FireTablet

Mobiles Service Interface zu Siemens Brandmeldeanlagen

Bachelorarbeit FS 2011

Daniel Bobst
(in Zusammenarbeit mit Samuel Hüppi)

INSTITUTE FOR
SOFTWARE

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate deutlich kenntlich gemacht zu haben.

Rapperswil, den 16.06.2011 Daniel Bobst

Aufgabenstellung

Moderne Brandmeldezentralen schützen Personen und Gebäude mit Hilfe modernster Technik. Hochoptimierte Rauchmelder unterscheiden zuverlässig Störgrössen von Bränden und alarmieren die Bewohner und die Feuerwehr oft bevor es überhaupt zu einem nennenswerten Schaden kommt. Siemens ist ein weltweit führender Hersteller solcher Anlagen und bietet in vielen Ländern sowohl die Produkte als auch entsprechende Serviceleistungen an.

Ein wichtiger Teil der Serviceleistungen ist die periodisch Wartung installierter Anlagen um sicherzustellen, dass diese stets zuverlässig funktionieren. Im Rahmen dieser Wartung werden die Rauchmelder auf korrekte Funktion geprüft und die Installation der Anlage kontrolliert.

Unser Ziel ist es immer die besten Produkte und Serviceleistungen anzubieten. Dazu gehört auch eine entsprechende Toolunterstützung für den Servicetechniker. Anhand eines portablen Gerätes mit Touchscreen, Kamera, Audiorekorder und Internetzugang (z.B. iPhone, iPad, Android Handy, etc.) sollen im Rahmen der Studie Benutzeroberflächen und Workflow-Konzepte erarbeitet werden mit denen der Service schneller und zuverlässiger erfolgen kann. Der Servicetechniker soll mit der geplanten Applikation in der Lage sein sowohl direkt mit der Brandmeldeanlage zu interagieren (aktuelle Statusinformationen, Bedienung, etc.) als auch seine Tätigkeit lückenlos zu dokumentieren (Fotos, Sprachkommentare). Nach Abschluss seiner Tätigkeit soll der Report direkt in interaktiver Form verfügbar sein.

Das Ziel ist es zum Abschluss der Studie eine lauffähige Applikation zu haben mit denen die wichtigsten Aspekte in Bezug auf UI und Workflow demonstriert werden können. Die Einbindung in bestehende Infrastruktur (Brandmeldesystem, IT Landschaft) spielt eine untergeordnete Rolle und kann auch simuliert werden.

Voraussetzungen:
Android, ggf. iPhone/iPad Entwicklung
Java, resp. Objective-C
Eclipse resp. XCode
Testautomation, Build-Server, etc., wie in SE Modulen gelernt
Netzprotokolle (HTTP, REST, etc)

Vorrang liegt bei Android (einfachere Umgebung, leichter Code auf Endgerät einspielbar) iPhone/iPad nur auf Wunsch d. Studenten.

Abstract

Siemens Brandmeldeanlagen sind komplexe Systeme, die nicht nur bei der Installation einen sehr hohen Arbeitsaufwand benötigen, sondern auch bei der Instandhaltung und bei jährlichen Revisionen viel Zeit in Anspruch nehmen. Aus diesem Grund versucht Siemens die dazu nötigen Arbeitsabläufe zu generalisieren und durch technische Hilfsmittel zu unterstützen. Eine komfortable Lösung zur Unterstützung eines Servicetechnikers bei der Anlagenrevision wäre ein mobiler Tablet-Computer mit Funkverbindung zur Brandmeldeanlage.

Die Arbeit soll eine getreue Abbildung der Anlagenstruktur auf dem Tablet umfassen, sowie eine stets präsente Kommunikationsmöglichkeit mit einer Brandmeldezentrale der Anlage darstellen. Um die Zentrale über HTTP zu erreichen steht ein Gateway zur Verfügung, das den Zugang zu dem BACnet-Protokoll (Building Automation and Control Networks) basierenden Anlagennetz gewährt. Funktionen wie das Auffangen und Weiterverarbeiten von Alarmereignissen, welche durch den Gateway nicht unterstützt werden, wurden für die Arbeit zu Demonstrationszwecken simuliert.

FireTablet ist eine auf Android basierende Applikation, die für das Samsung Galaxy Tab P1000 entwickelt wurde. FireTablet ermöglicht es, die Struktur einer Brandmeldeanlage über das Gateway zu laden und dazugehörige Jahresrevisionen zu verwalten. Einzelne Geräte oder ganze Meldezonen können zur Funktionsprüfung in einen Testmodus versetzt werden und von dem Techniker ausgelöste Testalarme durch FireTablet protokolliert werden. Wird an einem Gerät ein Mangel festgestellt, so kann dieser in Form von Bild, Ton und Text festgehalten werden. Tablet-unterstützt kann der Servicetechniker mit weniger Aufwand Anlagentests protokollieren und multimedial anreichern, was die Wartung von Siemens Brandmeldeanlagen sicherer, effizienter und strukturierter macht.

Management Summary

Ausgangslage

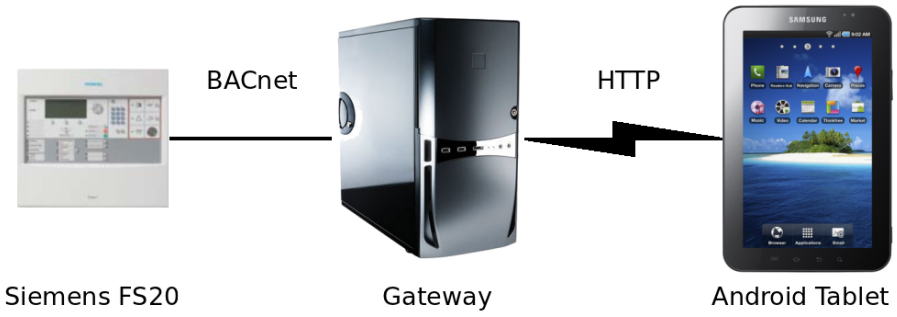
Moderne Brandmeldeanlagen schützen Personen und Gebäude, indem sie zuverlässig und frühzeitig Anzeichen von Bränden erkennen und Bewohner und Feuerwehr alarmieren, bevor es zu nennenswerten Schäden kommt. Damit sie dies garantiert erledigen können, ist eine regelmässige Kontrolle und Wartung nötig. Ein Servicetechniker untersucht eine Anlage und deren Zentralen in vielfältigen Aspekten auf Herz und Nieren. Unter anderem werden an die Zentrale angehängte Brandmelder mittels eines sogenannten Prüfpflückers auf Funktionstauglichkeit getestet. Das Gerät überprüft einerseits, ob der Sensor des Brandmelders einwandfrei funktioniert und sendet andererseits einen Probealarm an die Zentrale, um die Leitung zu testen.

Prüfpflücker zum Testen von Brandmeldern

Allerdings ist der bestehende Arbeitsprozess verbesserungsfähig: während der Techniker sich im Gebäude bewegt und Brandmelder testet, erhält er keine unmittelbare Rückmeldung von einer Zentrale, ob sein Test wirklich erfolgreich war. Erst im Ereignisprotokoll der Zentrale sieht er, ob der Melder korrekt ausgelöst hat. Ausserdem entsteht durch die manuelle Protokollierung auf Papier und elektronische Weiterverarbeitung der Testreihen ein Mehraufwand. Einerseits muss der Techniker das Logbuch für den Kunden nachführen, andererseits muss er nach einer Testreihe die Ergebnisse per Laptop protokollieren. Die so erfassten Daten schickt er zur Weiterverarbeitung an seine Zweigstelle, wo sie ins System übernommen werden. Konkret heisst das, dass ein Anlagentest im bestehenden Prozess dreifach bearbeitet werden muss.

Vorgehen

Es wurde eine Android-Applikation für Tablets entwickelt, die es ermöglicht, Informationen zur Anlagenstruktur von FS20 Brandmeldeanlagen zu laden, Anlagenrevisionen zu verwalten, während einer Revision auf Alarmereignisse zu hören und diese zu protokollieren. In ein Anlagennetz wird ein Gateway eingesetzt, der das zwischen Zentralen verwendete BACnet-Protokoll (Building Automation and Control Networks) versteht und den Nachrichteninhalt übersetzt. Das Tablet wiederum verbindet sich mittels WLAN mit dem Gateway, um diese Inhalte zu beziehen.
Funktionen wie das Auffangen und Weiterverarbeiten von Alarmereignissen, welche durch den zur Verfügung gestellten Gateway noch nicht unterstützt werden, wurden für die Arbeit zu Demonstrationszwecken simuliert. Dabei schickt ein separater Java-Server zufällig generierte Alarmereignisse an das Tablet.



Systemübersicht der beteiligten Geräte

Ergebnisse/Ausblick

Durch die Applikation kann der Servicetechniker sofort nach dem Auslösen eines Probealarms sehen, ob der Alarm korrekt der Zentrale gemeldet wurde und dies quittieren. Zur Funktionsprüfung können einzelne Geräte oder ganze Meldezonen mittels Tablet in einen Testmodus versetzt werden. Ist etwas nicht in Ordnung, kann dies pro Gerät in Form von Bild, Ton und Text festgehalten werden. Die Ergebnisse eines Anlagentests müssen nicht mehr von Hand protokolliert und dann elektronisch erfasst werden, sondern können direkt weiterverarbeitet werden.

Der Hauptbildschirm während einem laufenden Meldertest

Die Applikation demonstriert, was auf modernen Mobilgeräten realisierbar ist und wie der bestehende Arbeitsprozess benutzerfreundlich abgebildet werden kann. Sie ist sicher verbesserungsfähig, bildet aber auch einen Ausgangspunkt für vielfältige Weiterentwicklungen. Beispielsweise könnte nicht nur die Funktionsprüfung der Brandmelder, sondern auch die der Zentrale elektronisch abgebildet werden. Bestehende Formulare könnten durch die Applikation automatisch ausgefüllt und druckbereit gemacht werden. Der Export und das Weitersenden der erfassten Daten wurde im Rahmen der Arbeit rudimentär umgesetzt, dies auszubauen würde ebenfalls bedeutende Ressourcen sparen.

Inhaltsverzeichnis

1 Einführung

Autor: Daniel Bobst, Samuel Hüppi

In diesem Kapitel werden die Ausgangslage für die Arbeit und die darauf basierende Problemstellung erläutert. Es werden Ziele definiert, die Hardwareumgebung geschildert sowie der Aufbau dieser Dokumentation beschrieben.

1.1 Ausgangslage

Bei Siemens Brandmeldeanlagen handelt es sich um automatische Systeme, welche Brände automatisch erkennen, sowie alle nötigen Aktionen einleiten. Solche Aktionen könnten z.B. alle Lüftungen auszuschalten, Brandschutzklappen zuschliessen oder Lifte zu zu deaktivieren beinhalten. Brandmeldeanlagen sind also komplizierte Anlagen, die über ganze Gebäudekomplexe installiert und zentral verwaltet werden.

Sowohl Inbetriebnahme als auch Wartung solcher Anlagen sind aufwändige Prozesse und erfordern von den Servicetechnikern viel Konzentration und Zeit.
Viele Anforderungen an solche Brandmeldesysteme werden nicht direkt von Siemens erlassen, sondern sind von den jeweiligen Staaten und Normen vorgeschrieben. Die jährliche Überprüfung sämtlicher Richtlinien muss sehr gründlich vorgenommen werden, um erstens sicher zu stellen, dass die Anlage ordnungsgemäss funktioniert, und zweitens zur rechtlichen Absicherung, falls es zu Unfällen kommt. Der Ersteller solcher Anlagen ist somit sehr daran interessiert eine gute Dokumentation der Wartungs- und Inbetriebnahmearbeiten vorweisen zu können.
Um die Probleme um den bestehenden Arbeitsprozess zu verdeutlichen, wird im folgenden der Ist-Zustand genauer beschrieben.

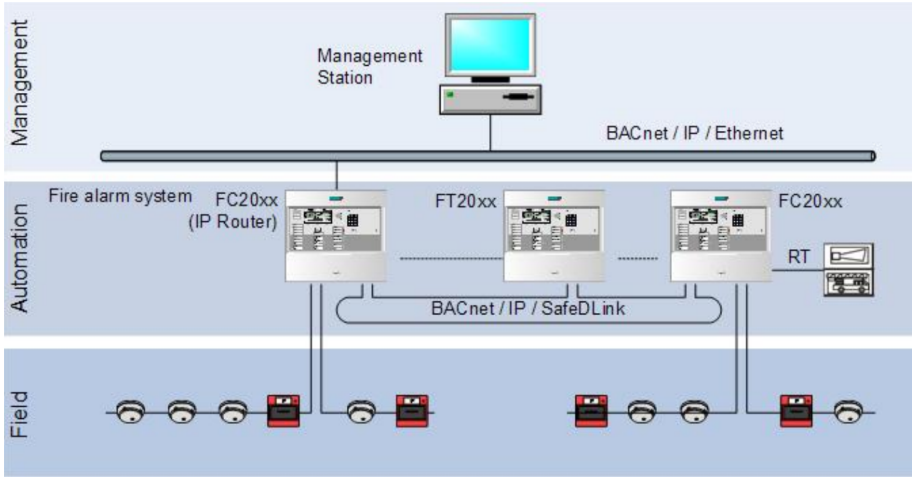


Abbildung 1.1: Hierarchische Struktur von Brandmeldeanalgen. Die Anlagen können über mehrere Gebäude verteilt sein.

1.2 Heutiger Arbeitsablauf

Die heutige Arbeitsweise sieht zwei verschiedene Prozesse vor: Kommissionierung und Wartung einer Anlage. Die Kommissionierung ist die Inbetriebnahme einer Brandmeldeanlage, und die Wartung ist eine Kontrolle, die jedes Jahr durch einen Siemens Servicetechniker durchgeführt wird. In beiden Fällen stehen dem Servicetechniker folgende Hilfsmittel zur Verfügung:

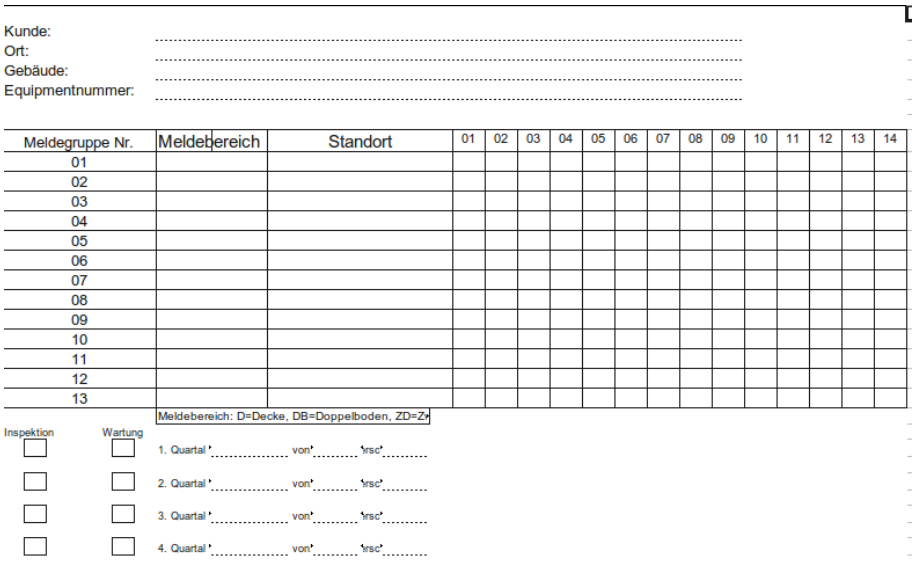
- **Panel**
  Eine Bedienschnittstelle, welche sich entweder direkt an der Zentrale oder an einer abgesetzten Bedieneinheit befindet. Ein Panel besitzt ein monochromes Display und ist mit diversen Tasten zu bedienen.

Abbildung 1.2: Bedieneinheit einer FS20 Brandmeldezentrale

- **Prüfplücker**
  Mit diesem Testgerät können Testalarme auf dem Brandmelder ausgelöst werden. Der spezielle Name kommt von der Art der Tätigkeit, wenn der Kontrolleur die Brandmelder, die normalerweise an der Decke hängen mit einer langen Stange prüft. Bei jedem Prüfvorgang wird der Pflücker an den Melder angedockt und löst automatisch den Test aus. Über drei Leuchtdioden wird dem Techniker angezeigt, ob der Test erfolgreich war oder nicht.

Abbildung 1.3: Prüfplücker um Brandmelder auf korrekte Funktion zu testen.

- **Checklisten**
  Damit der Arbeitsablauf die nötige Struktur hat und Resultate richtig notiert werden, bekommt der Servicetechniker eine ganze Auswahl an Listen, die im Verlauf einer Wartung oder Kommissionierung auszufüllen sind. Mit diesen Listen werden am Schluss die Prüfdokumente erstellt, sowie Rechnungen an den Kunden geschrieben. Werden Mängel festgestellt, muss der Monteur zusätzlich zu ersetzendes Material bestellen. Das Bild zeigt eine Liste, die bei Brandmeldertests zum Einsatz kommt. Wie unschwer zu erkennen ist, reicht der Platz für mehr Informationen als "ok" / "fehlerhaft" nicht aus.



Kunde: .....
Ort: .....
Gebäude: .....
Equipmentnummer: .....

Meldegruppe Nr.	Meldebereich	Standort	01	02	03	04	05	06	07	08	09	10	11	12	13	14
01																
02																
03																
04																
05																
06																
07																
08																
09																
10																
11																
12																
13																

Meldebereich: D=Decke, DB=Doppelboden, ZD=Z

Inspektion Wartung

1. Quartal '..... von'..... 'rsc'.....
2. Quartal '..... von'..... 'rsc'.....
3. Quartal '..... von'..... 'rsc'.....
4. Quartal '..... von'..... 'rsc'.....

