf eine Klasse zu minimieren und bündelt sämtliche Kommunikation mit anderen Paketen.

Der DataPointView übergibt dem DataPointAdapter die Datenpunkt-IDs, welche er per XML-Attribute übergeben bekommen hat. Der DataPointAdapter wertet diese aus und fragt den CommunicationManager aus dem Communication-Paket nach den entsprechenden Datenpunkten. Wurden diese erhalten, registriert sich der DataPointAdapter als Observer des Read-Datenpunktes. Sobald sich dessen Wert ändert, wird der DataPointAdapter informiert und dieser wiederum informiert den zugehörigen DataPointView, der dann seine Ansicht ändert. Diese Vorgänge sind auf Abbildung 29 und Abbildung 30 als Sequenzdiagramme visualisiert.

Zusätzlich ist in Abbildung 30 zu sehen, dass der DataPointAdapter, nachdem er die Information über die Änderung des Read-Datenpunktes erhalten hat, den Write-Datenpunkt auf den gleichen Wert setzt. Dabei benutzt er neben dem Datenpunkt-Wert „sync“ als Übergabeparameter. Grund hierfür ist das in Kapitel 5.3.4 angesprochene Synchronisieren der Datenpunkte, das nach Use-Case 3 implementiert wurde.

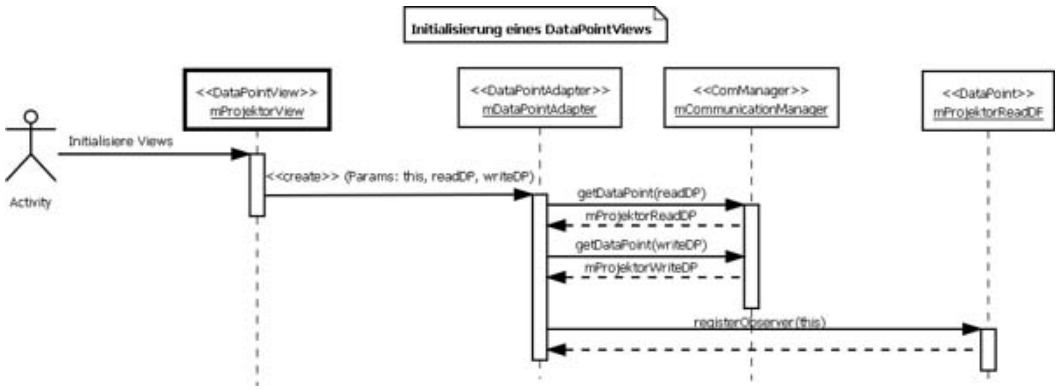


Abbildung 29: Initialisierung des DataPointViews

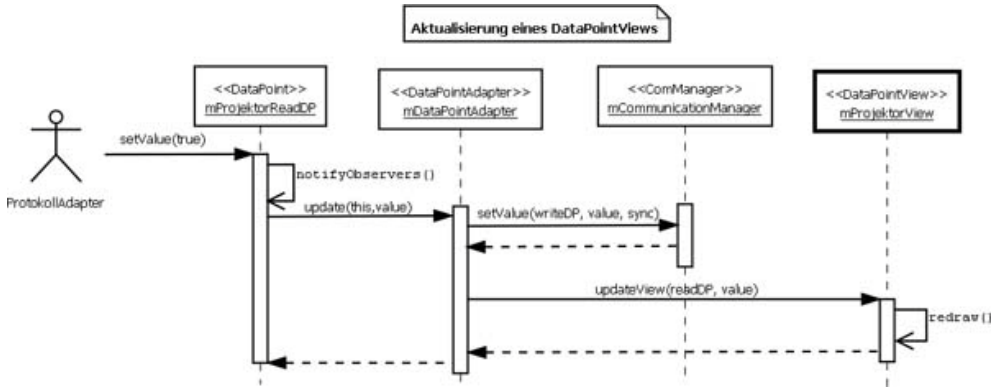


Abbildung 30: Aktualisierung des DataPointViews

Abbildung 31 zeigt den Prozess, der bei Änderungen eines DataPointViews durchlaufenen wird. Hier wird die setValue-Methode für den Write-DataPoint im Gegensatz zu der Synchronisierung in Abbildung 30 mit dem „manuell“-Parameter aufgerufen, da es sich in diesem Fall um eine vom Benutzer veranlasste Änderung handelt. Anschließend wird mit der syncDataPoints-Methode der Read-Datapoint so lange immer wieder ausgelesen, bis er den gesetzten Wert erreicht oder in einen Time-out läuft. Dies geschieht analog zu Use-Case 2. Schlussendlich werden alle Werte neu abgerufen.

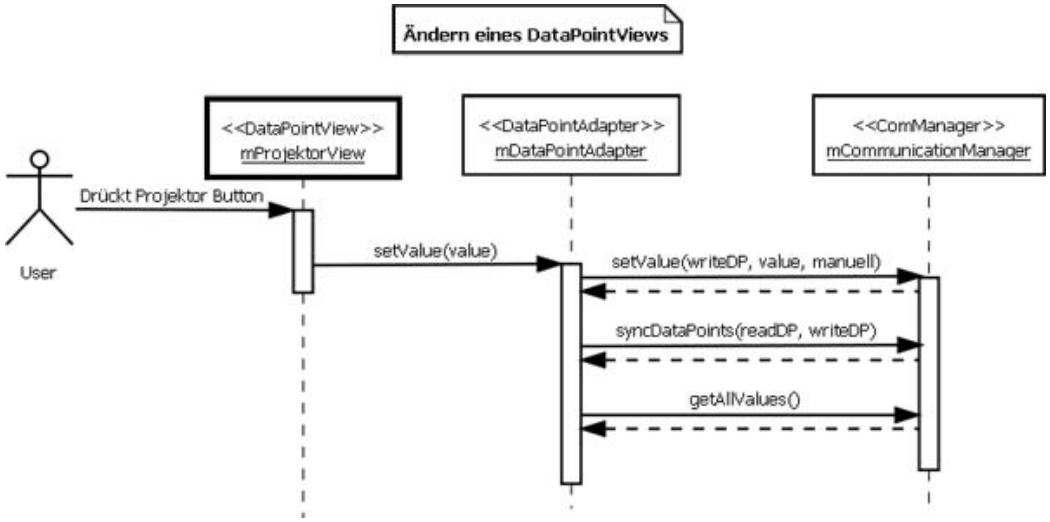


Abbildung 31: Änderung eines DataPointViews

7.3.4 Controller / Communication

Das Communication-Paket beinhaltet die Datenpunktverwaltung sowie die Schnittstelle zur Automationsstation, was die ganze Netzwerkkommunikation und beinhaltet.

Das Ändern eines Datenpunktwertes wird immer nur von dem Protokoll, das eine Verbindung zur Automationsstation hat, durchgeführt. Will ein GUI-Objekt den Wert eines Datenpunktes ändern, wird der Datenpunkt sowie der gewünschte Wert an den CommunicationManager weitergegeben. Dieser reicht Datenpunkt und Wert an das richtige Netzwerkprotokoll weiter, welches den Wert im Datenpunktobjekt erst dann setzt, wenn der Netzwerkbefehl zum Setzten des Wertes von der Automationsstation bestätigt wurde. Wenn dies nicht möglich ist, wird eine Fehlermeldung ausgegeben und der alte Wert des Datenpunktes bleibt bestehen. So bleibt der Datenpunktwert immer in einem konsistenten Zustand.

CommunicationManager

Basis und Schnittstellenklasse ist der CommunicationManager. Diese Klasse verwaltet die Datenpunkte und gibt Anfragen, Datenpunkte auszulesen oder zu schreiben, an die richten Objekte weiter.

Die Klasse ist als Singleton implementiert, da jeweils nur eine Instanz erwünscht ist. Alle Netzwerkaktivitäten gibt die Klasse in den eigens dafür bestimmten Netzwerk-Thread ab, um das GUI stets ansprechbar zu halten. Zusätzlich hat der CommunicationManager einen Listener auf die Poll-Einstellung der GUIs gesetzt. Bei der Aktivierung der Polling-Funktion wird einer Poll-Klasse, die wiederum in einem eigenen Thread läuft, der Auftrag gegeben, alle 3 Sekunden die getAllValues-Methode aufzurufen.

Protokollverwaltung

Um das Implementieren verschiedener Protokolle zu ermöglichen, die für den CommunicationManager aber transparent sind, wird eine Abstraktionsschicht geschaffen. Diese wird in zwei Ebenen realisiert. Die erste Ebene wird über das Adapter-Pattern aufgebaut.

Der Adapter besteht aus der abstrakten Klasse AbsProtokolAdapter, welche alle Methoden für das Lesen und Schreiben von einzelnen und mehreren Datenpunkten, sowie das Synchronisieren zweier Datenpunkte beinhält. Diese Klasse wird jeweils von BACnetAdapter und WebServiceAdapter implementiert, welche dann die eigentliche Netzwerkkommunikation mit dem entsprechenden Protokoll übernehmen. Aufträge, Datenpunktwerte zu lesen oder zu schreiben, werden dann jeweils mit einer Priorität in die Schlange des Netzwerk Threads eingereiht, die schnellstmöglich abgearbeitet wird. Zusätzlich wird ein DummyProtocolAdapter implementiert, der keine Netzwerkaktivitäten durchführt und für Testzwecke da ist.

Für BACnet sowie Webservices wurden Funktionen zum Abrufen und Schreiben von einzelnen und mehreren Datenpunktwerten implementiert. Für BACnet sollte zusätzlich noch die Change of Value Benachrichtigung implementiert werden, die das Pollen der Werte ersparen würde. Aufgrund des Aufwandes den es bedeutet hätte dieses Feature zu implementieren musste darauf verzichtet werden.

Die zweite Abstraktionsebene besteht aus einer angepassten Form des Factory-Patterns. Dieser wird in der ProtokolFactory-Klasse angewendet. Der CommunicationManager besitzt eine Instanz dieser Klasse und ruft für Netzwerkaktivitäten die Methode ProtokolFactory.getActiveProtocol auf, welche die aktuell vom Benutzer eingestellte AbsProtokolAdapter-Implementierung (BACnetAdapter, WebServiceAdapter oder DummyAdapter) zurückgibt.

Dies wird erreicht, indem die ProtocolFactory einen Listener auf die Protokoll-Settings setzt. Entsprechend dem gewählten Protokoll initialisiert die ProtocolFactroy die entsprechende konkrete AbsProtokollAdapter Implementierung und legt diese als aktives Protokoll ab. Bei dem Aufruf der getAktiveProtocol-Methode wird dann dieses Objekt zurückgegeben.

Datenpunkterweiterung

Um Datenpunkten überhaupt einen Wert geben zu können, wird die DataPoint-Klasse als ComDataPoint abgeleitet. Sie wird um Methoden zum Lesen und Schreiben von Werten, sowie um Attribute und Methoden zum Bestimmen von Zuständen des Datenpunktes erweitert. So wird der Zustand „updating“ hinzugefügt. Sobald ein getValue auf einen Datenpunkt gemacht wird, wird dieser von dem entsprechenden Protokoll-Adapter auf „updating“ gesetzt und in die Netzwerkschlange gegeben. Nachdem der Wert über das Netzwerk bezogen wurde, wird „updating“ wieder zurückgesetzt. Dies garantiert, dass ein Datenpunkt sich immer nur einmal in der Netzwerkschlange befindet.

Lesen und Schreiben von Datenpunkten

Der Lese- und Schreibvorgang eines Datenpunktes wie er in Abbildung 30 und Abbildung 31 aus Sicht des View-Paketes zu sehen ist, wird in den folgenden Abbildungen aus Sicht des Communication-Paketes dargestellt. Während in Abbildung 32 Webservices das aktive Protokoll ist, ist auf Abbildung 33 BACnet das momentan aktive Protokoll.