Abbildung 1.4: Altes listenbasierendes Prüfprotokoll. Quelle: [Prüfprotokoll]

Bei Kommissionierung und Wartung sind folgende Arbeiten zu erledigen (Quelle: [Prüfprotokoll]):

- **Sichtprüfung**
  - Dokumentation und Betriebsbuch
  - Kontakte und Material in der Zentrale
  - Verschmutzung
  - Mindestabstand bei Melder
  - Akku Beschädigung
- **Funktionsprüfung**
  - Meldergruppen
  - Steuerungen der Anlage
  - Meldungen
- **Messung**
  - Akkuleistung
  - Netzversorgung
  - Spannung auf Meldergruppe

Im Unterschied zur Kommissionierung sind bei der Wartung nicht alle Brandmelder mit dem Prüfpflücker zu testen, sondern nur jeweils ein automatischer Melder pro Melderlinie. Handtaster müssen allerdings alle mit einem speziellen Schlüssel betätigt werden, damit nicht jedes mal die Scheibe eingedrückt wird, nur um einen Test durchzuführen.
Bei beiden Testarten ist der Kundentext, der am Panel angezeigt wird, sehr wichtig. Dieser Text zeigt der Feuerwehr an, an welchem Ort ein Melder ausgelöst wurde. Darum muss mindestens bei der Komissionierung überprüft werden, ob auch der richtige Kundentext zum richtigen Alarm angezeigt wird. Auf dem Bild sieht man, wie eine Alarmmeldung bei der Zentrale eintrifft. Hierbei wird der Melder, der einen Alarm ausgelöst hat, sowie die gesamte Hierarchie der Anlage, in welcher sich der Melder befindet, angezeigt.

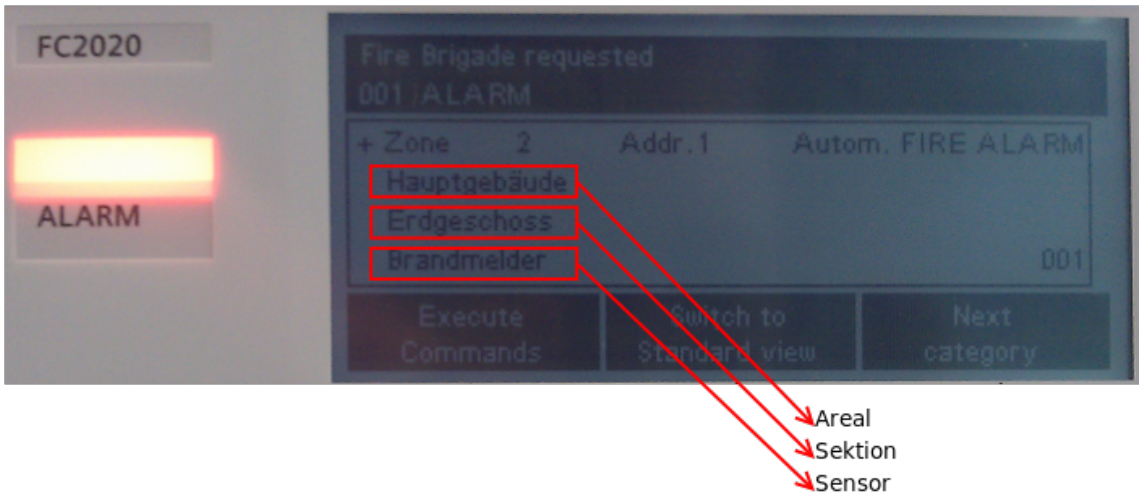


Abbildung 1.5: Anzeige eines Alarms auf dem Panel einer FC2020 Brandmeldeanlage.

1.3 Problemstellung

Die Wartung und Inbetriebnahme von Brandmeldeanlagen ist ein aufwändiger Prozess und muss sicherstellen, dass wirklich alles so funktioniert wie es der Test ergeben hat. Folgende Probleme oder Verbesserungen können durch neue Methoden und Geräte in Angriff genommen werden, damit die Überprüfung von Brandmeldeanlagen auch in Zukunft auf dem aktuellen Stand der Technik abläuft:

1. **Alle Mängel werden in Listen auf Papier eingetragen.** Die Listen müssen nach dem Test von Hand in Exceltabellen übertragen werden. Dieses System ist fehleranfällig, die Listen sind unübersichtlich, falls mehr Details eingetragen werden müssen und es entsteht unnötiger Zeitaufwand beim Übertragen der Resultate.
2. **Das Panel ist nicht sichtbar, wenn ein Test ausgelöst wird.** Es kann nicht sofort überprüft werden, ob Testalarme, die an Brandmeldern oder Handtastern ausgelöst werden, wirklich in der Zentrale mit korrektem Kundentext eintreffen, da man keine Verbindung zur Zentrale hat während man einen Testalarm auslöst. Erst wenn der Servicetechniker das Eventlog auf der Zentrale betrachtet, weiss er, ob die korrekten Sensoren mit der korrekten Standortbeschreibung ausgelöst wurden.
3. **Wenig Platz zur Mangelerfassung auf Papier.** Mängel können auf Excellisten nur sehr einfach erfasst werden oder es ist schwierig einen Mangel zu erfassen, wenn der Sensor an schlecht zugänglichen Stellen montiert ist.
4. **Keine zeitgemässen Schnittstellen für die Weiterverarbeitung.** Die Weiterverarbeitung von erfassten Tests ist zur Zeit sehr schwierig, da sie nicht elektronisch erfasst werden. Die Person, die einen Auftrag z.B. für eine Reparatur oder Erweiterung einer Anlage kalkuliert, kann sich kaum ein Bild der vorhandenen Lage machen.
5. **Unhandliches Arbeitsgerät Laptop.** Der zur Zeit verfügbare Laptop, den Servicemonteure bei sich haben, ist zu wenig mobil um ihn immer mit sich zu tragen und alle Testergebnisse direkt einzutragen. Gleichzeitig entsteht ein Diebstahlproblem, wenn der Laptop bei der Zentrale stehen gelassen wird, da sich Kommissionierungen oft noch auf der Baustelle abspielen.

1.4 Ziele

Um Verbesserungen in die Kontrollprozesse von Brandmeldeanlagen zu bringen, möchte Siemens Building Technologies Schweiz einen computergestützten Arbeitsablauf etablieren, der auch einen internationalen Standard bietet. Zum Einsatz soll ein Tabletcomputer kommen, welcher die Mobilität besitzt, ihn stets mit sich zu tragen, aber trotzdem eine komfortable Bedienung mit grossem Bildschirm ermöglicht. Sowohl in normalen als auch in speziellen Situationen, der Servicetechniker steht z.B. auf einer Leiter und hat nur eine Hand frei, ist eine gute Bedienung möglich. Auf dem Tablet soll eine Android Applikation laufen, die den Servicetechniker bei den Wartungsarbeiten unterstützt.
Damit die Brandmeldezentrale mit dem Tablet kommunizieren kann, stellt Siemens ein Gateway zur Verfügung. Das Gateway und das Tablet sind über WLAN miteinander

verbunden und kommunizieren über HTTP. Das Gateway tauscht über Ethernet mit der Brandmeldezentrale Daten aus und kommuniziert über das Gebäudeautomatisierungsprotokoll BACnet. Beim Design der Anwendung soll darauf geachtet werden die Bedienbarkeit auch in schwierigem Umfeld zu ermöglichen, indem z.B. grosse Buttons verwendet werden oder Fehler auch in Form von Voicememos gespeichert werden können.

1. **Zu allen Meldern oder Zonen Fehler erfassen.** Dies soll in Form von Bildern, Voicememo und Text möglich sein. Die Fehlerliste soll in digitaler Form verfügbar sein, damit sie bereit ist für die weitere elektronische Verarbeitung, Archivierung oder Protokollgenerierung. Bei allen Geräten kann nachgeschaut werden, welche Fehler dafür erfasst wurden und es ist ersichtlich, in welcher Zone wie viele Fehler vorhanden sind.

2. **Das Tablet empfängt Testalarme der Zentrale.** Das Gateway ermöglicht das Eintreffen von Testalarmen, die bei der Zentrale registriert werden. Dadurch wird es für den Servicetechniker möglich, gleich nach dem Auslösen eines Testalarms zu erfahren, ob der Kundentext korrekt ist, und ob die Brandmeldezentrale den Alarm mitgeschnitten hat. Sollte ein ausgelöster Testalarm nicht in der Brandmeldezentrale eintreffen, wird automatisch ein entsprechender Fehler für das Gerät erfasst.

3. **Konzept ermöglicht ständige Verfügbarkeit des Tablets.** Damit der Servicemonteur sein Tablet möglichst ohne Aufwand immer bei sich tragen kann, soll das Userinterface im Querformat erstellt werden, damit das Tablet an den Arm des Servicetechnikers montiert werden kann, um möglichst hohe Bewegungsfreiheit und Verfügbarkeit zu gewährleisten. Eventuell kann auch eine Halterung für das Tablet am Arm des Technikers befestigt werden.

4. **Das Tablet führt den Techniker durch den Test.** Das Tablet unterstützt den den Servicetechniker und hilft ihm dabei, die Tests Schritt für Schritt durchzuführen. Dies wäre dann ein Ersatz für die in der heutigen Zeit eingesetzten Checklisten.

1.5 Vorhandenes Umfeld

- Samsung Galaxy Tab P1000, Android 2.2, API Level 8
- REST BACnet Gateway 1.0
  Nicht alle nötigen Funktionen implementiert.
- FireTablet EventSimulator Server
- Brandmeldezentrale FS2020

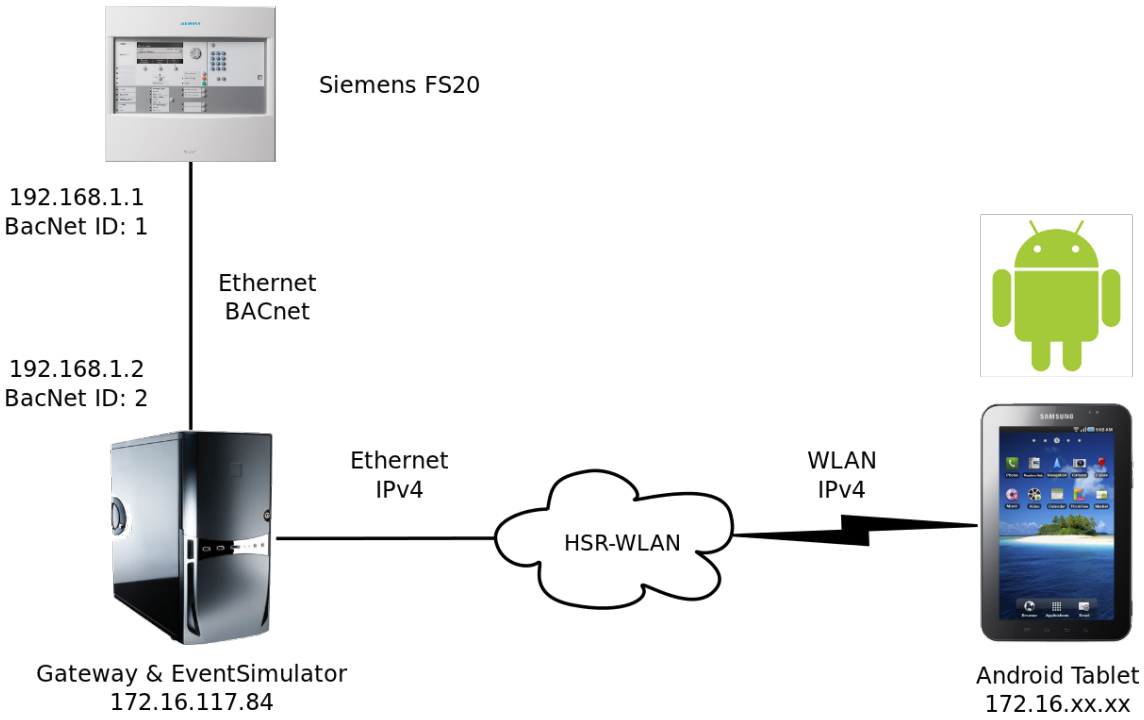


Abbildung 1.6: Architekturübersicht

Da das Gateway sich noch nicht bei einer Brandmeldezentrale für Eventnotifikationen registrieren kann, wird das BACnet Gateway nur verwendet, um die Anlagenstruktur von der Brandmeldezentrale zu laden. Events werden über den FireTablet EventSimulator Server simuliert, indem man sich mit dem Tablet beim EventSimulator für Events von bestimmten Meldern registrieren kann. Dies hat den Nachteil, dass nicht direkt am Brandmelder mit dem Prüfpflücker Events ausgelöst werden können, reicht aber für Demonstrationen des Tablets.

1.6 Verwendungszweck der Software

Die oben beschriebenen Ziele der Software sollen erreicht werden und sind für die Entwicklung der Software sehr wichtig. Es ist klar, dass innert 16 Wochen mit zwei Entwicklern und einem Teilzeitpensum keine produktionsfähige Applikation entsteht.
Unsere Applikation soll aber aufzeigen, was auf der Androidplattform möglich ist und könnte ein Grundstein sein für zukünftige Entwicklungen. Weiter soll die Software auch zeigen, welche Services Siemens Brandmeldeanlagen bieten und in welche Richtung die Entwicklung denkbar ist.

1.7 Aufbau der Dokumentation

Im folgenden wird der Aufbau dieser Dokumentation erklärt. Die Struktur der Dokumentation kann in vier Bereiche aufgeteilt werden, die hier kurz erläutert und kommentiert werden.

- **Einführung**
  Im Bereich 1 Einführung wird eine kurze Übersicht zum Thema vermittelt und die Probleme, Ziele und Voraussetzungen werden erläutert.

- **Analyse**
  In diesem Block befinden sich die Kapitel 2 (Technische Rahmenbedingungen) und 3 (Anforderungen). Ersteres beschäftigt sich mit dem Aufbau der eingesetzten Infrastruktur und den Erkenntnissen, welche durch die Analyse für die Software gewonnen wurden. Da physikalisch gesehen alles schon vorgegeben war, beschränkt sich die physikalische Architektur auf den Bereich Analyse. Beim zweiten Kapitel wird analysiert welche Anforderungen von Kundenseite an die Applikation gestellt werden. Daraus resultierten anschliessend die Use Cases, nach denen FireTablet entwickelt wurde.

- **Design & Architektur**
  Nachdem die Rahmenbedingungen klar abgesteckt wurden, wird in den nächsten zwei Kapiteln 4 (Interface Design (Autor: Daniel Bobst)) und 5 (Software Entwicklung) nur noch auf die Architektur und das Design der FireTablet Software eingegangen. Sämtliche durch den Analysebereich abgedeckte Erkenntnisse werden als gegeben betrachtet.

- **Schluss**
  Zu guter Letzt wird im Kapitel 6 noch einmal ein Blick zurück gewagt und das Resultat kritisch mit den Anforderungen verglichen. Welche Ziele konnten erreicht werden und mit welcher Qualität wurden sie erreicht? Allerdings wollen wir an dieser Stelle nicht nur zurück, sondern auch nach vorne blicken und uns fragen, was man alles noch implementieren könnte.

1.8 Erläuterung zur Arbeit

Die beiden Studenten Daniel Bobst und Samuel Hüppi arbeiten gemeinsam an dieser Arbeit. Allerdings soll die Abgabe nicht gemeinsam erfolgen, sondern getrennt. Der Grund für diesen Umstand ist der unterschiedliche Kontext der Arbeiten. Daniel Bobst arbeitet im Rahmen seiner Bachelor-Arbeit(BA) an diesem Projekt mit und Samuel Hüppi absolviert seine Semester-Arbeit(SA). Dadurch entsteht ein unterschiedlicher wöchentlicher Zeitaufwand für jeden Student und die Bedingungen sind nicht die gleichen, wie in folgender Tabelle zu entnehmen ist. Trotzdem wird im Folgenden von den zwei Arbeiten in Einzahl gesprochen.

	ECTS Punkte	Stunden/Punkt	Dauer	Stunden/Woche
SA	8	30h	14 Wochen	ca. 18h
BA	12	30h	16 Wochen	ca. 23h

Tabelle 1.1: Übersicht Arbeitsaufwand und Punkte

1.9 Betreuung und Partner

Die Arbeit wird von Herrn Prof. Peter Sommerlad, Partner am Institut für Software IFS an der Hochschule Rapperswil, betreut. Auf der Seite von Siemens stehen folgende Personen für Fragen und Informationen zur Verfügung:

Person	Bereich
Martin Botzler	Head of Architecture & Platforms
Dirk Stockmann	Head of CTO
Volker Redwitz	Innovation Manager Fire Safety Systems & Products
Philippe Götz	Architecture & Platform, Senior Developer BACnet
Reinhard Bauer	Fire Systems, Senior Developer BACnet

Tabelle 1.2: Ansprechpersonen von Siemens

2 Technische Rahmenbedingungen

Im Rahmen dieses Kapitels wird die Ausgangslage für die Arbeit analysiert. Einerseits befassen wir uns mit dem Kommunikationsprotokoll BACnet, andererseits untersuchen wir die Fähigkeiten des Gateways. Schliesslich wird das Android-Framework vorgestellt.

2.1 Analyse BACnet Protokoll

Autor: Daniel Bobst

Um die vom Gateway angebotenen Informationen zu verstehen, analysieren wir in diesem Kapitel den Aufbau einer FS20 Brandmeldeanlage und welche Nachrichten durch verschiedene BACnet-Objekte verschickt und empfangen werden.

2.1.1 Was ist BACnet?

BACnet (Building Automation and Control Network) ist ein in der Gebäudeautomation weit verbreiteter Kommunikationsprotokoll-Standard. Es wird zur Überwachung und Kontrolle in diversen Bereichen angewendet, zum Beispiel Heizung, Lüftung, Klimaanlagen, Licht, Zugangskontrolle und Brandmeldesysteme. Der BACnet Standard definiert die Kommunikation zwischen beliebigen BACnet-kompatiblen Geräten. Als physisches Übertragungsmedium werden eine Reihe von Trägern unterstützt, unter anderem Ethernet.