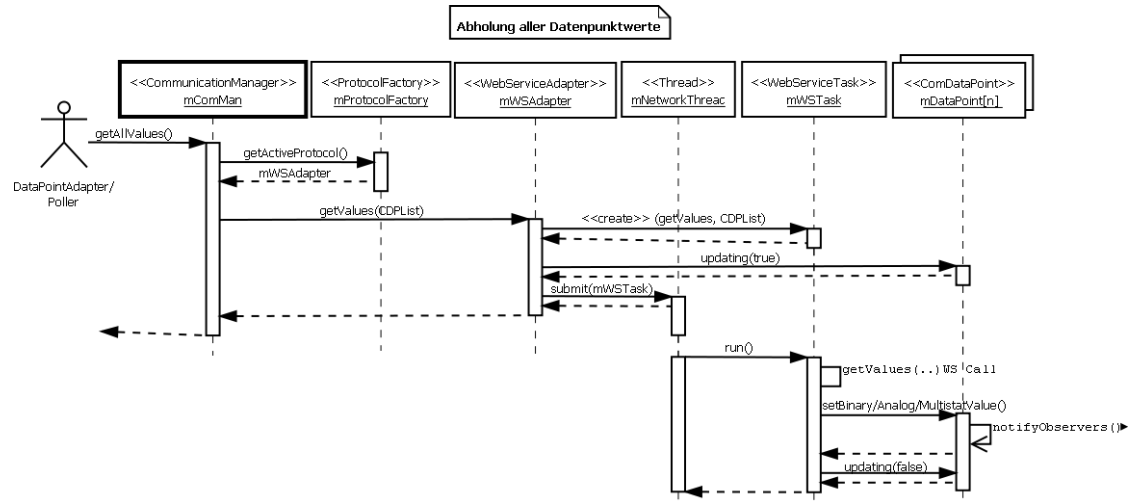


Abbildung 32: Lesen aller Datenpunkte

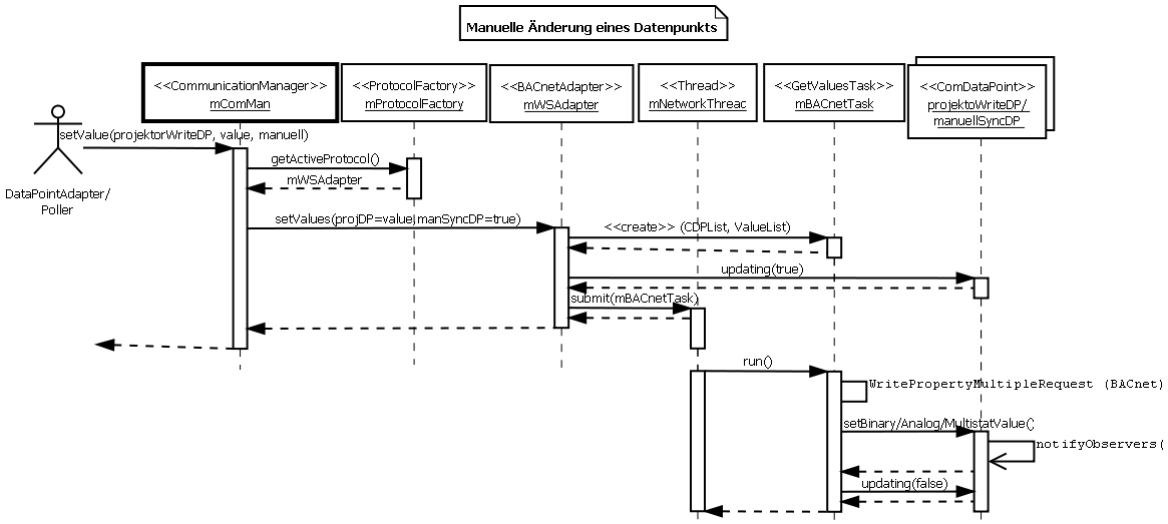


Abbildung 33: Manuelles schreiben eines Datenpunktes

Zu sehen ist, dass beim Schreiben des Projektors zusätzlich der Manuell/Sync-Datenpunkt gesetzt wird. Da es sich um einen manuellen Aufruf handelt, wird er in diesem Fall auf „true“ gesetzt. Würde es sich bei dem Aufruf um einen Synchronisierungsaufruf handeln, wäre er auf false gesetzt worden.

Indem vor jedem Schreiben eines Datenpunktes der Man/Sync Datenpunkt gesetzt wird, kann garantiert werden, dass das Schreiben der Werte von der Regelung richtig interpretiert wird. Mit der setValues Methode ist es möglich, beide Werte mit kaum Mehrraufwand in einem einzigen BACnet- bzw. WebService-Request zu schreiben.

8 Android Portierung auf ein 10,4“ Touch-Device

8.1 Allgemein

Um eine geeignete Hardware zu finden, werden zunächst verschiedene Hersteller von x86 Industrie Touch PCs bezüglich entsprechenden Geräten mit Linux-Unterstützung und kurzen Lieferzeiten befragt. Zusätzlich gab ein Besuch bei dem OHA Mitglied Noser Engineering AG [18] in Winterthur Aufschluss über Portierungen auf x86 und ARM sowie mögliche ARM Geräte mit entsprechenden Displaygrößen geben.

Noser hat als Mitglied der OHA einen großen Teil des Android-Application-Frameworks umgesetzt. Sie sind darauf spezialisiert, Android als Bedien- und Steuerinterfaces in der Industrie einzusetzen. Noser verfügt daher über eine hohe Kompetenz Portierungen von Android auf verschiedene Hardware durchzuführen.

8.2 Touch Technologien

Zunächst wird die zu verwendende Touch-Technologie untersucht, welche eine große Auswirkung auf die Usabilty eines Programms haben kann.

Momentan teilt sich der Großteil des Marktes in drei Technologien auf: Resistive-, Kapazitive- und Infrarot-Touchscreens.

Resistive Touchscreens sind hiervon die älteste Technologie und bestehen aus zwei leitfähigen Indiumzinnoxidschichten, welche sich beim Ausüben von Druck berühren und damit für einen Stromfluss sorgen. Somit kann festgestellt werden, ob und wo das Display berührt wurde [19].

Kapazitive Touchscreens bestehen aus mit durchsichtigem Metalloxid beschichteten Glassubstraten, die unter Spannung gesetzt werden. Durch die Kapazität des menschlichen Körpers, entsteht ein kleiner Entladungsstrom bei einer Berührung mit der Hand, durch welche die berührte Position berechnet werden kann [19].

Infrarot-Touchscreens besitzen als Berührungsfläche eine herkömmliche Kunststoff- oder Glasscheibe. In die Ränder der Screens sind Infrarot-Sender und -Empfänger eingebaut. Sobald ein lichtundurchlässiger Gegenstand auf

das Glas kommt, wird das IR Licht durchbrochen und die Position kann festgestellt werden [19].

Nachdem, alle drei Technologien getestet wurden, fiel die Wahl auf IR-Screens. Gegen Resistive Touchs spricht, dass der Druck eines Fingers oft nur unpräzise interpretiert wurde und eine Aktion oft mehrmals durchgeführt werden musste, bis das Touch-Display sie korrekt erkannte. Resistive Touchs sollten mit dafür vorgesehenen Stiften bedient werden und sind in den meisten Fällen nur schlecht für eine Bedienung mit der Hand geeignet.

Dieses Problem haben kapazitive Touch-Displays nicht, jedoch muss hier die nackte Hand benutzt werden. Das Steuern mit einem Handschuh oder sonstigen Hilfsgegenständen ist nur dann möglich, wenn diese eine Eigenkapazität haben.

IR-Touch-Displays überraschten mit einer sehr genauen Auflösung der Touchpunkte. Sie können mit beliebigen Gegenständen gesteuert werden und eignen sich aus diesen Gründen für das geforderte Device am besten.

8.3 Evaluation

Im nächsten Schritt wurden unterschiedliche Hersteller von x86 Industrie PCs mit IR-Touchscreens auf Linux-unterstütze Geräte mit kurzer Lieferzeit angefragt. Zusätzlich wurde der Termin bei Noser wahrgenommen, in dem das Projekt vorgestellt wurde sowie einige Fragen zur Portierung von Android beantwortet wurden.

Laut Noser ist Android auf der x86 Plattformen grundsätzlich lauffähig. Auch sie haben Android auf einen Dell Laptop portiert und gebootet. Es gibt jedoch einige Probleme mit der Benutzung verschiedener Flashspeicher bzw. Festplatten sowie Eingabegeräten. Insgesamt sind Portierungen auf die X86 Plattform sehr aufwändig und häufig nicht stabil. Für Google steht im Moment noch ARM als zentrale Plattform in Mittelpunkt. In den kommenden Monaten und Jahren ist laut Noser jedoch mit einer nativen Unterstützung der x86 Plattformen zu rechnen.

Portierungen von Android auf ARM-Plattformen wurden bei Noser ebenfalls erfolgreich durchgeführt. Diese sind mit wesentlich weniger Aufwand verbunden, auch wenn es hier ebenso schon zu Problemen mit Eingabe- und Periphe-

riegeräten gekommen ist. Grundfunktionalitäten sollten jedoch mit nur wenigen Problemen portierbar sein.

Aufgrund dieser Aussagen und dem Angebot von Noser uns eine Portierung mit den gewünschten Anforderungen auf Basis eines Toradex Colibri Evalboards durchzuführen, fiel die Wahl auf die ARM Plattform.