2.1.2 Struktur Brandmeldeanlage

Eine Brandmeldeanlage wird als Domänen mit verschiedenen Wissensständen und Verantwortlichkeiten modelliert. Jede Domäne ist hierarchisch aufgebaut. Die drei Domänen sind Detection, Control und Physical. [FS20 BACnet Spec, S.15]

Domäne	Aufgabe
Detection	Geographische und logische Struktur, Alarmauswertung
Control	Alarmierungs- und Evakuierungsausrüstung, andere Kontrollfunktionen
Physical	Hardwarekomponenten, physische Struktur

Tabelle 2.1: Domänen einer Siemens Brandmeldeanlage

Die Detection Domain umfasst Geräte wie Rauchmelder und Handtaster. In der Control Domain befinden sich Geräte wie Alarmsirenen, Evakuierungssignalisationen, Benachrichtigungseinheiten sowie weitere Gebäudefunktionen, die in einem Brandfall aktiviert werden müssen. Durch eine Benachrichtigungseinheit wird falls nötig zum Beispiel die Feuerwehr aufgeboten. Weitere Gebäudefunktionen kann beispielsweise bedeuten, dass Brandschutztüren geschlossen, Lüftungen ausgeschaltet und Aufzüge ins Erdgeschoss gefahren werden.

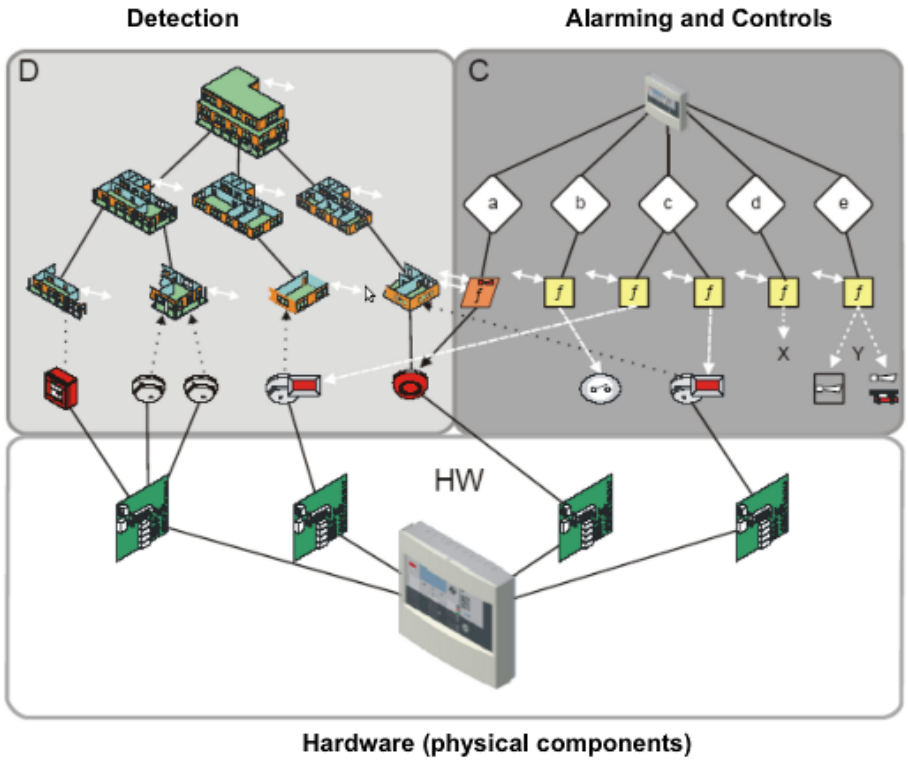


Abbildung 2.1: Grafik der Domänen einer Siemens Brandmeldeanlage

Eine Anlage (oder "Site") besteht aus einer oder mehreren miteinander verbundenen Zentralen ("Panels"). Jede Zentrale ist in der Physical Domain vertreten und hat Zugriff auf die Detection und Control Domains, die aus den an ihr angehängten Geräten gebildet werden.

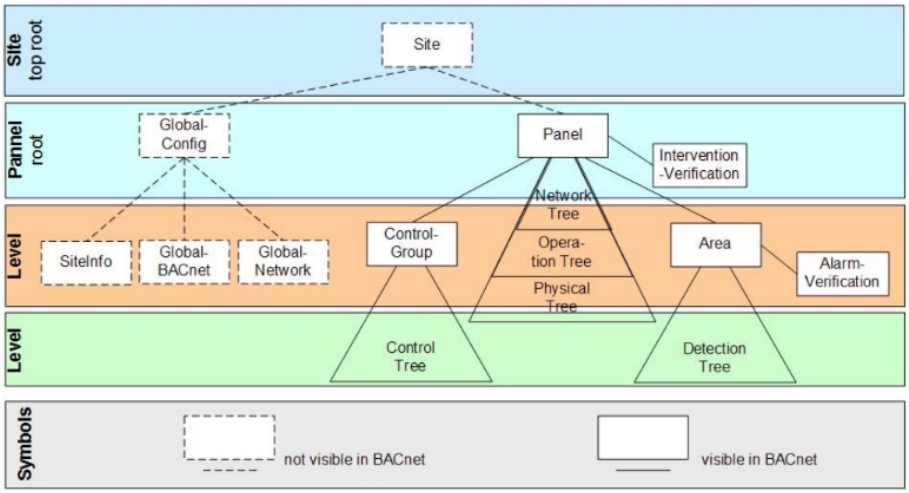


Abbildung 2.2: Logischer Aufbau einer Siemens Brandmeldeanlage.
Quelle: [FS20 BACnet Spec]

Betrachten wir die für uns am relevantesten Domäne, die Detection Domain, etwas genauer.

2.1.2.1 Detection Domain

Die Detection Domain beinhaltet alle Objekte, die Alarme auslösen und auswerten können. Die Domäne kann hierarchisch weiter aufgeteilt werden in Panel, Area, Section, Zone und Channels

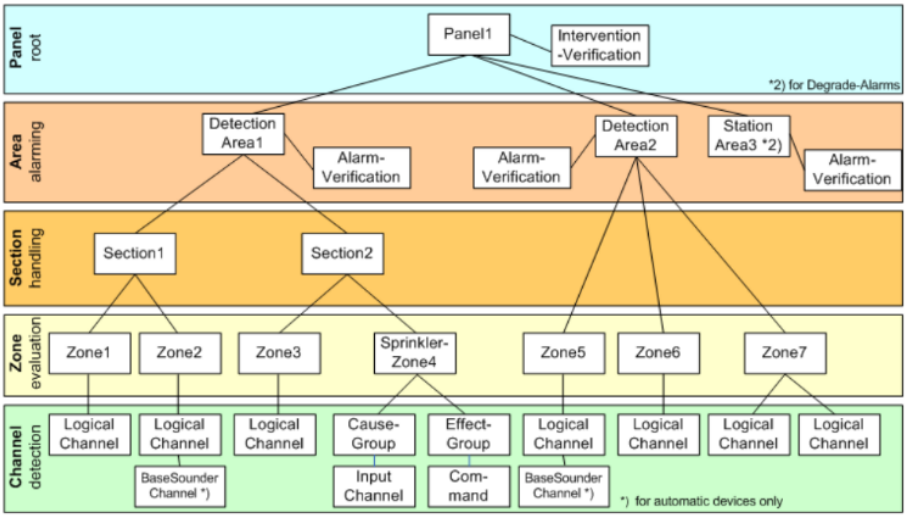


Abbildung 2.3: Baum der Detection Domain einer Siemens Brandmeldeanlage. Quelle: [FS20 BACnet Spec]

- Panel: Ein Panel entspricht einer Zentrale oder einem Terminal. Ein Terminal ist eine einfache Bedieneinheit mit einem Bildschirm und wenigen Knöpfen, mit dem eine Zentrale von einem entfernten Ort bedient werden kann.

Abbildung 2.4: Ein Bedien-Terminal einer Brandmeldeanlage

- Area: Die Area entspricht üblicherweise einem Gebäude oder einem Gebäudeteil. Einstellungen, die auf eine Area angewendet werden, gelten auch für die untergeordneten Einheiten. So können allgemein gültige Parameter an einem einzigen Ort gesetzt werden, statt dass bei jedem Objekt der Area eine Eigenschaft geändert werden muss.
- Section: Eine optionale Einheit, mit der beliebige Zonen zusammengefasst werden können. So können zum Beispiel komfortabel mehrere Zonen miteinander ein- oder ausgeschaltet werden. Eine Sektion umfasst typischerweise ein ganzes Geschoss oder ein Treppenhaus.
- Zone: Typischerweise bildet ein Raum eine Zone. Eine Zone fasst die Meldungen der in ihr enthaltenen Brandmelder zusammen und analysiert sie gemäss ihreer Einstellungen, ob ein Alarm ausgelöst werden muss. Eine Zone kann in verschiedenen Modi betrieben werden: Ein, Aus, Renovation, Detektortest, Installationstest.
- Channel: Repräsentiert einzelne Geräte wie automatische Melder, Handtaster oder andere Geräte als logische Kanäle. Diese logische Kanäle sind intern mit den entsprechenden physischen Kanälen verbunden. Bei automatischen Meldern ist immer auch ein logischer Kanal angehängt, der den eingebauten Signalgeber darstellt.

In der Grafik sind ausserdem die Elemente Intervention und Alarm Verification sichtbar.

- Intervention Verification: behandelt die Eskalation und Verzögerung von Fehlermeldungen und anderen Systemzuständen
- Alarm Verification: behandelt die Eskalation und Verzögerung von Alarmen

2.1.3 BACnet Objekte

BACnet unterstützt eine Vielzahl von Objekten, um eine beliebige Gebäudeanlage zu modellieren. Bei der FS20 Brandmeldeanlage werden die Objekte "Device", "Notification Class" sowie "Life Safety Zone" benutzt.

BACnet Property Identifier	Nr.	BACnet Property Type	Impl	Default values
Object_Identifier	75	BACnetObjectIdentifier	RO	<fs20-notificationclass-object-identifier>
Object_Name	77	CharacterString	RO	<fs20-notification-class-object-name>
Object_Type	79	BACnetObjectType	RO	NOTIFICATION-CLASS
Description	28	CharacterString	RO	<customer text>
Notification_Class	17	Unsigned	RO	<object instance number>
Priority	86	BACnetARRAY[3] of Unsigned	RO	[3, 3, 10]
Ack_Required	1	BACnetEventTransition	RO	TRUE, TRUE, FALSE
Recipient_List	102	List of BACnetDestination	RW	<empty>
Profile_Name	168	CharacterString	RO	7-FI-FS20-NotificationClass-1

Tabelle 2.2: BACnet Notification Class Object
(RO: ReadOnly, RW: Read/Write), Quelle: [FS20 BACnet Spec]

2.2 Analyse Gateway

Autor: Samuel Hüppi

2.2.1 Ziel

Der Gateway für die Siemens FS20 stellt die Verbindung zwischen FS20 und dem normalen IP-Netz her. Dies ist nötig, weil die Kommunikation der FS20 über BACnet läuft. BACnet wäre zwar IP fähig, aber es benötigt zusätzliche Komponenten (Broadcast BACnet Management Devices) für die Kommunikation über zwei IP-Subnetze. Diese Komponenten leiten die Broadcastmeldungen der BACnet-Geräte ins nächste Subnet weiter, damit sich die BACnet-Teilnehmer gegenseitig auch in verschiedenen Subnetzen finden. Aus diesem Grund kann BACnet nicht über ein LAN verwendet werden und es wird ein Gateway zwischengeschaltet. Dieser wandelt die in HTTP ankommenden Befehle in BACnet-Befehle und umgekehrt von BACnet in HTTP um. Zusätzlich vereinfacht er die Komplexität von BACnet.
Der Gateway ist eine in C geschriebene Applikation, die von Siemens entwickelt wurde und zur Verfügung gestellt wird. In diesem Abschnitt werden auf die technischen Eigenschaften, sowie die Installation des Gateways eingegangen. Während der Analyse des Gateways hat sich zusätzlich gezeigt, dass wichtige Funktionen, um sich für Events zu registrieren, noch nicht auf dem Gateway implementiert sind. Als Ersatz kommt ein Event Simulationsserver zum Einsatz. Am Ende des Abschnitts Analyse Gateway wird auf die Technik des Simulationsservers eingegangen.

2.2.2 Schnittstelle

Verbindungen	Protokolle
Gateway zu FS20	Ethernet / IP / BACnet
Gateway zu Tablet	Ethernet / IP / TCP / HTTP / XML

Tabelle 2.3: Schnittstellen zwischen FS20 - Gateway - Tablet

2.2.3 Aufbau Testumgebung

Der Testaufbau im Labor wurde gemäss Bild erstellt. Dabei ist es wichtig zu beachten, dass zwischen dem Gateway und der FS20 keine aktiven Komponenten sind, da das IP Protokoll, auf welchem BACnet läuft, nicht kompatibel mit Routern ist. Das kommt daher, weil alle BACnet Geräte sich über Broadcast kennenlernen. Die Broadcast Pakete würden also ein Subnetz nie verlassen und zwei BACnet-Geräte in verschiedenen Subnetzen, würden sich nie kennen lernen. Die FS20 sowie der Gateway haben statische IP Adressen. Sie können so einfacher erreicht werden. Über WLAN werden dem Tablet dynamisch IP-Adressen verteilt.

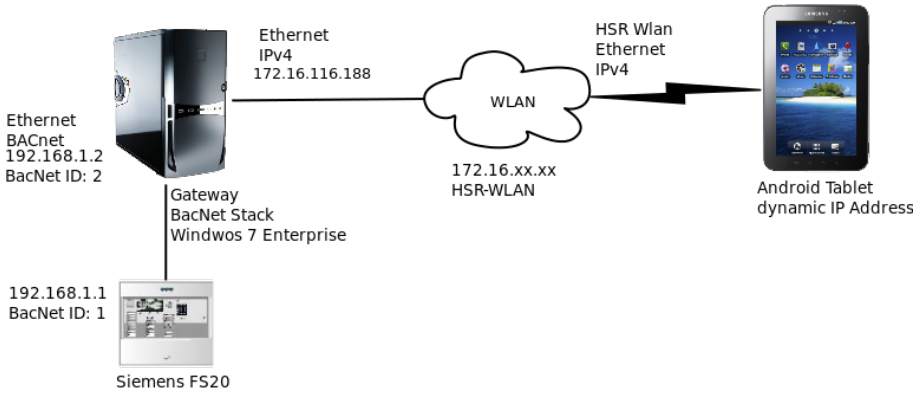


Abbildung 2.5: Testumgebung Verbindung zum Gateway

2.2.4 Möglicher Aufbau Realität

In der Realität gibt es verschiedene Möglichkeiten, wie man das Tablet mit der FS20 verbinden könnte. Am sinnvollsten scheint es das Gateway beim Kunden zu platzieren und das Gateway über Internet anzusprechen. Der Vorteil dieser Lösung wäre die grosse Verfügbarkeit von GSM. Man hätte in fast jedem Gebäude Empfang und ist nicht auf ein vorhandenes WLAN angewiesen. Der Nachteil ist ein sehr langer Weg von der Zentrale zum Rechenzentrum und wieder zurück zum Tablet. Zudem könnte ein Sicherheitsproblem entstehen, wenn der Gateway vom Internet her erreichbar sein soll.

2.2.5 Installation des Gateway

Im folgenden wird die Installation des Gateways beschrieben damit das vorhandene System jederzeit nachgebaut werden kann.

BACnet Stack

Als erstes muss auf dem Rechner, auf dem das Gateway laufen soll, der BACnet Stack installiert werden. Der BACnet Stack umfasst den Application Layer, Network Layer, sowie den Media Access Layer (MAC) und ermöglicht BACnet Kommunikation über ein normales Ethernet. Der BACnet Stack ist eine Open Source Library und unter Linux, Windows und anderen Betriebsystemen verfügbar. [BACnet Stack] Bei der Installation sollte beachtet werden, dass das richtige Netzwerkinterface ausgewählt wird, welches mit der FS20 kommuniziert.

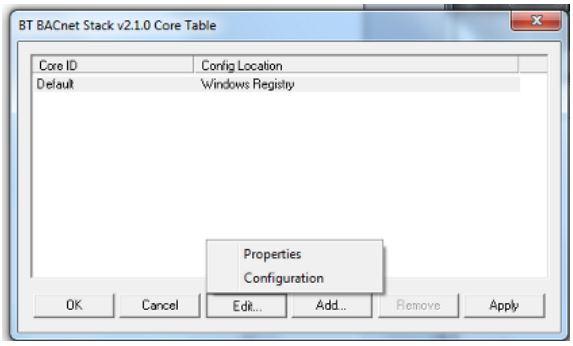


Abbildung 2.6: Dialog während der Installation, um die Konfiguration zu ändern. Den Eintrag Default auswählen und danach auf Edit klicken. Im Kontextmenü "Configuration" auswählen.

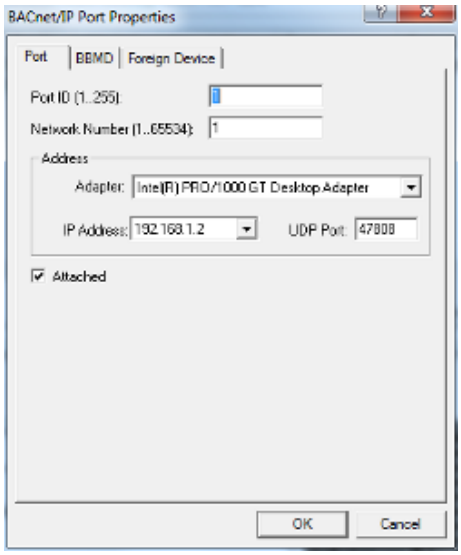


Abbildung 2.7: Port zur FS20 konfigurieren. Wichtig ist das richtige Netzwerkinterface, sowie IP-Adresse. Der UDP Port kann auf dem Standartwert belassen werden.

Gateway

Die Konfiguration des Gateways geschieht mittels der apphttp.txt Datei. Folgende Parameter müssen dabei definiert werden.

Parameter	Beschreibung
BBSConnectionString	Der Name der Verbindung
PortID	BACnet ID der FS20. Muss mit Konfiguration auf FS20 übereinstimmen
DeviceID	BACnet ID des Gateway. Muss mit Konfiguration auf FS20 übereinstimmen
DeviceName	Der Name des Gateway, unter dem er in BACnet sichtbar ist.
WebBinding	IP-Adresse auf die der Gateway hört. Am besten die Wildcard 0.0.0.0:80 verwenden.

Tabelle 2.4: Tabelle mit den Parametern zum BACnet Gateway

```
# Parameters
$(BBSConnectionString)?="\\.\pipe\BACstacDefault"
# Used for BBS Multicore 2.1.X on the core "Default"
$(PortID)?=1 # Used PortID in the previous core
$(DeviceID)?=2 # BACnet Device ID of the BFC REST Gateway
$(DeviceName)?="BFC REST Gateway"
$(WebBinding)?="127.0.0.1:80" #
VERSION END$
```

Listing 2.1: Attribute der Konfigurationsdatei für den BACnet Gateway

Das Gateway wird mittels apphttp.bat gestartet. Die Batchdatei konfiguriert den BACnet Stack damit das Gateway darauf arbeiten kann. Sobald das Gateway sich im Infinite Loop befindet, können Befehle zum Gateway abgesetzt werden.

2.2.6 Wichtige Funktionen und Werte

Die Funktionen sind als Http-Befehle nach folgendem Schema zu verstehen:

```
[Protokoll]://[IP-Adresse Gateway]:[UDP Port]/[PortID]/[Befehl]
```

Listing 2.2: Schema der Http-Befehle

In unserem Fall bezeichnet man mit der PortID 1 die FS20, da sie im BACnet die ID 1 besitzt, welche durch die Gateway-Konfigurationsdatei zugewiesen wurde.

Funktion	Beschreibung
http://Address/1/.search	Alle Elemente von Device 1 werden in einer Liste zurück geschickt.
http://Address/1/life-safety-zone,X	Alle Elemente der Life-Safety-Zone X werden angezeigt

Tabelle 2.5: Wichtige Http-Befehle für den BACnet Gateway

Die Antwort des Gateways ist in XML formatiert und kann vom Empfänger geparsed werden. Alle Objekte auf dem Gateway sind in sogenannten Life-Safety-Zones gespeichert. Mit dem .search Befehl werden zuerst alle vorhandenen Life-Safety-Zones abgerufen. Danach wird jede Life-Safety-Zone einzeln geparst und als Device weiter im Programm verwendet.

```
<Object xmlns="http://www.BACnet.org/CSML/1.1">
<ObjectIdentifier name="object-identifier"
         value="life-safety-zone,16"/>
<String name="object-name" charset="5"
         value="LZ_AREA_DETECTION_1_1/16"/>
<Enumerated name="object-type" value="life-safety-zone"/>
<Enumerated name="present-value" value="quiet"/>
<BitString name="status-flags">
<Value/>
</BitString>
<Enumerated name="event-state" value="normal"/>
<Enumerated name="reliability" value="no-fault-detected"/>
<Boolean name="out-of-service" value="false"/>
<Enumerated name="mode" value="unmanned"/>
<List name="accepted-modes">
```

Listing 2.3: Ausschnitt einer XML Antwort des Gateways auf den Befehl /life-safety-zone.16

Die folgende Tabelle zeigt alle verwendeten Objekte und erklärt ihre Bedeutung:

Name	Beschreibung
object-identifier	Einmalige Identifikation jeder Life-Safety-Zone.
object-name	Name der Life-Safety-Zone.
description	Kundentext einer Zone, in welcher z.B. beschrieben wird, in welchem Raum sich ein Gerät befindet.
device-type	Zonentyp. In FireTablet werden AreaElem, ZoneElem, SectionElem und SensorElemente verwendet.
notification-class	Meldungsklasse bei welcher sich die Zonen melden. Gleichzeitig kann sich ein Observer bei einer Meldungsklasse registrieren um die Events aus dieser Klasse zu empfangen
notify-type	Die Art des Events der ausgelöst werden kann.
maintenance-required	Dieser Wert wird zu "true"wenn die Life-Safety-Zone gewartet werden sollte.
isa-event-message-texts	Letzte geloggte Aktion die auf der Zone ausgeführt wurde.
profile-name	Profilname der Zone
member-of	Liste von Object Identifiers die zur aktuellen Zone Elternelemente sind.
zone-members	Liste von Object Identifiers die Kindelemente der aktuellen Zone sind.

Tabelle 2.6: Bedeutung der Attribute einer Life-Safety-Zone

2.2.7 Event Simulationsserver

Autor: Daniel Bobst

Wie bereits erwähnt fehlen auf dem Gateway gewisse Funktionen um alle Use Cases nur über das Gateway abzuwickeln. Man kann sich auf dem Gateway nicht für Events auf den Notification Classes (s. Kap. 2.1.3) registrieren. Daher ist es unmöglich über Testalarme, die bei der Zentrale eintreffen, informiert zu werden. Eine Lösung wäre gewesen, das Gateway um Funktionen zu erweitern, so dass es auf Events hört und diese dem Tablet weiterleitet. Allerdings reichte die Zeit nicht für die nötige Analyse des Quellcodes und entsprechende Implementation der neuen Funktionalität. Aus diesem Grund haben wir uns für die Simulation dieser Events entschieden.

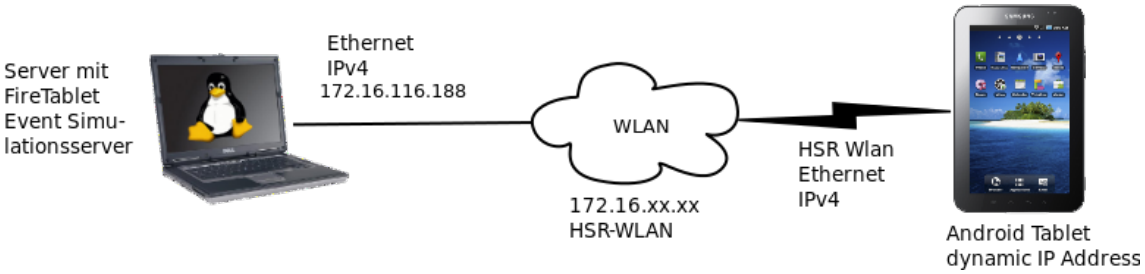


Abbildung 2.8: Event Simulationsserver in Verbindung mit dem Tablet.

Dazu wurde ein simpler Server in Java realisiert, der parallel zum Siemens-Gateway läuft und stets auf Verbindungen wartet.

Wird vom Benutzer der Testmodus gestartet, öffnet das Tablet (Client) öffnet einen Socket zum Server und schickt ihm einen String der Form

```
SUB;<port>;<object-identifier1>;<object-identifier2>;...
```

Listing 2.4: Subscription von Objekten für Eventgenerierung auf dem Server

Danach öffnet der Client auf dem genannten Port einen ServerSocket, um auf ankommende Nachrichten zu hören.
Der Server kann entweder in zufälligen Abständen einen der mitgegebenen Object-identifier auswählen und für diesen ein Ereignis zurückschicken oder auf einen Knopfdruck warten, um dies zu tun. Pro angemeldetem Object-identifier wird nur eine Nachricht zurückgeschickt. Die Nachricht besteht nur aus dem Object-identifier, was in unserem Prototyp ausreicht, um einen Event zu symbolisieren. Ein echtes BACnet Ereignis würde in Form einer Notification Class in XML zurückgeschickt werden. Der Server öffnet seinerseits einen Socket zum Client und schickt auf diesem die Nachricht.
Empfängt der Client auf seinem ServerSocket eine Nachricht, parst er diese als Object-identifier und löst für diesen einen Warndialog aus (s.Kap. 4.3.3).

Stoppt der Benutzer den Testmodus, schliesst der Client seinen ServerSocket und sendet den String `UNSUB` an den Server. Empfängt dieser solch einen String, löscht er alle noch nicht gemeldeten Object-identifier und macht sich wieder bereit, um auf den nächsten Subscribe-Befehl zu warten.

Das folgende Sequenzdiagramm zeigt den Start eines Testmodus vom Bildschirm "FacilityInfo" aus (s.a. Kap. 4.3.3). Unter anderem ist die Kommunikation zwischen dem Client (besteht aus *CheckEventsThread* und *ListeningThread*) und Server (Akteur rechts) abgebildet.

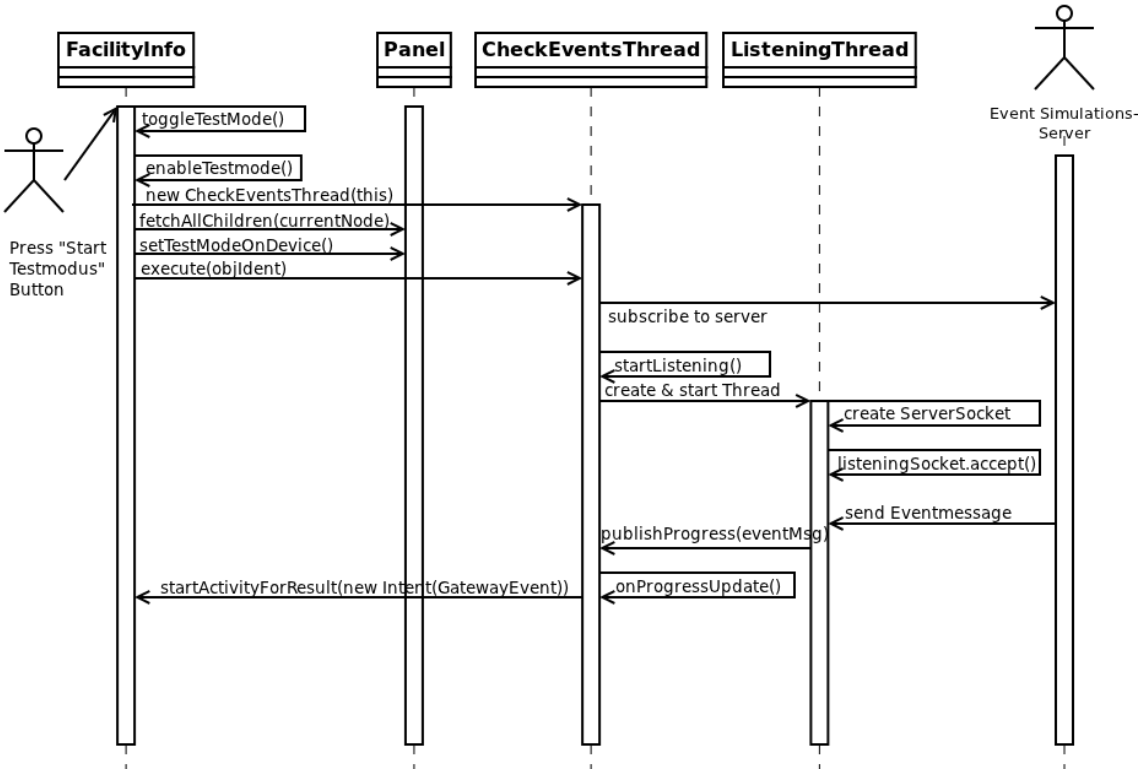


Abbildung 2.9: Ablauf nachdem der Testmodus gestartet wurde.

2.3 Android

Autor: Daniel Bobst, Samuel Hüppi

Da Android als Basis für FireTablet dient, wird in diesem Bereich kurz auf Android eingegangen. Der Schwerpunkt liegt auf ein paar ausgewählten Androidmechanismen, die über die gesamte Applikation immer wieder vorkommen. Ausserdem werden häufig gebrauchte Begriffe erläutert.

2.3.1 Mobile Plattform Android

Abbildung 2.10: Android Maskottchen

Android war ursprünglich ein Betriebssystem für Mobiltelefone. [Wikipedia Android] Im Verlauf der Zeit wurde Android aber für weitere Geräte portiert, wie zum Beispiel Tabletcomputer oder Fernseher. Android basiert auf dem Linux Kernel 2.6 und verwendet die Speicherverwaltung, Prozessverwaltung und die Netzwerkkommunikation von Linux. Gleichzeitig dient der Kernel auch als Hardwareabstraktionsschicht.
Applikationen, die für Android geschrieben werden, laufen im Normalfall auf einer Virtuelle Maschine (VM) namens Dalvik. Das bringt den Vorteil, dass jedes Programm in einer eigenen Instanz dieser VM ablaufen kann, und somit eine hohe Sicherheit und Stabilität entsteht. Dies nennt man das Sandboxprinzip. [Android 2, S. 27] Die Architektur von Android ist im Bild unten dargestellt.

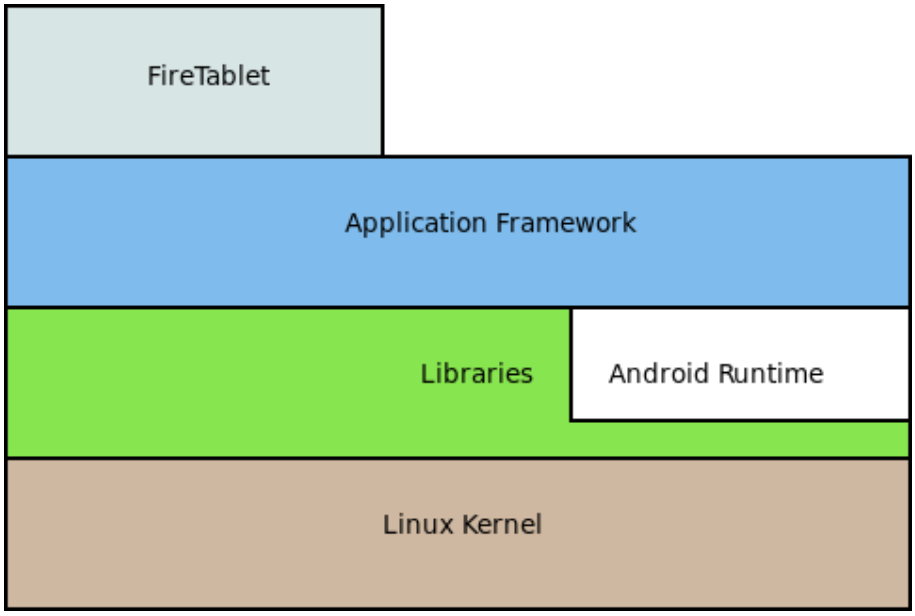


Abbildung 2.11: Aufbau von Android

Die folgenden Abschnitte befassen sich mit wichtigen Mechanismen von Android.

2.3.2 Activities und Intents

Jede Androidapplikation besteht aus einer beliebigen Menge von Activities.
[Activity Reference] Unter einer Activity versteht man in den meisten Fällen einen Bildschirm auf dem Display. Die Activity ist verantwortlich für die Darstellung der einzelnen Gui-Komponenten, aber auch für die Entgegennahme von Benutzereingaben. Aus diesem Grund hat eine Activity meistens einiges mehr an Verantwortung als reine Gui-Klassen. So muss zum Beispiel jede Activity ihren Lebenszyklus selber verwalten, wozu es aber geeignete Hookmethoden gibt. Im folgenden Bild wird der Lebenszyklus von Activities gezeigt. Am interessantesten sind die Methoden *onCreate()* und *onPause()*. Mit *onCreate()* wird die Activity initialisiert, da diese Methode bei jedem Start aufgerufen wird und eigentlich als Konstruktor gilt. OnPause() wird genau im umgekehrten Fall aufgerufen. Nämlich dann, wenn eine Activity in den Hintergrund tritt, sei es weil sie beendet wurde, oder weil eine neue Activity gestartet wird.