8.4 Portierung auf ein ARMv5 Toradex Colibri Evalboard

8.4.1 Verbaute Hardware

Als Basis wird, wie bereits erwähnt, ein Toradex Colibri Evaluation Board benutzt. Auf diesem befindet sich eine Colibri PXA320 Prozessorplatine, die mit einem Marvell ARMv5 Prozessor ausgestattet ist. Tabelle 13 zeigt die wichtigen technischen Daten des Boards und der Prozessorplatine. Für nähere Informationen siehe [20] und [21].

Toradex Colibri Evalboard mit PXA320 CPU Prozessorplatine	
CPU	Marvell PXA320 (ARMv5)
Taktfrequenz	806 MHz
RAM	128 MB DDR (32 Bit)
Flash	1 GB (8 Bit)
Ethernet	10/100Mb
USB	2xUSB Host
Display	VGA / Generic Display Port

Tabelle 13: Technische Daten des Colibri Boards

Für die Visualisierung wird ein AUO 10.4“ TFT-LCD Display mit einer 800x600 Auflösung verwendet, welches über LVDS angesprochen wird. Um es mit dem Toradex Board zu verbinden, wird ein Toradex LVDS Converter eingesetzt, der an den Generic Display Port angeschlossen wird.

Für die Touch-Bedienung wird ein 10,4“ Infrarot Touch von IR Touch Systems eingesetzt, das über USB mit dem Toradex-Board verbunden wird.

8.4.2 Portierung

Die Portierung wurde von Marcel Ziswiler bei Noser Luzern durchgeführt. Sie fand in drei Tagen statt, wovon ich zwei Tagen anwesend war. Sie teilt sich in drei Hauptkategorien ein. Zum einen der Android Linux Kernel, auf dem der

Rest des Android Frameworks ohne Anpassungen laufen sollte. Zum anderen das Portieren des Touchscreens sowie das des LCD Displays.

Android

Um den Android Kernel auf dem Toradex-Board laufen zu lassen, wurde er mit dem Toradex Linux-Kernel gemerged, um sämtliche Hardware auf dem Gerät zu unterstützen. Benutzt wurde der neueste Android Kernel 2.6.28 sowie der neueste Toradex Kernel, ebenfalls in der Version 2.6.28. Die Android-Plattform entspricht einem Checkout der Cupcacke Version vom 28.07.2009. Hierbei handelt es sich um die neueste Version von Android.

Es mussten nur minimale Anpassungen an das Android-System durchgeführt werden. Zum einen musste das Powermanagement von Android gehacked werden, da das System nach dem Booten sofort wieder heruntergefahren wurde. Grund hierfür war, dass weder ein Akku noch eine externe Stromversorgung erkannt wurde und das System ein Not-Shutdown verursachte, um einen Datenverlust zu vermeiden. Durch einen Hack wird der Akku nun immer auf 100% voll angezeigt. Weitere Kernelanpassungen werden im nächsten Abschnitt erläutert.

Touchscreen

Da das Android-System auf kapazitive Touch-Screens ausgelegt ist, welche nicht kalibriert werden müssen, stellt das System keine Möglichkeit zur Verfügung den IR-Touch zu kalibrieren. Um dies zu bewerkstelligen, wurde TSLib [22], eine C-Bibliothek für Touch-Kalibrierung, auf dem System installiert.

Um die daraus generierte Kalibrations-Konfiguration in das Android-System zu integrieren, wurde die InputDevice-Klasse aus dem Java Framework so abgeändert, dass sie die Kalibrations-Daten auf die rohen Touch-Positionen anwendet, die von der Hardware übermittelt werden.

Ansonsten musste lediglich die Unterstützung von USB-Touch-Devices in der Kernel-Konfiguration aktiviert werden.

Display

Für das Display mussten zunächst die Display-Konfigurationen angepasst werden. Zusätzlich ergab sich das Problem, dass das Embbedded Linux 16Bit Farben ausgab, der Generic Display Port jedoch 18Bit Farben erwartet. Die von

Android ausgegebenen 16Bit Farben ließen die zwei Most Significant Bits vom Rot Leer. Für eine korrekte Darstellung müssten aber jeweils das Least Sigificant Bit von Blau und Rot leer gelassen werden. So mussten die Grafik-Output Pins der Prozessorplatine mit Steckkabeln auf die richtigen Pins des General Displayport gelegt werden, um eine korrekte Farbdarstellung zu erhalten.

8.4.3 Ergebnis

Nach den vorgenommenen Anpassungen lief das Android-System mit Touch und LCD-Display in 800x600 Auflösung auf dem Toradex Board.

Das Android-System verfügt bereits über einen DHCP-Client, mit dem die Ethernet-Schnittstelle gestartet und benutzt werden kann. Diese wird ohne Probleme an die Java-Applikationen weitergereicht und erlaubt problemlosen Zugriff auf das Netzwerk.

Auch der Android ADB Server funktioniert einwandfrei und ermöglicht es das Toradex-Board auf dem Entwickler PC als Android Device zu erkennen und es wie ein G1 oder Emulator mit allen Android Development Tools anzusprechen. Abbildung 34 zeigt eine Beta-Version der Entwickelten Android Software auf dem Toradex Evaluierungsboard mit Touchscreen und LCD-Display. Auf dem Board sind in blau und orange klar die Überbrückungskabel zu sehen, die für die korrekte Ansteuerung des LCD-Displays nötig waren.

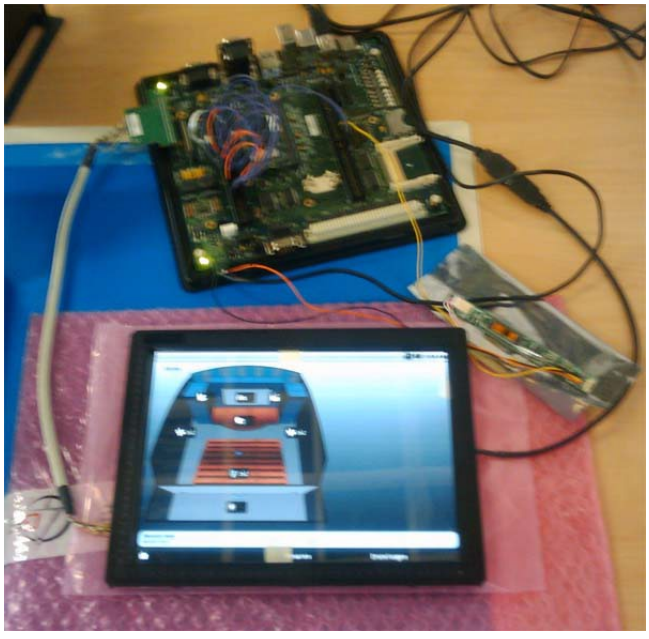


Abbildung 34: Android auf Toradex Board

Obwohl Android grundsätzlich auf dem Toradex Board läuft, gibt es noch einige Probleme, die genauer untersucht und gelöst werden müssen. Dies betrifft zum einen die Eingabe von Daten. Für Text und Zahleneingaben wird eine virtuelle Tastatur auf dem Display eingeblendet, die über das Touch bedient werden kann. Grundsätzlich funktioniert diese sehr gut und zeigt die Genauigkeit des Infrarot Touchs. Gibt man jedoch als erstes Zeichen eine Zahl ein oder drückt man die Backspace-Taste, stürzt die gestartete Applikation ab. Dieses Verhalten zeigt sich durchgängig bei allen Applikationen, bei denen Text oder Zahleneingaben möglich sind.