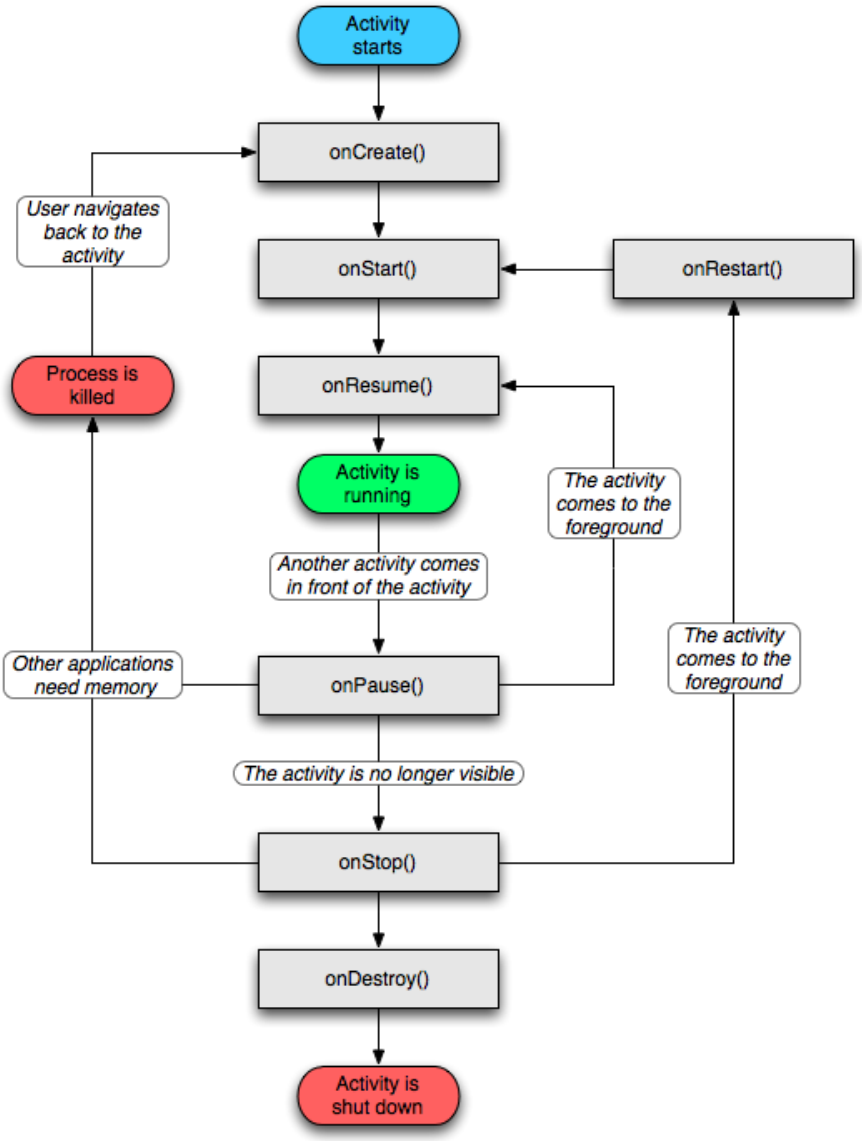


Abbildung 2.12: Lebenszyklus einer Activity. Quelle: [Activity Reference]

Um von einer Activity zur nächsten zu kommen, werden Intents eingesetzt. Dadurch können Activity sehr lose verknüpft werden und das Betriebssystem hat immer die Kontrolle darüber, welche Activities gestartet werden. Mit einem Intent übergibt man eigentlich dem Betriebssystem den Klassennamen der Activity, die gestartet werden soll. Das Betriebssystem startet diese anschliessend. Intents bieten auch die Möglichkeit primitive Datentypen mitzuschicken, um diese zwischen Activities Daten auszutauschen. Intents werden, wie im Listing unten, erstellt und dem Betriebssystem übergeben. [Android 2, S. 137]

```
Intent facilityTree = new Intent(this, LoadFacilityTree.class);
startActivity(facilityTree);
```

Listing 2.5: Intent erstellen und abschicken

2.3.3 Kontexte

Die Contextklasse ist eine dauerhaft präsente Klasse innerhalb des Androidframeworks. Jede Activity leitet vom *Context* ab, denn mit einem *Context* wird der Bezug der Applikation zum Androidsystem hergestellt. Folgende wichtigen Ressourcen macht ein *Context* gemäss Android 2 zugänglich: [Android 2, S. 26]

- Classloader
- Dateisystem
- Berechtigungen der Anwendung
- Datenbanken
- Anwendungseigene Resourcen
- etc.

Der *ApplicationContext* ist eine Spezialform von *Context*. Dieser *Context* wird einmalig pro Applikation von Android gestartet und während der ganzen Laufzeit des Programms am Leben gehalten. Dadurch können in den *ApplicationContext* Dinge gespeichert werden, die man während der gesamten Laufzeit immer wieder benötigt. Es ist möglich einen eigenen *ApplicationContext* zu erstellen, indem von der *Application* Klasse abgeleitet wird und diese Klasse im Android Manifest vermerkt wird. [Android Manifest Reference] Das Android Manifest ist im Rootverzeichnis jedes Androidprojekts und beinhaltet alle nötigen Informationen, die eine Applikation benötigt. Dazu gehören Berechtigungen für Systemresourcen sowie die Definitionen aller Activities.

```
<uses-permission android:name="android.permission
          .WRITE_EXTERNAL_STORAGE" />

<application android:icon="@drawable/icon"
          android:label="@string/app_name"
          android:name=".domain.FireTabletApplication">

<activity android:name=".gui.VoiceMemo"></activity>
<activity android:name=".gui.Camera"></activity>
```

Listing 2.6: Ausschnitt aus dem Android Manifest. Der *ApplicationContext* wird bestimmt.

2.3.4 AsyncTasks

AsyncTask ist eine Hilfsklasse des Androidframeworks, die das Zusammenspiel von Activities und Threads vereinfachen. [Painless Threading] Man verwendet Threads um das Userinterface ansprechbar zu halten, während länger dauernde Aktionen in Bearbeitung sind. Dies können Datenbankzugriffe oder Abfragen über das Netzwerk sein. Würden diese Aktionen im gleichen Thread ablaufen wie das Gui, würde dieses blockiert. Darum lagert man grössere Aktionen in eigene Threads aus, die dann parallel zum Userinterface ablaufen.
AsyncTasks kapseln den eben beschriebenen Vorgang, indem sie verschiedene Hilfsmethoden zur Verfügung stellen. Die Wichtigste ist *doInBackground()*. Diese Methode wird überschrieben mit dem Code, der in einem separaten Thread ausgeführt werden soll. Alle anderen Funktionen werden wieder im gleichen Thread der Activity ausgeführt. Die Methoden sind:

- *onCancelled()*
- *onPostExecute(Result result)*
- *onPreExecute()*
- *onProgressUpdate(Progress... values)*

Je nach Methode wird der Code vor, nach oder während der *doInBackground()* Methode ausgeführt.
Beim Erstellen eines *AsyncTask* werden drei Typenparameter deklariert. Diese bezeichnen die akzeptierten Parameter von *doInBackground()*, *onProgressUpdate()* sowie *onPostExecute()*. So kann das Verhalten eines *AsyncTask* völlig flexibel gestaltet werden.
AsyncTasks werden zum Beispiel dafür verwendet, um Fortschrittsanzeigen zu aktualisieren. Dabei läuft ein Prozess ab, der immer an einer gewissen Stelle die Methode *onProgressUpdate()* aufruft, die dann im Gui-Thread die Fortschrittsanzeige aktualisiert.

2.3.5 Parcelable

Objekte, die Parcelable implementieren, können mit Intents verschickt werden. Diese Objekte werden von Android serialisiert und müssen dazu ein paar Funktionen besitzen, die beim Serialisieren und Deserialisieren helfen. [Parcelable Reference]

3 Anforderungen

In diesem Kapitel werden die durch Siemens gestellte Anforderungen an die Applikation dokumentiert. Anschliessend werden die darauf aufbauenden UseCases beschrieben.

3.1 Anforderungen Siemens

Autor: Daniel Bobst

Nach dem Entwurf der Brief Use Cases (Kap. 3.3.1, S. 28) fand das erste Treffen mit Siemens statt. Anwesend war V.Redwitz in Vertretung von D.Stockmann. Bei diesem Treffen ging es darum, den bestehenden Arbeitsprozess einer Anlagenrevision zu analysieren sowie den Umfang der Arbeit festzulegen.
Um den Prozess einer Anlagenrevision zu beobachten, konnten wir über den Hausdienst der HSR einen Termin mit dem für die HSR zuständigen Techniker abmachen. Siehe dazu Kap. 3.2.

Die Applikation soll den Servicetechniker nicht nur beim Meldertest unterstützen, sondern den gesamten Arbeitsprozess beim Kunden vor Ort vereinfachen. Einerseits heisst das, ihm das sofortige Ablesen von Zentraleninformationen und das Absetzen von Befehlen an die Zentrale zu ermöglichen. Andererseits soll auch die elektronische Protokollierung ermöglicht werden, so dass weniger Schreibarbeit anfällt und die Weiterverarbeitung der Testergebnisse effizienter gestaltet wird. Ausserdem bearbeitet der Servicetechniker vor und nach dem Brandmeldertest Checklisten, welche beim Kunden auf Papier vorhanden sind. [Checkliste] Diese zu elektronisieren und auf dem Tablet bearbeitbar zu machen wäre ebenfalls eine Erleichterung für den Techniker. Diese Checklisten sollen nicht nur dazu dienen, die geleistete Arbeit zu dokumentieren, sondern auch als Gedankenstütze dienen, ob alle Arbeitsschritte erledigt wurden.

Zur Hardware gab uns Hr. Redwitz neben Dokumentation auch noch einige interessante Fakten mit. Geräte wie Detektoren, Meldetaster und Signalhörner werden heutzutage mittels Ringleitungen an eine Zentrale angeschlossen. Das bedeutet, dass bei einem Unterbruch der Leitung immer noch alle Geräte mit der Zentrale verbunden sind. Allerdings untersucht die von uns betrachtete Anlagenwartung nicht solche Leitungsunterbrüche. Dies ist unnötig, da ein Unterbruch sofort von der Zentrale bemerkt wird. An einer Ringleitung können verschiedene Gerätetypen in zufälliger Reihenfolge angehängt sein, das heisst es liegt an uns, jene Geräte aus der Anlagenhierarchie herauszufiltern, welche wir benötigen. Pro Ring können maximal 128 Geräte angeschlossen werden.

Die Testauslösung von Meldern mittels Prüfpflücker wurde früher mittels einem Prüfgas durchgeführt. Die moderneren Prüfpflücker verbinden sich mittels Induktion mit dem zu prüfenden Gerät und instruieren es, einen Testalarm abzusetzen. Somit wird nicht die eigentliche Funktion des Sensors im Brandmelder getestet, sondern die Verbindung zur

Zentrale und deren korrekte Konfiguration. Die Überprüfung des Sensors auf Funktionstüchtigkeit wird periodisch vom Melder selbst erledigt. Ist der Verschmutzungsgrad des Sensors zu hoch, um zuverlässig messen zu können, wird dies der Zentrale berichtet.
Was die Protokollierung von Mängeln betrifft, so war ein weiterer wichtiger Hinweis, den Servicetechniker nicht darauf zu beschränken, einen Mangel an einem Melder zu erfassen. Es kann auch sein, dass dem Techniker zum Beispiel ein Defekt an einem Signalhorn auffällt und er dies entsprechend festhalten möchte. Ein weiteres Beispiel wäre, dass der vorgeschriebene Mindestabstand von jeglichem Material zum Brandmelder nicht eingehalten wird.

3.1.1 Ideen für Features

Hr. Redwitz brachte ebenfalls weitere Ideen mit, welche Features die Applikation anbieten könnte.
In keiner bestimmten Reihenfolge waren das:

- Tablet am Arm des Servicetechnikers befestigen
- CAD Grundriss der Anlage auf Tablet laden und Geräte darauf selektierbar machen
- Dimensionen eines Tests kreisförmig als sog. Spider-Chart(Abb. 3.1) darstellen
- Tabletbedienung über Sprachsteuerung und Bluetooth-Headset
- Stundenerfassung für den Servicetechniker integrieren
- Anzeige der Zentrale 1:1 abbilden, um dem Techniker einen vertrauten Anblick zu bieten

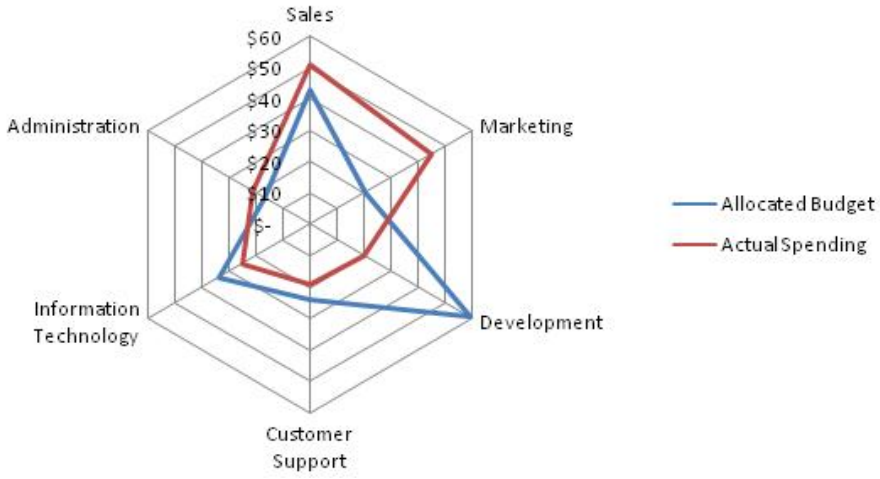


Abbildung 3.1: Beispiel eines Spider-Charts. [Quelle]

Wir entschieden, dass solche Features zwar interessant wären, aber nicht implementiert werden, bevor die Grundfunktionalität gewährleistet ist. Eines der Hauptziele der Arbeit soll sein, die Bedienung der Zentrale über ein mobiles Gerät zu ermöglichen. Bevor dies nicht gelöst ist, wird auf verschönernde Features wie Spider-Charts und Gebäudegrundrisse

verzichtet. Allerdings wären solche Features für eine aufbauende Folgearbeit ideal, um zum Beispiel die Benutzeroberfläche auszubauen oder eine alternative Form der Bedienung zu bieten.

3.2 Anforderungen durch Servicetechniker

Autor: Daniel Bobst

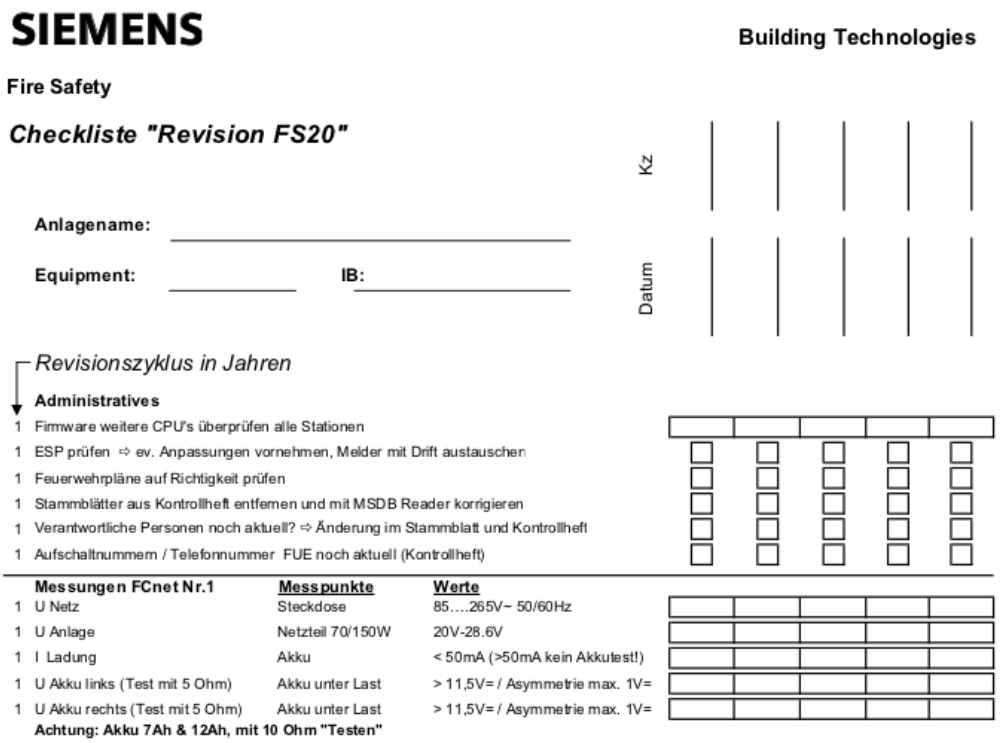
Zufällig fand gerade zum Zeitpunkt unserer Analysephase die jährliche Revision der Brandmeldeanlage an der Hochschule für Technik Rapperswil (HSR) statt. Somit konnten wir einige Stunden lang einem Servicetechniker über die Schulter schauen und offene Fragen klären. Zwar handelte es sich bei der betrachteten Anlage nicht um eine FS20, vom Testablauf her ist sie allerdings vergleichbar.

Die Brandmeldeanlage der HSR besteht aus je einer Zentrale pro Gebäude. Für jede Zentrale existiert ein separates Logbuch, welches bei der Zentrale selbst untergebracht ist. Für jede Zentrale werden Testergebnisse und festgestellte Mängel im Logbuch eingetragen sowie auf dem Laptop des Servicetechnikers protokolliert. So bleibt eine Version des Protokolls beim Kunden und eine wird durch den Techniker an seine Siemens-Vertretung zurückgeschickt.
Die Haupterkenntnis des Treffens ist, dass in einer jährlichen Revision nicht etwa alle Brandmelder getestet werden, sondern nur ein Melder pro Ringleitung. Ausserdem stellte sich heraus, dass verschiedene Aspekte einer Brandmeldeanlage mit unterschiedlicher Periodizität getestet werden. Dies ist auch dokumentiert in [FS20 Maintenance, S.135].
Ebenfalls ist auf der Checkliste links von jedem Eintrag der Revisionszyklus in Jahren angegeben. [Checkliste]

3.2.1 Testablauf

Der Servicetechniker folgt im Prinzip der Checkliste in [Checkliste]. Zuerst wird Administratives erledigt: Sind die Angaben im Logbuch wie verantwortliche Personen oder Feuerwehrpläne noch korrekt? Im nächsten Schritt wird die Funktionstüchtigkeit der Anlage unter verschiedenen Stromversorgungssituationen getestet. Jede Anlage verfügt über Batterien, welche die Anlage für eine gewisse Zeit mit Notstrom versorgen können müssen.

Nach den Messungen wird der physische Zustand der Zentrale geprüft. Es wird auch untersucht, ob die Melderlinien Kurzschlüsse und Unterbrüche korrekt mit einer Störungsanzeige melden.
Danach wird pro Melderlinie ein automatischer Brandmelder ausgelöst , um das korrekte Funktionieren des Ereignisspeichers auf der Zentrale zu testen. Im Gegensatz dazu werden alle Handtaster der Zentrale überprüft. Die Testergebnisse werden im Prüfprotokoll festgehalten. [Prüfprotokoll]
Da der Techniker den Laptop bei der Zentrale lässt, während er die Brandmelder testen geht, muss er bei seiner Rückkehr das Prüfprotokoll anhand der Ereignisanzeige an der Zentrale nachträglich ausfüllen. Dabei muss er sich auf sein Erinnerungsvermögen verlassen, ob das Ereignisprotokoll mit den getätigten Auslösungen übereinstimmt.



SIEMENS Building Technologies

Fire Safety

Checkliste "Revision FS20"

Kz

Datum

Anlagename: _____

Equipment: _____ **IB:** _____

Revisionszyklus in Jahren

Administratives

1	Firmware weitere CPU's überprüfen alle Stationen					
1	ESP prüfen ⇨ ev. Anpassungen vornehmen, Melder mit Drift austauschen					
1	Feuerwehrpläne auf Richtigkeit prüfen					
1	Stammblätter aus Kontrollheft entfernen und mit MSDB Reader korrigieren					
1	Verantwortliche Personen noch aktuell? ⇨ Änderung im Stammblatt und Kontrollheft					
1	Aufschaltnummern / Telefonnummer FUE noch aktuell (Kontrollheft)					
	Messungen FCnet Nr.1	**Messpunkte**	**Werte**			
1	U Netz	Steckdose	85....265V~ 50/60Hz			
1	U Anlage	Netzteil 70/150W	20V-28.6V			
1	I Ladung	Akku	< 50mA (>50mA kein Akkutest!)			
1	U Akku links (Test mit 5 Ohm)	Akku unter Last	> 11,5V= / Asymmetrie max. 1V=			
1	U Akku rechts (Test mit 5 Ohm)	Akku unter Last	> 11,5V= / Asymmetrie max. 1V=			

Achtung: Akku 7Ah & 12Ah, mit 10 Ohm "Testen"

Abbildung 3.2: Ausschnitt aus der Checkliste für Revision FS20.

Nach dem Test der Brandmelder werden unter anderem die Alarmeinstellungen überprüft und ob im Ernstfall ein Alarm korrekt weitergeleitet würde. Nach einigen abschliessenden Einstellungen an der Zentrale wird diese wieder in den Normalzustand versetzt.

3.3 Use Cases

Autor: Samuel Hüppi

Im Kapitel Use Cases wollen wir die Entstehung der verschiedenen Einsatzmöglichkeiten und Anwendungsbereiche untersuchen. In einem ersten Teil sind die "Brief Use Cases" zu sehen. Diese Use Cases sind kurz nach Projektstart entstanden und basieren auf geringem Wissen über die technischen Details. Im weiteren Projektverlauf konnte die Sichtweise auf die Probleme durch viele Gespräche und Spezifikationen anschliessend verdeutlicht werden, was wiederum in angepassten Use Cases resultierte. Am Schluss des Abschnitts werden drei zentrale Use Cases detailliert beschrieben.

3.3.1 Brief Use Cases

- **Priorität 1: Routinetest durchführen**
  - Tablet mit Brandmeldezentrale verbinden
  - Brandmeldertest durchführen
    - Test erfolgreich: Zeit, MelderID und Raum in Protokoll eintragen.

    * Test nicht erfolgreich: Fehler protokollieren, möglicherweise mit VoiceMemo, Bild, Text kommentieren
  - Externe Alarme konfigurieren damit keine Alarmierung erfolgt (Alarm an Feuerwehr etc.)
  - Angebundene Aktionen unterdrücken (Brandschutztüren, Rauchabzugklappen etc.)
  - Testdokumente generieren und ausdrucken (Logeintrag, Übersicht Wartungsbedarf?)
- **Priorität 2: Commissioning**
  - Überprüfen ob Location und ID von Gerät korrekt erfasst ist (Mapping von Geräten zu Location anzeigen)
  - Commissioning Dokumente generieren und ausdrucken.
- **Priorität 3: Zusätzliche Funktionen und Ideen**
  - Brandmeldezentrale mittels Tablet konfigurieren
  - Leuchtdiode am Tablet als Taschenlampe für Monteur benutzen
  - Barcodeerkennung/Texterkennung um weitere Informationen zu einem Gerät abzurufen

3.3.2 Use Cases

Das Design und die Ideen der Use Cases basieren auf verschiedenen Einflüssen. Als erstes konnten wir die schriftlichen Informationen von Siemens auswerten, um zu verstehen, was genau mit dem Programm erreicht werden soll. Weiter haben Meetings mit Servicetechnikern und Ingenieuren von Siemens weitere wertvolle Informationen für das Design der Use Cases geliefert.

Bei den Use Cases spielen folgende Akteure eine Rolle:

- Der Benutzer der Anwendung. Meistens ein Servicetechniker von Siemens, der gerne Informationen über die Anlage hätte oder einen Routinetest durchführen will.
- Das Tablet mit der FireTablet Applikation.
- Das Gateway, welches das Tablet mit der Brandmeldezentrale (BMZ) verbindet und HTTP-Befehle in BACnet-Befehle umwandelt.

Abbildung 3.3: Beteiligte Akteure der FireTablet Applikation

3.3.3 Use Cases

Use Case	Beschreibung
01 Mit Gateway verbinden	Tablet mit dem Gateway verbinden. Dazu wird die IP-Adresse, sowie einige Angaben, zur Anlage beim Benutzer abgefragt. Die Angaben werden anschliessend abgerufen und auf dem Tablet persistiert.
02 Sensortest durchführen	Es können Zonen, Sektionen, Areale oder Panels in den Sensortestmodus versetzt werden. Dabei hört das Tablet auf Testalarme der in diesen Regionen vorhandenen Sensoren.
03 Anlage betrachten	Gesamte Struktur der Anlage wird in Form von Listen dargestellt. Es kann durch den Objektbaum der Anlage navigieren und nach Geräten gesucht werden.
04 Zentrale Funktionsprüfung	Funktionen auf der Zentrale gemäss einer Checkliste prüfen. Falls Mängel fest gestellt werden können diese erfasst werden.
05 Mangel erfassen	Mängel an einem Gerät können in Form von Text, Bild und Ton erfasst werden.
06 Prüfdokumente generieren	Nachdem die gesamte Anlage kontrolliert wurde, kann das Prüfdokument generiert werden. Das Dokument enthält alle vorhandenen Geräte mit den Mängel, welche dafür erfasst wurden. Der Zeitpunkt, wann der Tests durchgeführt wurde ist klar ersichtlich.
07 Bild aufnehmen	Ein Teil des UseCase ”05 Mangel erfassen”. Einem Mangel können Bilder, die mit der Kamera des Tablets erstellt wurden, hinzugefügt werden.
08 Voicememo aufnehmen	Ein Teil des UseCase ”05 Mangel erfassen”. Einem Mangel können Voicememos die mit dem Mikrophon des Tablets erstellt wurden, hinzugefügt werden.
09 Licht an	Falls der Servicetechniker in einem Schacht, einer Hohldecke oder sonst einer dunklen Umgebung testen muss, kann er die eingebaute LED zur Hilfe nehmen.

Tabelle 3.1: Auflistung aller geplanten Use Cases

3.3.4 Ausführliche Use Cases

3.3.4.1 Use Case 01: Mit dem Gateway verbinden

Ziel

Nach dem Starten der Applikation soll das Tablet mit dem Gateway verbunden werden, damit alle Daten zur Anlage geladen werden können.

Ausgangslage

- Der User befindet sich nach dem Start der Applikation im Hauptmenü.

- Es ist noch keine Verbindung mit der BMZ hergestellt worden.

Vorgehen

1. Der Benutzer gibt im Hauptmenü an, dass er einen neuen Zentralentest erstellen will.
2. Ein Dialog zur Eingabe von Anlagenname, IP-Adresse und weiteren technischen Details erscheint. Der Benutzer gibt die IP-Adresse und den Namen ein und bestätigt den Dialog. Das Tablet versucht, mit dem Gateway zu verbinden.
   - Das Tablet konnte sich erfolgreich mit dem Gateway verbinden. Die gesamte Anlagenstruktur wird vom Gateway abgerufen. Anschliessend befindet sich der Benutzer in der Zentralenübersicht.
   - Es konnte keine Verbindung hergestellt werden. Dem Benutzer wird dies via Fehlermeldung mitgeteilt.

3.3.4.2 Use Case 02: Sensortest durchführen

Ziel

Der User möchte testen ob alle Geräte einer Zone, Sektion oder einem Areal richtig funktionieren. Bisher wurde nach den durchgeführten Tests das Eventlog am Anlagenterminal mit den Tests verglichen und der Testerfolg protokolliert. Das Ziel ist, dem Servicetechniker sofortige Rückmeldung über den Test zu geben, sowie die Protokollierung zu automatisieren, was beides zu einem effizienteren Testablauf beiträgt.

Ausgangslage

- Das Tablet ist mit dem Gateway verbunden, die Anlagenstruktur wurde von der BMZ geladen.
- Der Benutzer befindet sich auf dem Übersichtsbildschirm einer Zentrale und möchte eine Zone testen.

Vorgehen

1. Der Benutzer navigiert mit der Zentralenübersicht zum gewünschten Objekt (Panel Areal, Section, Zone).
2. Als Nächstes startet er einen Testlauf für das Objekt, welches gerade angezeigt wird.
3. Das Tablet beginnt auf ankommende Testmeldungen zu warten.
   - Das Tablet besitzt eine Verbindung zum Gateway und wird von diesem informiert, welcher Sensor Alarm ausgelöst hat. Dem Benutzer wird eine Meldung mit der ID des Melders und seinem Standort angezeigt. Sind die angezeigten Informationen korrekt, kann der Techniker dies bestätigen. Falls nicht kann ein Mangel erfasst werden.
   - Das Tablet ist nicht mit dem Gateway verbunden. Es kommen keine Events vom Gateway an. Der Testmodus wird sofort wieder unterbrochen.

4. Sobald der Servicetechniker auf allen Sensoren, die er in den Testmodus versetzt hat ein Testalarm auslöst, beendet er auf dem Tablet den Testmodus. Dabei wird für alle Sensoren, die keinen Testalarm empfangen haben, ein Mangel erstellt und es werden alle Geräte als getestet markiert.

3.3.4.3 Use Case 03: Anlage betrachten

Ziel

Der Benutzer möchte wissen wie die Hierarchie der Brandmeldeanlage aussieht, oder zu einem bestimmten Objekt navigieren um den Sensortest zu starten. Sein Ziel könnte aber auch sein abzuklären, ob auf einem Gerät oder Objekt Mängel erfasst sind.

Ausgangslage

- Das Tablet ist mit dem Gateway verbunden. Die Anlagenstruktur wurde vom Gateway geladen.
- Der Benutzer befindet sich auf dem Bildschirm "Übersicht Zentrale".

Vorgehen

1. Auf diesem Bildschirm wird das Rootelement der Anlage dargestellt. Dies ist normalerweise das Panelobjekt.
2. Jeder Bildschirm besitzt ein Tab, auf welchem die Kindelemente des jeweiligen Objekts angezeigt werden. Wenn der Benutzer ein Kindelement auswählt, wechselt die Sicht zu diesem Objekt.
3. Mit diesem System kann durch die Zentrale navigiert werden, mit dem Backbutton geht es wieder ein Level höher.

4 Interface Design

Autor: Daniel Bobst

Im Kapitel 4.1 wird der erste Entwurf der Benutzeroberfläche beschrieben. Dieser Entwurf wurde aufgrund der Brief Use Cases erstellt. Das bedeutet auch, dass diese Variante der Oberfläche vor dem ersten Meeting mit Siemens und daher auch vor der Finalisierung der Use Cases entworfen wurde. Nach dem Meeting wurde der Entwurf entsprechend der Anforderungen von Siemens überarbeitet. Ausserdem wurde die Menuführung vereinfacht, wie im Kapitel 4.2 zu sehen ist. Schliesslich erkannten wir während der Implementation, dass der vorgesehene Arbeitsfluss noch weiter verschlankt werden konnte, besonders die Integration des Testmodus. Dies ergab den finalen Entwurf und die implementierte Oberfläche, wie sie im Kapitel 4.3 zu sehen ist.

4.1 Erster Entwurf

Beim ersten Entwurf des GUI wurden sämtliche Bildschirme im Hochformat konzeptioniert. Für einzelne Bildschirme wurden zur Erforschung des Konzepts alternative Layouts im Querformat skizziert (vgl. als Beispiel Abb. 4.1 mit Abb. 4.5), aber hauptsächlich gingen wir davon aus, dass das Tablet in einer Hand gehalten wird. Ausserdem deuten die Orientierung der Logos und der Knöpfe an der Vorderseite des Gehäuses darauf hin, dass das Hochformat als Standardausrichtung angesehen wird. Daraus ergab sich die Entscheidung, das Layout vorerst auf dieses Format auszulegen.

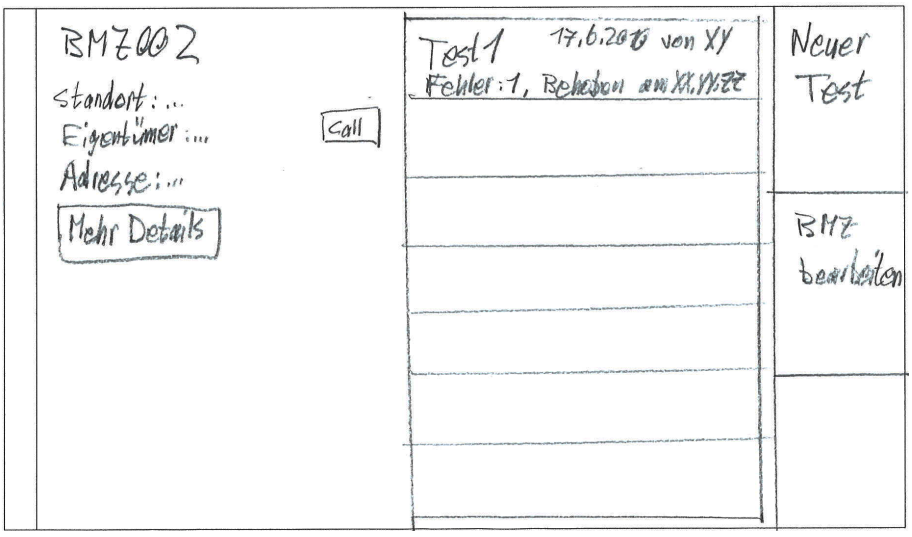


Abbildung 4.1: Alternatives Layout des Bildschirms "BMZ Übersicht"

4.1.1 Übersicht Navigation

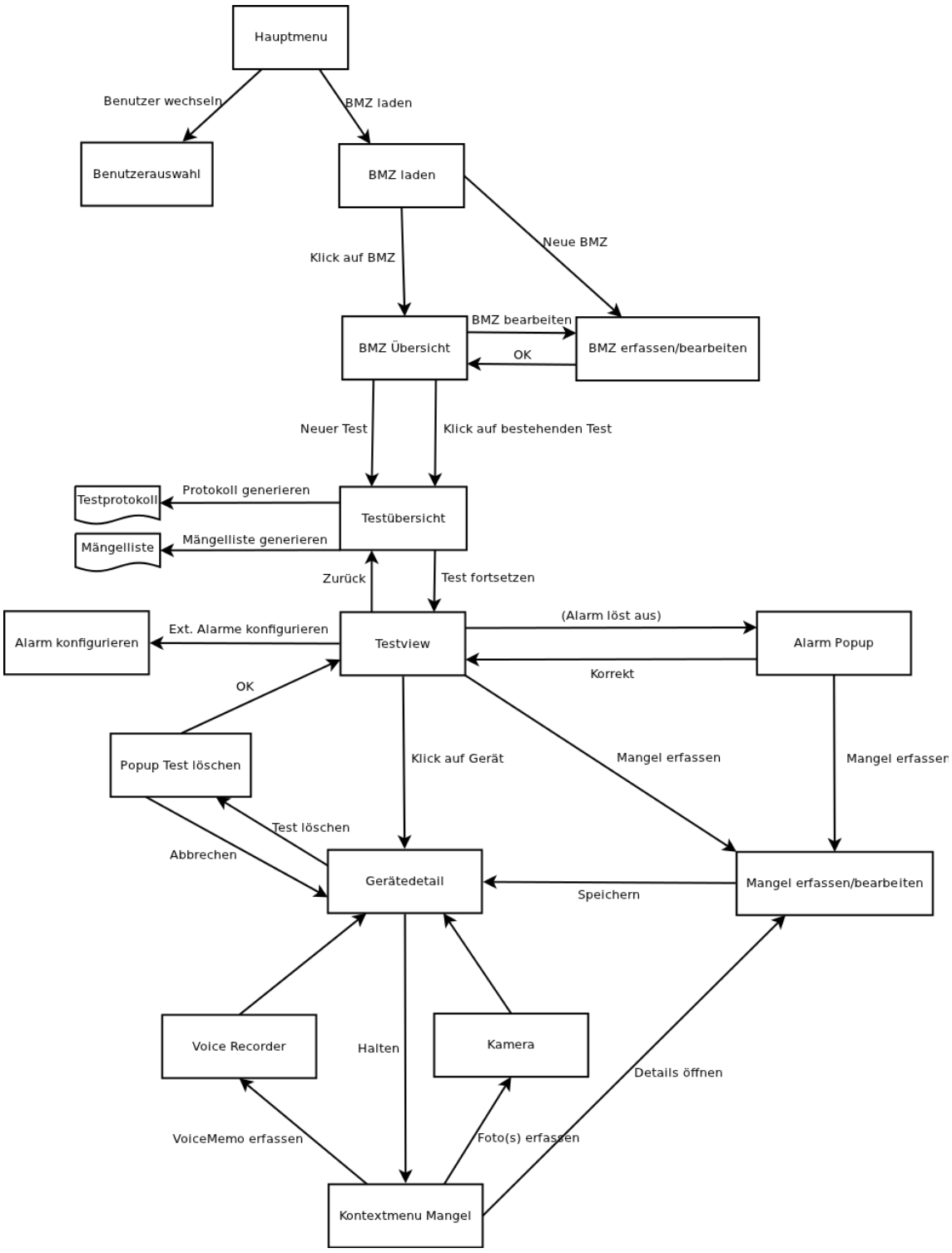


Abbildung 4.2: Navigationsbaum des ersten GUI-Entwurfs

Im obenstehenden Diagramm sind alle Bildschirme und Dialoge sowie die möglichen Navigationspfade zwischen ihnen dargestellt.
Die Bildschirme können entsprechend der durch die Brief Use Cases (Kap.3.3) definierten Tätigkeiten gruppiert werden:

Use Case	Abbildende Bildschirme und Dialoge
Tablet verbinden	Hauptmenu, BMZ laden, BMZ erfassen/bearbeiten
Anlage betrachten	BMZ Übersicht, Testübersicht, Gerätedetail
Melderlinientest durchführen	Testview, Alarm konfigurieren, Alarm Popup, Popup Test löschen
Mangel erfassen	Mangel erfassen/bearbeiten, Kontextmenu Mangel, Voice Recorder, Kamera
Wartung abschliessen	Testübersicht

Tabelle 4.1: Relation der Use Cases zu den entworfenen Bildschirmen

Im Folgenden werden die zusammengehörenden Bildschirme, ihre Funktion und wie sie zusammenhängen erläutert.