Ein zweites Problem besteht bei hardwareunterstützten Berechnungen von Animationen. Während rein Software-basierte Animationen ohne Probleme dargestellt werden, erscheint bei hardwareunterstützten Animationen ein Blinken bei der Überlagerung von Bildern.

9 Tests

9.1 Allgemein

Nachdem alle Komponenten des Systems fertiggestellt sind, werden diese nun als Gesamtsystem getestet. Die Tests sollen dazu dienen, eine Beurteilung über die Praxistauglichkeit der Android-Software aus Sicht der Netzwerkgeschwindigkeiten sowie der Usability machen zu können.

Hierzu werden die Reaktionszeiten vom Betätigen einer Datenpunktänderung in der Software bis zu der physischen Reaktion der Elektronik auf dem G1 und dem Toradex Board gemessen.

Zusätzlich wird ein Usability-Test durchgeführt, in dem verschiedene Personen Aufgaben mit dem Umgang der Software zu lösen haben und die Schwierigkeit dieser bewerten sollen.

9.2 Reaktionszeiten

Das Toradex Evalboard ist als Stationäres Gerät denkbar, das in die Wand eingelassen wäre und sich neben Türen bzw. in dem Podium befände. Eine Anbindung an das lokale Netzwerk ist hier vorgegeben und so werden sehr niedrige Reaktionszeiten erwartet. Sofern sich das Board in dem gleichen Netz wie die Automationsstationen befindet, ist BACnet zu bevorzugen. Wird es allerdings von dem BACnet Netz getrennt, können auch Webservices eingesetzt werden.

Das HTC G1 ist ausschließlich auf Wireless-Verbindungen angewiesen. Hier können wahlweise WLAN oder Mobilfunknetze eingesetzt werden. Im Normalfall wird sich das Gerät nicht im gleichen Netz wie die Automationsstationen befinden, trotzdem sollen die Reaktionszeiten von BACnet über WLAN gemessen werden. In der Regel dürften jedoch Webservices das bevorzugte Protokoll auf dem G1 sein. Abbildung 35 zeigt den Aufbau, der für die Tests vorgenommen wurde.

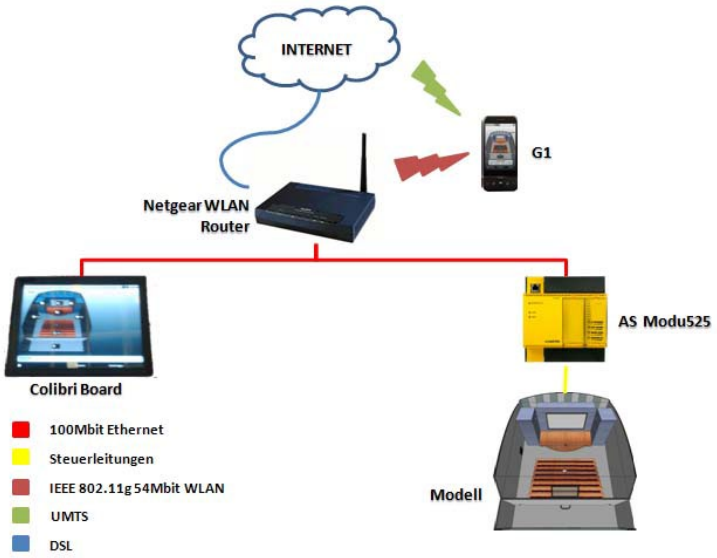


Abbildung 35: Testaufbau

Um die Reaktionszeiten feststellen zu können, wurde in das in Abbildung 35 rot gekennzeichnete Ethernet Netzwerk, ein Linux PC integriert, der mit Wireshark das Netzwerk überwacht.

Da nach dem Setzen eines Wertes immer der entsprechende Ausgangs-Datenpunkt so lange abgerufen wird, bis dieser sich zu dem gewünschten Wert ändert, können die Abstände zwischen den einzelnen Nachrichten gemessen werden. Dies entspricht der Zeit, die benötigt wird, um eine entsprechende WebService- bzw. BACnet-Anfrage in der Applikation aufzubauen, über das Netz zu schicken und von der Station bestätigen zu lassen. Um ein repräsentatives Ergebnis zu erhalten, wurden jeweils mindestens 50 Webservice bzw. BACnet Anfragen abgeschickt und anschließend der Wert gemittelt.

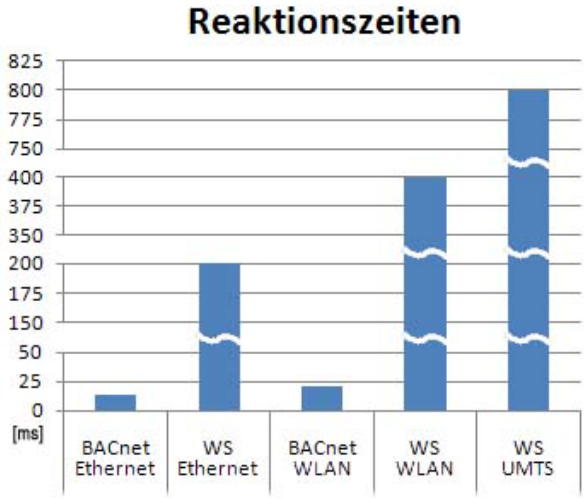


Abbildung 36: Reaktionszeiten

Das Ergebnis zeigt einen nicht ganz so deutlich erwarteten Unterschied zwischen BACnet und Webservices. Auch wenn es klar war, dass BACnet schneller wie SOAP Webservices ist, ist der Unterschied der Reaktionszeiten doch sehr enorm. BACnet kommt über Ethernet auf Reaktionszeiten von 6ms, wohingegen Webservices ebenfalls über Ethernet immer noch 100ms-400ms benötigen. Dieser Unterschied hat verschiedene Ursachen. Die Gravierendste dürfte die Benutzung von UDP bei BACnet im Gegensatz zu TCP bei Webservices sein. Während bei BACnet lediglich zwei UDP-Pakete genügen, eines für die Anfrage und eines für die Bestätigung, werden für das Aufrufen eines Webservice-Requests mindestens 12 TCP Pakete, bei Anfragen mit vielen Datenpunkten sogar mehr benötigt. Hinzukommt, dass BACnet auf der AS nativ implementiert ist und die Webservices in Java vorliegen.

Trotz den großen Unterschieden liegen beide Protokolle bei der Verwendung von Ethernet oder WLAN deutlich unter den geforderten 1000ms. Sogar bei der Kommunikation über UMTS konnte ein Durchschnittswert unter einer Sekunde erreicht werden. Hier schwanken die Werte jedoch deutlich. Es wurden Werte zwischen 300ms und 3 Sekunden gemessen. Diese Latenzen sind jedoch eindeutig auf das UMTS-Netz bzw. den Sleep-Modus des Mobile Device zurück zu führen. Dieses fährt bei Nichtbenutzung der UMTS-Schnittstelle diese herunter, um Strom zu sparen. Bei der ersten Nachricht nach einem solchen Sleep verursacht das Hochfahren eine etwas höhere Latenz.

9.3 Usability Tests

Um zu testen, ob die Software die Anforderungen zur Bedienbarkeit aus Kapitel 3.4.4 erfüllen kann, sollen Usability Tests durchgeführt werden. Ein Test wird zusammengestellt, der das Aufrufen aller Grundfunktionen der Software beinhaltet und einem realen Szenario nachempfunden ist. Verschiedene Testpersonen sollen diese Aufgaben lösen und bewerten, wie leicht es für sie war, die entsprechenden Funktionen zu finden. Der Usability Test kann in Anlage B eingesehen werden. Es handelt sich hierbei um einen beispielhaften Test, für eine produktive Anwendung müsste dieser umfangreicher und mit mehr Personen durchgeführt werden.

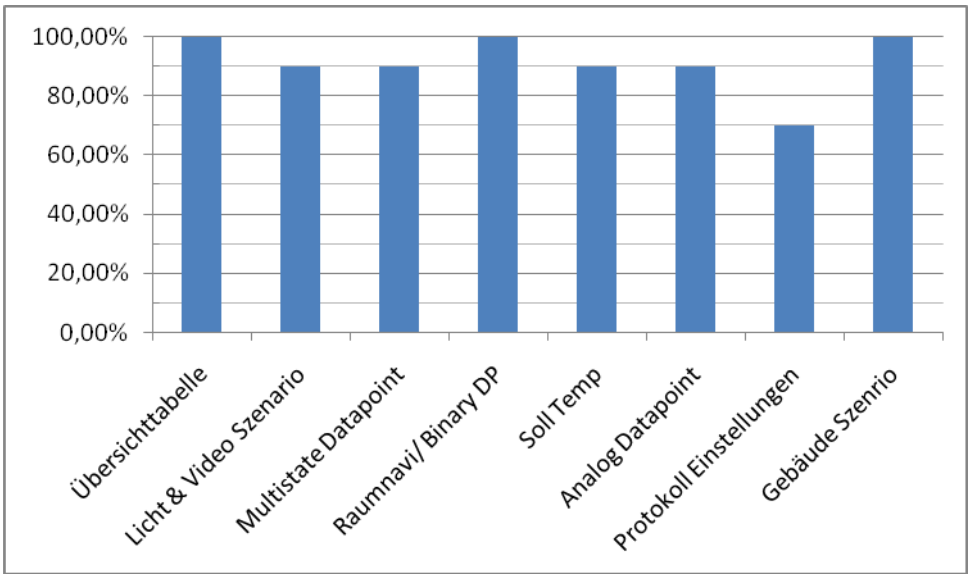


Abbildung 37: Ergebnis des Usability Tests

Das Ergebnis, das auf Abbildung 37 zu sehen ist, zeigt, wie viel Prozent der 10 befragten Personen die jeweilige Frage leicht lösen konnten. Auch wenn nur wenige Personen befragt wurden, war das Spektrum von Personen dennoch rech breit. Das Alter der Testpersonen lag zwischen 18 und 62 Jahren. Die befragten Personen üben außerdem sehr unterschiedliche Berufe aus, von Pädagogik-Studenten, über Industrie Designer bis hin zu Elektronikern.

Das Feedback über die Applikation fiel sehr positiv aus. Alle befragten Personen fanden sich grundsätzlich gut mit der Software zurecht. Bei einigen gab es an einer oder zwei Stellen Probleme, insgesamt hatten aber alle einen sehr guten Eindruck von der Software.

10 Bewertung der Android-Plattform für den Einsatz in der Gebäudeautomation

10.1 Allgemein

Die Entscheidung, ob sich Android als Software-Plattform für die Gebäudeautomation und speziell für den Einsatz bei Sauter eignet, obliegt grundsätzlich zwei Kriterien. Zum einen den wirtschaftlichen und zum anderen den technologischen Kriterien. In der hier verfassten Bewertung sollen lediglich die technologischen Aspekte betrachtet werden, die sich aus dem Software-Erstellungs-Prozess, den Debugging und Testmöglichkeiten, der Usability und den Reaktionszeiten zusammensetzen.

10.2 Android SDK

Softwareerstellung

Um auf der Android-Plattform Software zu entwickeln, ist eine Auseinandersetzung mit der Architektur und dem Design des Applikation-Frameworks unablässig. Dies betrifft unter anderem das Activity- und Intent-Prinzip sowie die GUI-Erstellung und das Recourse-Prinzip. Um das ganze Architekturkonzept zu verstehen und richtig anzuwenden, wird eine angemessene Zeit der Einarbeitung benötigt.

Allgemeine Software Architektur Prinzipen wie beispielsweise die Model-View-Controller-Architektur können weiterhin angewendet werden, wie auf Abbildung 27 zu sehen. Für die Umsetzung müssen jedoch die Application-Lifecycles, die Kommunikation zwischen Software und GUI-Elementen sowie weitere in Kapitel 2.3 vorgestellte Unterschiede von Android zu Sun Java Applikationen berücksichtigt werden.

So ist für den Aufbau von Acitvities eine gute Kenntnis über deren Lifecycles, den aufgerufenen Callback-Methoden sowie den Callback-Referenzen zwingend nötig. Es können sonst schnell ungewollte Speicherleaks entstehen, die entweder durch Unachtsamkeit bzw. Unwissenheit des Entwicklers oder durch bestehende Bugs im Framework entstehen und irgendwann zum Abstürzen der Applikation führen.

Nachdem eine Einarbeitung in das Android Framework geschehen ist, wird einem die Mächtigkeit des Frameworks klar. Programme können durch das Intent-Prinzip einzelne Komponenten anderen Programme benutzen und müssen nicht alles selbst umsetzten. Dies schafft eine große Modularität, die es ermöglicht in kurzer Zeit seht umfangreiche Applikationen zu entwickeln.

So wäre es möglich, eine Palette von Applikationen mit verschiedenen Anwendungszwecken für die Gebäudeautomation zu entwickeln, welche auf einer gemeinsamen Basis von Activities aufbauen.

Auch das Integrieren von externen Java-Bibliotheken, wie beispielweise die BACnet4J oder KSOAP2 Bibliotheken, die für die Netzwerkprotokolle genutzt wurden, erfolgte Problemlos. Dank der standardkonformen Umsetzung des Java-Frameworks sind hier wenige Probleme zu erwarten.

Debugging und Testing

Das Debugging von Applikationen findet in herkömmlicher Eclipse-Manier statt. Besonders gut ist die Möglichkeit, den Code live auf dem Emulator oder auf einer Android-Hardware zu debuggen.

Mit Hilfe der, in Kapitel 2.4 genauer beschriebenen, DDMS Ansicht, sowie den JUnit-Klassen und den zusätzlichen Test-Tools, die das SDK zur Verfügung stellt ist ein umfangreiches Debuggen, Testen, Profilen und Analysieren von erstellter Software möglich. An diesem Punkt ist Android wirklich beispielhaft.