4.1.1.1 Tablet verbinden

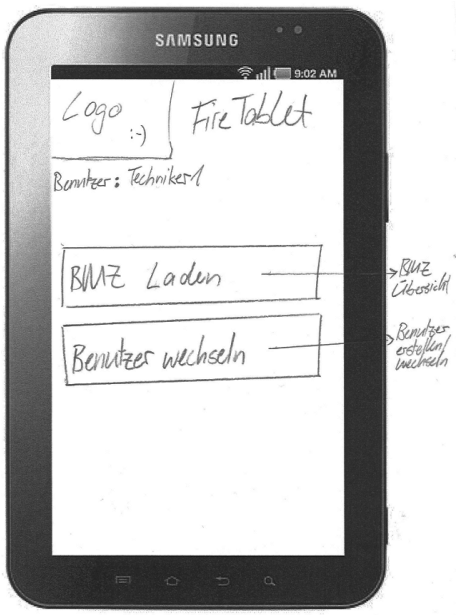


Abbildung 4.3: Hauptmenu

Beim Start der Anwendung wird das Hauptmenu (Abbildung 4.3) angezeigt. Dessen Funktion besteht hauptsächlich darin, den aktuellen Benutzer zu wechseln, bevor mit dem Laden einer Anlage fortgefahren wird. Der Benutzername wird zur Protokollierung und Generierung der Enddokumente benutzt.

Über den Knopf "BMZ laden" erreicht der Benutzer den Bildschirm mit dem selben Titel (Abbildung 4.4). Auf diesem wird eine Liste aller auf dem Gerät gespeicherten BMZs angezeigt. Zu jeder Anlage sollen Angaben wie ihr Name, eine Kontaktperson und die Verbindungsinformationen für den lokalen Gateway zur Anlage erfasst sein. Ein Suchfeld soll bei einer grossen Menge an erfassten Anlagen helfen, die richtige zu finden. Wählt der Benutzer eine BMZ aus der Liste aus, verbindet sich das Tablet mit dem konfigurierten Gateway, lädt die Struktur sowie neue Daten zur Anlage herunter und startet den Bildschirm "BMZ Übersicht" (Abbildung 4.5).Sollte die gewünschte Anlage noch nicht erfasst sein, kann dies mittels des Knopfs "Neue BMZ erstellen" erledigt werden.

Der dazugehörende Bildschirm "BMZ erfassen/bearbeiten" wurde nicht skizziert. Allerdings würden darin Informationen zur BMZ erfasst, die von allgemeinem Interesse oder für die Funktion des Tablets notwendig sind. Einerseits wären das Angaben wie der Anlagenname, Eigentümer und Kontaktinformationen, andererseits technische Angaben wie IP-Adresse und Port des Gateways, um mit der Zentrale zu verbinden.

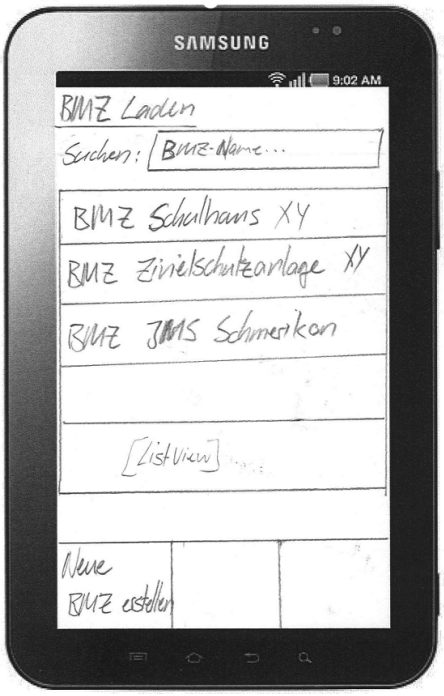


Abbildung 4.4: BMZ laden

4.1.1.2 Anlage betrachten

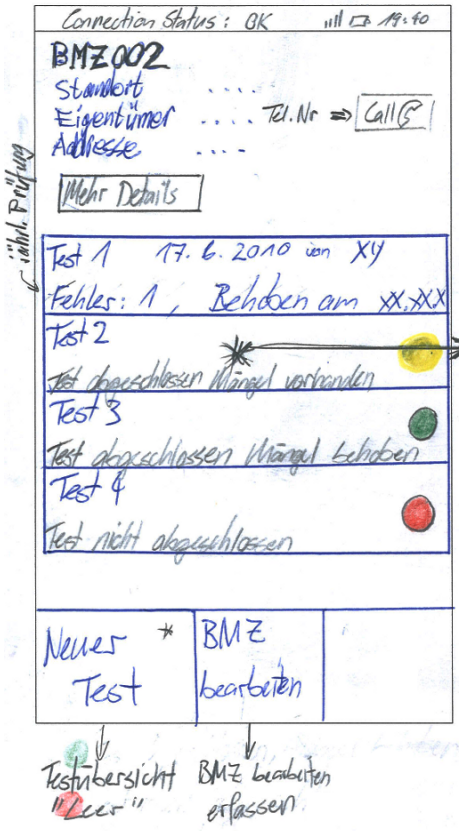


Abbildung 4.5: BMZ Übersicht

Auf dem Bildschirm "BMZ Übersicht" (Abbildung 4.5) werden die wichtigsten Informationen zur Anlage angezeigt. Im oberen Teil stehen Informationen wie der Standort, die Kontaktperson und die IP-Adresse des Gateways. Der Knopf "Mehr Details" führt zu einem Dialog, der als zusätzlicher Platz für ähnliche Informationen zur Anlage dient.
Der grösste Teil des Bildschirms wird von einer Liste eingenommen, die eine Historie der zuletzt durchgeführten Revisionen darstellt. Neben dem Prüfungsdatum und dem Namen des durchführenden Technikers wird der Status des Tests angezeigt. Ein Test kann abgeschlossen (grün), abgeschlossen mit behobenen Mängeln (ebenfalls grün), mit offenen Mängeln abgeschlossen (gelb) oder nicht abgeschlossen (rot) sein. Ein Klick auf einen Test auf der Liste führt zum Bildschirm "Testübersicht" (Abbildung 4.6) des entsprechenden Tests. Ein Klick auf den Knopf "Neuer Test" startet einen neuen Test und wechselt zum selben Bildschirm. Der Knopf "BMZ bearbeiten" führt zu einem ähnlichen Dialog wie der Knopf "Neue BMZ erstellen" auf dem Bildschirm "Hauptmenu". Darin können die zur Anlage erfassten Daten mutiert werden.
Man beachte die Titelleiste in diesem Bildschirm. Eine Idee war, den Zustand der Verbindung zum Gateway mittels der Titelleiste auf jedem Bildschirm sichtbar zu machen und den Benutzer auf irgend eine Art zu warnen, sollte die Verbindung unterbrochen werden.

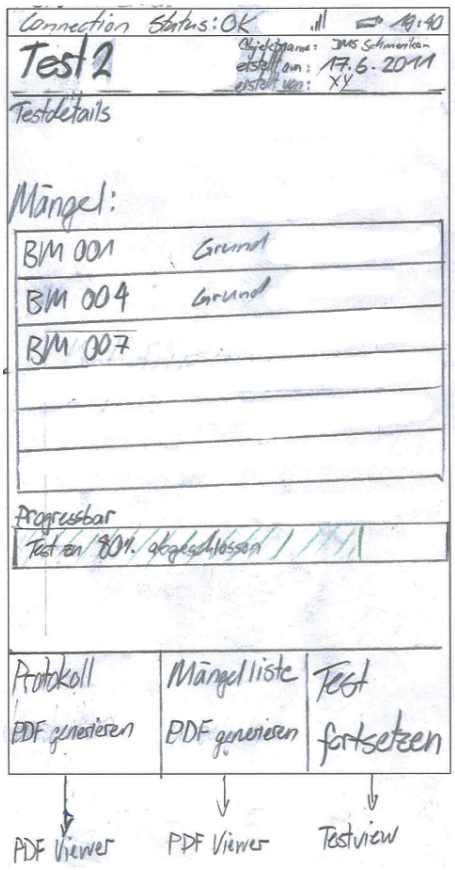


Abbildung 4.6: Testübersicht

Der Bildschirm "Testübersicht" (Abbildung 4.6) stellt den Start- und Endpunkt einer Wartung dar. Neben Angaben wie dem Namen der Anlage, dem Startdatum der Wartung und dem durchführenden Servicetechniker wird zusammenfassend eine Liste mit allen beim Testen von Geräten gefundenen Mängeln angezeigt sowie ein Fortschrittsbalken, welcher Prozentsatz der Melder bereits getestet wurde. Durch Klick auf einen der Listeneinträge werden die Details zu jenem Mangel aufgeführt (s.Kap. 4.1.1.5)
Mit dem Knopf "Test fortsetzen" wird das Tablet in den Testmodus versetzt. Dabei wird der Bildschirm "Testview" (s.Kap. 4.1.1.4 Abbildung 4.9) aufgerufen und das Tablet beginnt, auf vom Gateway kommende Ereignisse zu horchen.
Wird ein Test beendet oder unterbrochen, kehrt der Benutzer zu diesem Bildschirm zurück. Zu den Knöpfen "Protokoll PDF generieren" bzw. "Mängelliste PDF generieren" siehe Kap. 4.1.1.3.

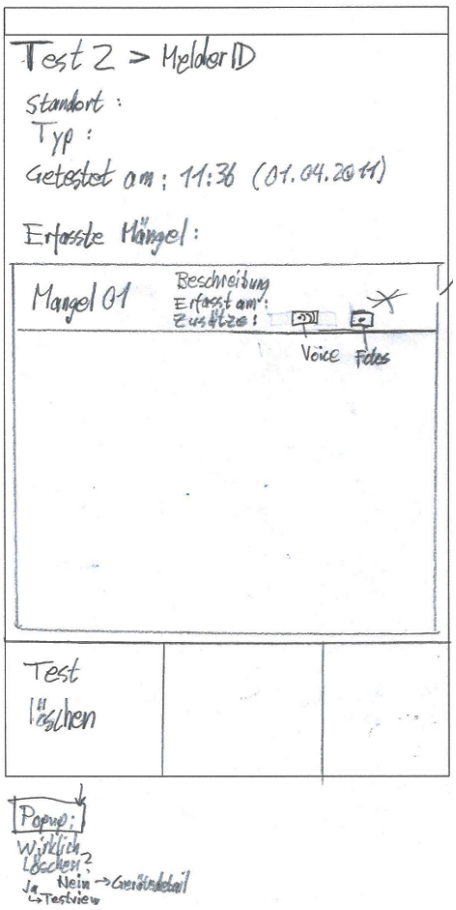


Abbildung 4.7: Detailansicht eines Geräts

Der Bildschirm "Gerätedetails" (Abbildung 4.7) kann zwar nur aus einem laufenden Test heraus betrachtet werden, gehört aber logisch gesehen zu diesem Use Case.
Neben Informationen zum Gerät wie Id, Standort und Meldertyp ist vermerkt, ob das Gerät im Laufe der aktiven Wartung bereits getestet wurde und falls ja, zu welchem Zeitpunkt. Ein Grossteil des Bildschirms wird von einer Liste belegt, die alle zu diesem Gerät während dieser Wartung erfassten Mängel auflistet. Ein Listeneintrag beinhaltet einen Auszug der Mangelbeschreibung, das Erfassungsdatum sowie Icons, ob zum Mangel Voicememos oder Fotos aufgenommen wurden. Durch Klicken auf einen Eintrag in der Liste wird ein Kontextmenu mit möglichen Handlungen geöffnet (s.Abb. 4.8). Am unteren Rand des Bildschirms kann über den Knopf "Test löschen" gewählt werden, ob dieser Testeintrag gelöscht werden soll, als ob das Gerät noch nicht getestet wurde. Dies kann nützlich sein, falls zum Beispiel im Testmodus (s.Kap. 4.1.1.4) eine Ereignismeldung falsch quittiert wurde. Zur Sicherheit erscheint ein Bestätigungsdialog. Wird bei diesem "Ja" gewählt, wird der Testeintrag gelöscht und der Techniker gelangt zurück zum Bildschirm "Testview" (Abb. 4.9). Wählt der Techniker "Nein", bleibt die Anwendung im Bildschirm "Gerätedetails".

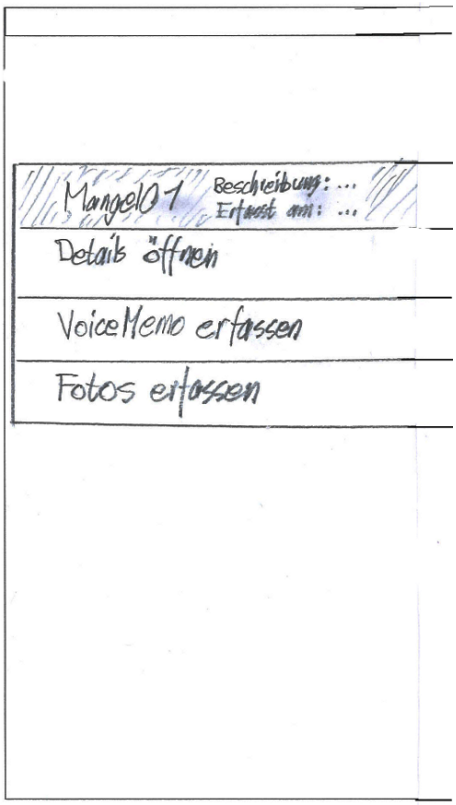


Abbildung 4.8: Kontextmenu bei
der Auswahl eines Mangels im Bildschirm "Gerätedetails" (Abbildung 4.7)

Nach der Auswahl eines Mangels im Bildschirm "Gerätedetails" öffnet sich ein Kontextmenu mit weiteren Optionen. Es können die Details zum gewählten Mangel angezeigt werden, eine Voicememo erfasst oder ein Foto aufgenommen werden. Soll eine VoiceMemo erfasst werden, startet ein Tonaufnahmeprogramm. Ebenso wird das im Tablet eingebaute Kameraprogramm gestartet, falls ein Foto geschossen werden soll. Wird der Punkt "Details öffnen" gewählt, gelangt der Benutzer zur Detailansicht des gewählten Mangels (s.Kap. 4.1.1.5).

4.1.1.3 Wartung abschliessen

Ist die Wartung abgeschlossen, können vom Bildschirm "Testübersicht" aus (Abbildung 4.6) mittels der Knöpfe "Protokoll PDF generieren" und "Mängelliste PDF generieren" die entsprechenden Dokumente erzeugt werden. Diese werden dann auf dem Tablet gespeichert und können beliebig weiterverarbeitet werden.

4.1.1.4 Melderlinientest durchführen

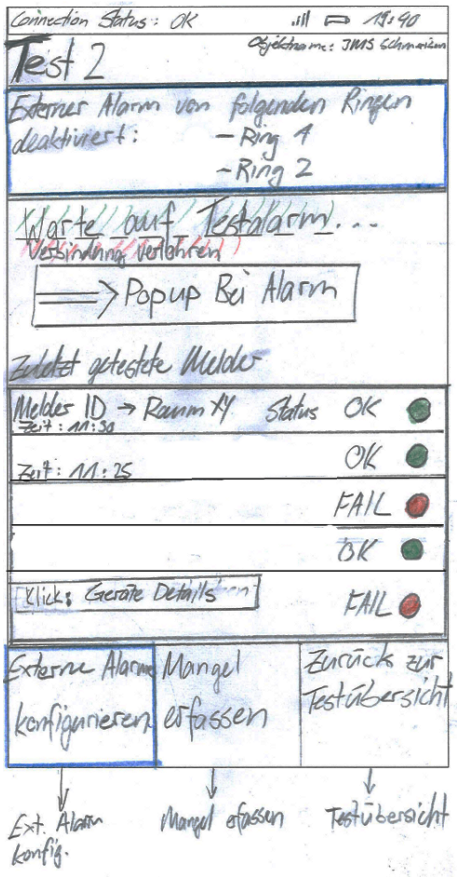


Abbildung 4.9: Testansicht

Bevor ein Testlauf gestartet wird, muss der Benutzer auswählen, welche Geräte in den Testmodus versetzt werden sollen. Dazu drückt er den Knopf "Externe Alarme konfigurieren". Es wird ein Dialog mit allen Areas, Sektionen und Zonen angezeigt, von denen der Benutzer beliebig viele auswählen kann. Nach Bestätigung des Dialogs werden die ausgewählten Objekte auf der Zentrale in den Modus "Detector test" versetzt. (Bem.: Zum Zeitpunkt dieses Entwurfs wurde davon ausgegangen, dass es möglich ist, die an der Zentrale angeschlossenen Geräte entsprechend ihrer physischen Melderlinienringe, an denen sie angeschlossen sind, zusammenzufassen. Wie sich in der Analyse herausstellte, werden die Melderringe in der Detection Domain (s. Kap. 2.1.2) und somit in der Anlagenstruktur auf der Zentrale nicht abgebildet.