GUI – Look and Feel

Das Look and Feel der Standard Android-Widgets ist teilweise ansprechend, teilweise wirken diese etwas veraltet. Für eine Steuerung über ein Touch-Display ist sie ideal geeignet und kann unter Verwendung von Animationen ein gutes Look and Feel erzeugen. Dies kann durch die in Kapitel 7.2.7 gezeigten Buttons, Regler und Schalter, sowie der in Kapitel 7.2.5 beschriebenen Szenario-Animationen belegt werden.

Die Möglichkeiten, das Design der Widgets sehr einfach zu verändern und eigene Animationen zu erstellen, machen das GUI zu einer sehr mächtigen, einfach zu benutzenden Grundlage. Der Ansatz, GUIs mit XML zu erstellen, ist ebenfalls eine willkommene Erneuerung im Gegensatz zu dem umständlichen Erstellen von Swing GUIs. Die Definition eines Datenpunkt-Views, die in Tabelle

12 zu sehen ist, veranschaulicht die simple und effiziente Weise des GUI-Layoutens. Selbst unerfahrenen Programmieren wird so die Chance geboten, ansprechende grafische Oberflächen zu erstellen.

Durch Erweiterungen und Umdesignen der vorhandenen Widgets konnten ansprechende, intuitive Datepunkt-Widgets erstellt werden. Und diese stellen nur einen Ausschnitt aus den Möglichkeiten dar, Regler und Schalter zu visualisieren.

10.3 Visualisierungs und Bediensoftware

Die implementierte Android-Software kann durchaus als erfolgreich angesehen werden. Sämtliche Anforderungen aus Kapitel 3.4 konnten erfüllt werden. Das einzige, was aufgrund von Zeitmangel nicht geschafft wurde, war die Implementierung der COV-Funktionalität auf Seiten des BACnet Clients.

Sowohl BACnet als auch Webservices zeigten sich als gute Protokollwahl. So können Standard AS Modu525 Stationen mit der Software angesprochen werden, ohne irgendeinen Eingriff in die Firmware vornehmen zu müssen.

Die Reaktionsergebnisse des Modells bestätigen, wie in Kapitel 9.2 nachzulesen ist, ebenfalls die Wahl der Protokolle. Einzig die Kommunikation über UMTS ergab teilweise etwas längere Latenzen, die aber übertragungsspezifischer Natur sind und nur Teilweise mit dem verwendeten Protokoll zusammenhängen.

Auch die Usability der Software konnte bei sämtlichen Testpersonen überzeugen und wurde wie das Diagramm auf Abbildung 37 zeigt als sehr intuitiv und verständlich empfunden, womit auch hier die Anforderungen erfüllt wären. Es zeigt sich somit, dass aus Sicht der GUI-Oberflächen sowie der verwendeten Netzwerkprotokolle Android durchaus als Plattform für Visualisierungs- und Bediensoftware geeignet ist.

10.4 Plattformunabhängigkeit / Portierung

Von Portierungen auf die X86 Plattform sowie allen anderen Plattformen außer ARM ist für den produktiven Einsatz, aus denen in Kapitel 8.3 beschrieben Gründen, momentan abzuraten. In der Zukunft könnte dies eine Möglichkeit

sein, momentan sind diese Portierungen allerdings mit zu vielen ungelösten Problemen und Risiken verbunden.

Auch für Portierungen von Android auf eine ARM-Architektur sind ohne Frage vor allem Linux-Spezialisten gefragt. Ohne tiefreichende Kenntnisse über den Linux-Kernel und seine Eigenschaften wie sie Herr Ziswiler von Noser Engineering besitzt, ist hier kaum etwas möglich. Hat man den Kernel mit sämtlichen Treibern jedoch portiert, läuft der Rest des Android-Systems zumeist mit sehr wenigen Anpassungen.

Kleine Probleme, wie bei der in Kapitel 8.4 beschriebenen Portierung auf das Colibri Evaluation Board, sind jedoch auch auf ARM-Plattformen zu erwarten. Nach der eigentlichen Portierung müssen meistens Bugs gefunden und behoben werden, die nicht zu unterschätzen sind. Von einer grundsätzlichen funktionierenden Portierung bis zu einem stabilen, marktreifen Produkt vergehen meist noch viele Stunden von Bugfixes.

10.5 Zusammenfassung und Ausblick

Insgesamt kann gesagt werden, dass das Android Prototypen Projekt sehr erfolgreich war. Alle Komponenten des Systems funktionieren, auch wenn der Regelungsplan nicht für Szenarien optimal ist und bei weiterem Bedarf solcher Funktionen entsprechende Bausteine zur Verfügung gestellt werden sollten.

Es ist möglich sowohl über das G1, als auch das Colibri Evalboard das Modell zu steuern. Die Reaktionszeiten sind im Ethernet und WLAN hervorragend, über UMTS stellt sich eine unvermeidbare Verzögerung ein.

Auch die Usability der Software konnte über die Usability Tests in Kapitel 9.3 verifiziert werden. Selbst systemfremden Personen war es schnell möglich, sich mit der Software zurechtzufinden.

So kann zusammenfassend gesagt werden, dass Android technologisch mit Sicherheit eine Option für die Gebäudeautomation darstellt. Die Portierungsprobleme müssen natürlich nochmals genauer untersucht werden, sollten sich jedoch in angemessener Zeit lösen lassen. Man muss sich bei Android jedoch stets im Klaren sein, dass es sich hierbei nicht um normale Java-Applikationen handelt, sondern speziell auf Embedded Touch Devices optimierte Software, die auch entsprechend designet und implementiert werden muss.

Bei einer produktiven Einführung von Android als Software-Plattform für Sauter wäre für jeden Entwickler eine umfassende Einarbeitungszeit, bestmöglich ein Seminar zum Android-Framework zu empfehlen.

Es wäre außerdem zu raten, einen kompetenten Partner in Sachen Android, wie beispielsweise Noser Engineering zur Seite zu nehmen, der Portierungen übernehmen oder begleiten könnte, sowie für Rückfragen bezüglich des Android-Frameworks zur Verfügung steht.

Unabhängig davon, ob Android oder eine andere Embedded Plattform eingesetzt wird, die Informationen über Webservices abruft, sollte man sich Gedanken machen, parallel zu SOAP RESTful-Webservices anzubieten, die vor allem auf Embedded Devices die benötigte Rechenleistung, den verbrauchten Speicher und die Netzwerklast senken würden.

Mit zunehmender Etablierung der Android-Plattform in der Mobile-Branche als auch in der Industrie wird Android mit Sicherheit eine immer interessantere Software-Plattform, die alle benötigten Mittel bietet, Java-Applikationen mit ansprechenden grafischen Oberflächen komfortabel zu entwickeln und umfangreich zu testen. Auch die Möglichkeit eigene C und C++ Bibliotheken und Programme auf Linux-Ebene einzusetzen, kann durchaus sehr attraktiv sein.

So wäre Android sogar ein Kandidat, der auf den linuxbasierten Sauter Automationsstationen selbst laufen könnte. Da die Stationen, wie in der Einleitung erwähnt, ebenso wie Android auf der unteren Ebene ein Embedded Linux und für High-Level Anwendungen, wie beispielsweise das Webinterface, eine Java VM nutzen, wäre eine eins zu eins Portierung denkbar.

Die im Laufe dieser Arbeit erworbenen Kenntnisse sprechen durchaus dafür, diese Möglichkeit in Betracht zu ziehen. Eine weitere Studie, welche die Portierbarkeit der AS-Software auf die Android Plattform untersucht und Benchmarkings durchführt, wäre ein denkbarer nächster Schritt.

Um letztendlich festzustellen, ob Android als Visualisierungs- und Bedien-Interface bei Sauter Zukunft hat, müsste eine Markt- und Wirtschaftsanalyse durchgeführt werden. Aus technologischer Sicht kann dies unter Berücksichtigung der in diesem Kapitel beschriebenen Maßnahmen mit einem klaren Ja beantwortet werden.

Anlage A: Literaturverzeichnis

[1] Offizielle Homepage der Open Handes Alliace <*http://www.openhandsetalliance.com/*> [Zugriff: 13.08.2009, 14:21 MEZ]

[2] Lizenz Bedingungen der Android-Plattform <*http://source.android.com/license*> [Zugriff: 07.09.2009, 13:27 MEZ]

[3] Offizielle Homepages zu Adobe Flex, Micosoft Silverlight und Sun Java FX <*http://www.adobe.com/products/flex/, http://silverlight.net/, http://javafx.com/*> [Zugriff: 07.09.2009, 14:33 MEZ]

[4] Offizielle Homepages zu Microsoft Windows Mobile, Symbian OS, Apple IPhone OS X und Android <*http://www.microsoft.com/windowsmobile/, http://www.symbian.org/, http://developer.apple.com/iphone/, http://www.android.com*/> [Zugriff: 07.09.2009, 15:11 MEZ]

[5] Embedded Linux Wiki: Android on OMAP <*http://elinux.org/Android_on_OMAP*> [Zugriff: 28.06.2009, 11:43 MEZ]

[6] Android-x86 Project - Run Android on Your PC <*http://www.android-x86.org/*> [Zugriff: 08.06.2009, 17:41 MEZ]

[7] R. Rogers, J. Kombarde, z. Mednieks und B. Meike (2009): Android Application Development, Programming with the Google SDK. Sebastopol: O'Reilly

[8] Burnette, Ed (2009): Hello, Android. Introducing Google's Mobile Development Platform. Dallas: The Pragmatic Bookshelf

[9] F. Ableson, C. Collins, R.Sen (2009): Unlocking Android, A Developer's Guide. Greenwich: Manning

[10] Google (2009): Google Android Online Dokumentation. Siehe Angang C, /Bachelorarbeit/Quellen.

[11] Offizielle Homepage des HTC Dreams <*http://www.htc.com/www/product/dream/overview.html*>[Zugriff: 07.09.2009, 15:55 MEZ]

[12] Google 3D-Warehouse <*http://sketchup.google.com/3dwarehouse/*> [Zugriff: 13.08.2009, 16:15 MEZ]

[13] Offizielle BACnet Homepage <*http://www.bacnet.org/* > [Zugriff: 13.08.2009, 16:44 MEZ]

[14] W3C (2007): SOAP Specefications <*http://www.w3.org/TR/soap/*> [Zugriff: 07.09.2009, 16:51 MEZ]

[15] (2009): Remote Method Invocation. Auf: *Wikipedia, the free encyclopedia*. <*http://de.wikipedia.org/wiki/Remote_Method_Invocation*> [Zugriff: 07.09.2009, 17:16 MEZ]

[16] Larman, Craig (2009): Applying UML and patterns : an introduction to object-oriented analysis and design and interative development. München: Patrice Hall

[17] O. Vogel, et al. (2009): Software-Architektur : Grundlagen Konzepte Praxis. Heidelberg: Spektrum Akademischer Verlag.

[18] Offizielle Homepae von Noser Engineering < *http://www.noser.com/*> [Zugriff: 07.09.2009, 17:33 MEZ]

[19] (2009): Touchscreen. Auf: *Wikipedia, the free encyclopedia*. <*http://en.wikipedia.org/wiki/Touchscreen*> [Zugriff: 06.09.2009, 13:54 MEZ]

[20] Toradex (2007) Colibri Evaluation Board Datasheet. Horw: Toradex

[21] Toradex (2009) Colibri XScale® PXA320 Datasheet. Horw: Toradex

[22] Offizielle Homapage des TSLib Projekts <*http://tslib.berlios.de/*> [Zugriff: 07.09.2009, 17:57 MEZ]

[13] Sourceforge Projekt der Bacnet for Java Bibliothek. Serotonin Software <*http://bacnet4j.sourceforge.net/*> [Zugriff: 09.09.2009, 14:50 MEZ]

Anlage B: Usability Test

Android Building Remote Control User Interface Test:

Name:_____ Alter:_____ Beruf:_____

1. Bitte notieren Sie die aktuelle Temperatur im Modell, die Temperatur, die das Modell haben sollte sowie das aktive Licht und Video Szenario:

   _____

   _____

   ☐ Leicht zu finden
   ☐ Schwer zu finden
   ☐ Nicht gefunden

2. Sie wollen mit ihrem Vortrag starten und ändern das Szenario auf "Beamer Präsentation".
   ☐ Leicht zu finden
   ☐ Schwer zu finden
   ☐ Nicht gefunden

3. Da es draussen sehr laut ist, schliessen Sie das Fenster.
   ☐ Leicht zu finden
   ☐ Schwer zu finden
   ☐ Nicht gefunden

4. Es kommt ein verspäteter Gast, bitte schalten Sie für ihn das Licht im Foyer ein.
   ☐ Leicht zu finden
   ☐ Schwer zu finden
   ☐ Nicht gefunden

5. Es ist Ihnen etwas zu kalt, bitte setzen sie die Solltemperatur auf 28°C hoch.
   ☐ Leicht zu finden
   ☐ Schwer zu finden
   ☐ Nicht gefunden

6. Da Ihre Powerpoint Präsentation zu Ende ist, Sie aber noch etwas sagen wollen, schalten Sie den Projektor aus. Damit man Sie besser sehen kann, schalten Sie zusätzlich den Bühnen Spotlight auf 100% Leuchtkraft.
   ☐ Leicht zu finden
   ☐ Schwer zu finden
   ☐ Nicht gefunden

7. Sie haben Probleme mit dem BACnet Protokoll und wollen stattdessen Web Services benutzen.
   ☐ Leicht zu finden
   ☐ Schwer zu finden
   ☐ Nicht gefunden

8. Sie verlassen das Gebäude und stellen Gebäude Szenario auf "Nicht Belegt" und das Protokoll wieder auf BACnet.
   ☐ Leicht zu finden
   ☐ Schwer zu finden
   ☐ Nicht gefunden

Haben Sie gefunden was Sie gesucht haben? Gab es Probleme? Haben Sie Verbesserungsvorschläge?

_____

_____

_____

_____

Anlage C: CDROM

Anlage D: Ehrenwörtliche Erklärung

Erklärung

Ich habe die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

---------- ----------

Ort Datum Unterschrift