Nachdem der Benutzer die zu testenden Geräte ausgewählt hat, verbindet sich das Tablet mit dem Gateway und wartet auf Ereignisse. Im unteren Teil des Bildschirms ist eine Liste der zuletzt in diesem Testdurchlauf getesteten Geräte zu sehen. Bei jedem Listeneintrag ist vermerkt, wann das Gerät getestet wurde und was das Testergebnis war. Wurde beim Auslösen des Melders ein korrekter Testalarm vermerkt, wird der Status "OK" (grün) angezeigt. Trat ein Fehler irgend einer Form auf, wurde ein Mangel erfasst und der Melder

bekommt den Status "FAIL" (rot).

Abbildung 4.10: Popup bei einem eintreffenden Alarmereignis

Wird von der Zentrale für eines der sich im Testmodus befindenden Geräte ein Alarmereignis verschickt, erscheint ein Popup mit Informationen zum Alarm (s. Abb. 4.10). Im Popup ist ersichtlich, welcher Melder laut Zentrale ausgelöst hat. Entspricht dies dem Melder, der vom Benutzer mittels Prüfpflücker ausgelöst wurde, bestätigt der Benutzer den Alarm als "Korrekt". Im Bildschirm "Testview" wird der Melder als "OK" protokolliert. Wird der falsche Melder angezeigt, kann sogleich ein neuer Mangel erfasst werden. Siehe dazu Kap. 4.1.1.5.

Wurde mittels Prüfpflücker ein Testalarm ausgelöst, der nach einer gewissen Zeit offensichtlich nicht von der Zentrale bemerkt wurde, kann der Benutzer vom Bildschirm "Testview" aus mittels des entsprechenden Knopfs einen Mangel erfassen.
Möchte der Benutzer einen laufenden Test beenden, kann er über den Knopf "Externe Alarme konfigurieren" die unter Test stehenden Geräte zurück in den Normalzustand versetzen. Nach Drücken des Knopfs "Zurück" werden nach einem Bestätigungsdialog die Ergebnisse des Tests gespeichert und der Benutzer gerät zurück zum Bildschirm "Testübersicht" (Abb. 4.6). Falls beim Betätigen des "Zurück"-Knopfs der Testmodus noch aktiv ist, werden die entsprechenden Geräte in den Normalzustand versetzt und das Tablet meldet sich bei der Zentrale ab.

4.1.1.5 Mangel erfassen

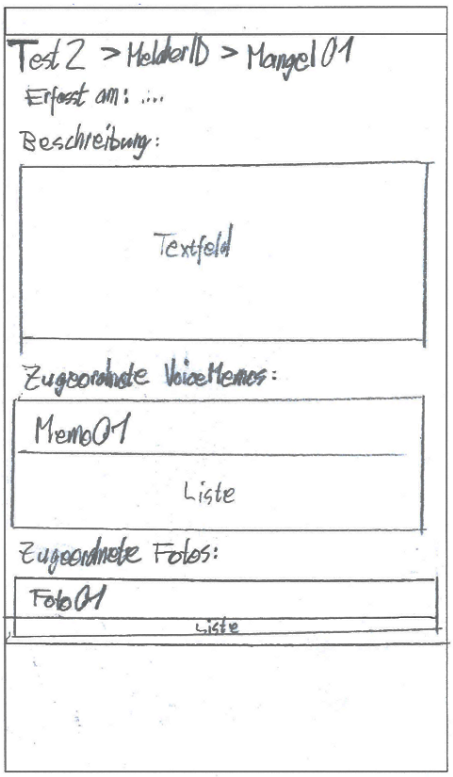


Abbildung 4.11: Detailansicht eines Mangels

Wie in den vorherigen Kapiteln ersichtlich ist, kann aus dem Bildschirm "Testview" (Abb. 4.9) heraus ein neuer Mangel zu einem Gerät erfasst werden. Ebenso kann im Falle eines fehlerhaften Alarmereignisses ein neuer Mangel erstellt werden (Abb. 4.10). Soll ein bestehender Mangel bearbeitet werden, ist dies über den Bildschirm "Gerätedetails" möglich (Abb. 4.7). In jedem Fall gelangt der Benutzer nach dem Neuerfassen oder Bearbeiten eines Mangels mittels des Knopfs "Speichern" zurück zum Bildschirm "Gerätedetails", wo er noch einmal die getätigten Eingaben überprüfen und dann zurück zum Bildschirm "Testview" fortfahren kann.
Wie in der Abbildung 4.11 zu sehen ist, zeigt der Bildschirm neben dem Erfassungsdatum alle vom Benutzer erfasste Informationen an. Im Textfeld im oberen Teil des Bildschirms kann mittels Bildschirmtastatur eine Beschreibung des Mangels eingegeben werden. In den Listen darunter werden erfasste Voicememos bzw. aufgenommene Fotos aufgelistet. Durch einen Klick auf einen Listeneintrag kann dieser abgespielt bzw. angezeigt werden.

4.2 Zweiter Entwurf

Nach der Formulierung des ersten Entwurfs anhand der Brief Use Cases (Kap. 3.3.1) wurden die Anforderungen an die Anwendung in Meetings mit unserem Betreuer sowie Fachverantwortlichen von Siemens präzisiert. Daraus ergaben sich eine Reihe von Änderungen am Entwurf der Oberfläche sowie an der Featureliste. In diesem Abschnitt werden die neuen Anforderungen dargelegt sowie die daraus resultierenden Designänderungen dokumentiert.

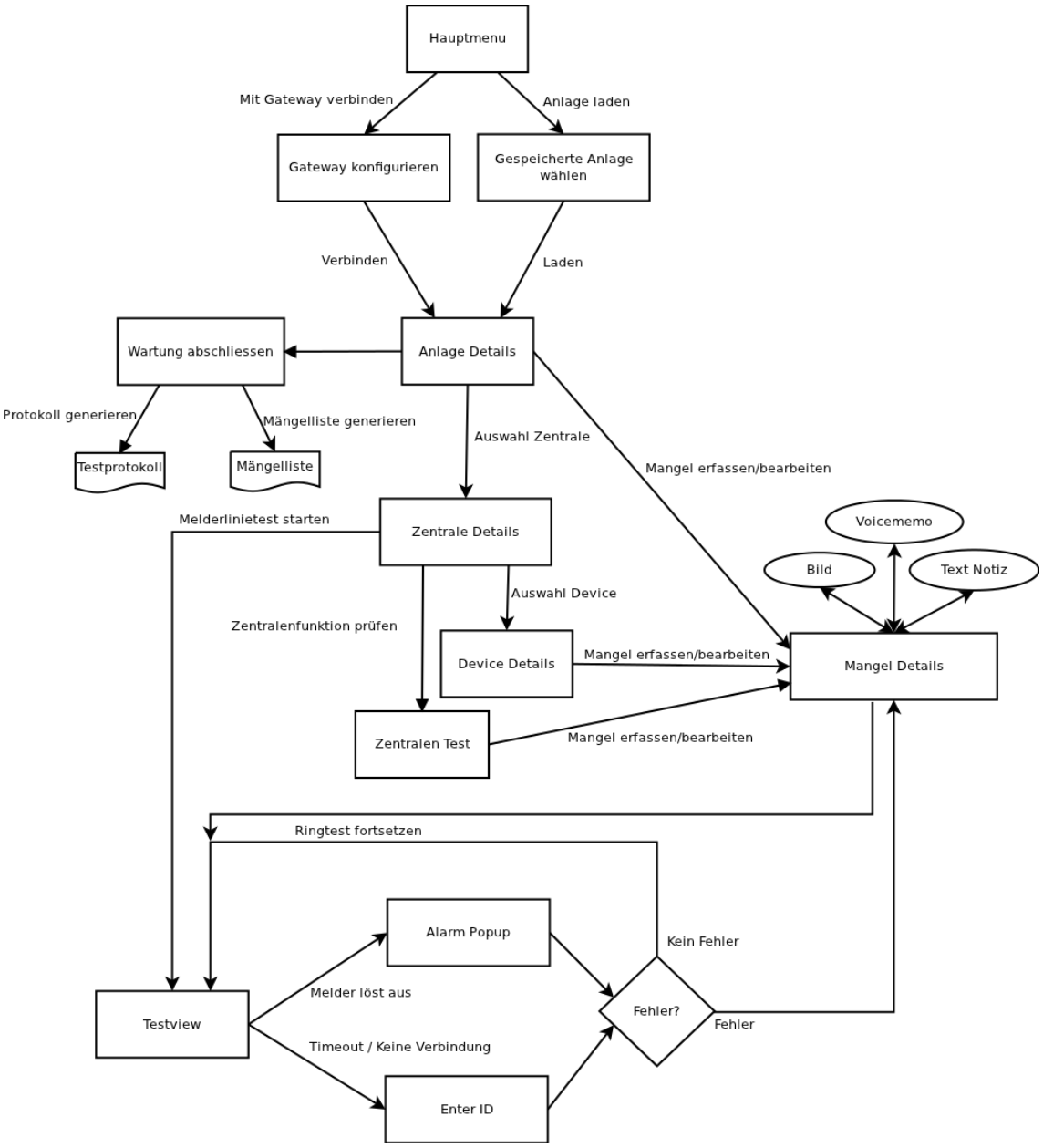


Abbildung 4.12: Navigationsbaum des zweiten GUI-Entwurfs

Im Vergleich zur Abb. 4.2 (S. 34) wurden an einigen Stellen zusätzliche Features eingefügt. Andere wurden weggelassen, wiederum andere verschoben.

Grundsätzlich wurde von Siemens die Idee aufgeworfen, dass das Tablet im Querformat benutzt wird. Eine Anwendungsmöglichkeit wäre, das Tablet am Unterarm zu befestigen und so beide Hände freizuhalten, eine andere, das Tablet an der Stange des Prüfpflückers zu befestigen. Auf jeden Fall wäre bei dieser Anwendungsart ein Quer- praktischer als Hochformat. Ausserdem wäre Platz für grössere Knöpfe und mehr Zusatzinformationen auf dem Bildschirm, ohne dass die sichtbare Grösse von angezeigten Listen wesentlich beeinträchtigt wird. Daher entschieden wir uns, ab dem zweiten Entwurf exklusiv im Querformat zu arbeiten.

Beim ersten Entwurf wurde davon ausgegangen, dass Zentralen als höchstes abzubildendes Element verwendet werden. Zentralen können separat erfasst und geprüft werden. Dies entspricht dem bisherigen Konzept der jährlichen Wartung, die für jede Zentrale getrennt protokolliert wird, wie in [Checkliste] ersichtlich ist. In dieser Entwurfsiteration wurde die Modellierung der Anlage eingeführt. Dank einem zusätzlichen Bildschirm (Abb. 4.13) werden Zentralen gruppiert, wie es der realen Situation entspricht.

Die erste Neuerung, die sogleich beim Betrachten des obigen Diagramms auffällt, ist, dass im Hauptmenu keine Benutzer mehr unterschieden werden. Die einzige Funktion, bei der aus Sicht der Applikation der Benutzer entscheidend ist, ist beim Export der Prüfdokumente, um den durchführenden Techniker zu vermerken. Dies ist einfacher zu lösen, indem beim Dokumentexport der Benutzer nach seiner Personalnummer o.Ä. gefragt wird.

Statt erst im Bildschirm "BMZ laden" (Kap. 4.1, Abb. 4.4) die Möglichkeit zu geben, eine neue Anlage zu erfassen, werden beide Funktionen (Neue Anlage laden & Gespeicherte Anlage laden) bereits im Hauptmenu angeboten.

- Wird "Neue Anlage" gewählt, erscheint ein Dialogfeld, in dem Verbindungsparameter für den Gateway eingegeben werden können. Beim anschliessenden Laden der Anlagenstruktur werden die eingegebenen Parameter mitgespeichert.

- Soll eine gespeicherte Anlage geladen werden, erscheint eine Liste mit allen auf dem Gerät gespeicherten jährlichen Wartungen aller Anlagen. Das heisst, wurde eine Anlage mehrmals mit diesem Tablet geprüft, erscheint für jede Wartung ein datierter Eintrag in der Liste. Das bedeutet, dass der Benutzer nach der Auswahl einer Anlageninstanz aus der Liste sich bereits in einem bestimmten Test der Anlage befindet. Alle Angaben im darauf angezeigten Bildschirm "Anlage Details" (Abb. 4.13) sowie allen folgenden Bildschirmen beziehen sich auf die Struktur und den Zustand der Anlage zum Zeitpunkt jenes Tests.
  Der Vorteil dieser Abbildung ist, dass der Bildschirm "BMZ Übersicht" (Kap. 4.1, Abb. 4.5) des ersten Entwurfs überflüssig wurde. Statt dass die verschiedenen Tests auf der Ebene der Zentralen verwaltet werden, wird der gesamte Anlagenzustand für jede Jahreswartung separat festgehalten. Das ist auch sinnvoll, weil zwar wie oben erwähnt jede Zentrale isoliert getestet wird, aber bei der Revision doch alle Zentralen einer Anlage geprüft werden. Als Nebeneffekt werden so auch Änderungen an der Anlage wie das Hinzufügen von Brandmeldern festgehalten.

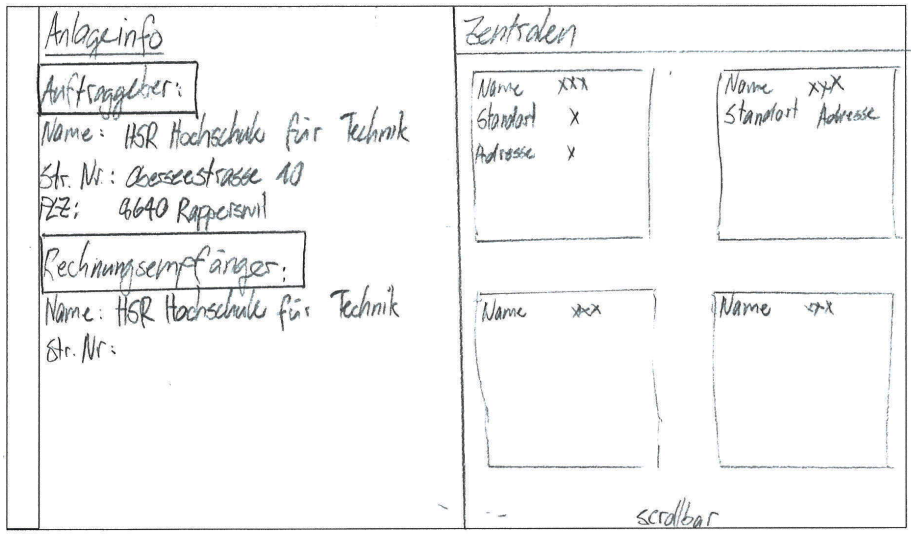


Abbildung 4.13: Anlage Details

4.2.1 Neuer Use Case: Zentrale Funktionsprüfung

Ein weiteres von Siemens vorgeschlagenes Feature ist, den Servicetechniker nicht nur beim Testen der Brandmelder, sondern bei der gesamten Revision zu unterstützen. (s. Kap. 3) Das bedeutet, dass nicht nur das Prüfdokument für Brandmeldertests abgebildet werden soll, sondern auch Checklisten und Dokumente zur Zentralenprüfung wie [Checkliste]
In der Funktionsprüfung sind Aufgaben enthalten wie die Kontrolle der Batteriespannung, der Alarmeinstellungen, etc.

4.2.2 Erweiterter Use Case: Wartung abschliessen

Da wir uns im Bildschirm "Anlage Details" (Abb. 4.13) bereits im Abbild einer jährlichen Wartung befinden, macht es auch Sinn, diesen als Endpunkt einer Wartung zu benutzen. Nachdem der Benutzer alle gewünschten Meldertests durchgeführt und allfällige Mängel dokumentiert hat, klickt er auf den Knopf "Wartung abschliessen". Darauf wird ihm zur Erinnerung eine Checkliste angezeigt, die alle nötigen Schritte auflistet, um die Zentrale wieder vollständig in den Normalzustand zu versetzen. Erst nachdem alle Schritte abgehakt worden sind, lässt sich der Test endgültig abschliessen und die Prüfdokumente können mittels des Knopfs "Dokumente generieren" erstellt werden.

4.2.3 Dokumentgenerierung

Nachdem der Benutzer die abschliessende Checkliste durchgegangen ist, kann er vom Bildschirm "Anlage Details" aus den Knopf "Dokumente generieren" betätigen. Die Dokumente belegen analog zu ihren Papiervorlagen das Datum der Revision, Informationen zur Anlage, den durchführenden Servicetechniker, welche Melder getestet wurden, wo Mängel auftraten sowie die Ergebnisse der Zentralen-Funktionsprüfung. [Checkliste], [Prüfprotokoll]
Der Benutzer hat die Wahl, die Prüfdokumente für eine einzelne Zentrale oder für die komplette Anlage zu generieren. Der Export könnte in Form eines Pdf-Formulars geschehen oder in einem einfach weiterverarbeitbaren Format wie Comma Separated Value (csv).

Ein mögliches Feature für eine Fortsetzungsarbeit wäre, das Dokument nach der Generierung per Email oder FTP-Upload zu versenden anstatt es lokal zu speichern. So könnte das Ergebnis der Revision sofort weiterverarbeitet werden.

4.2.4 Aktualisierte Views

Die Aufteilung der Bildschirme orientiert sich an der physischen Hierarchie der Geräte. Für Anlagen, Zentralen und Geräte gibt es separate Bildschirme. Die Zentralenansicht wurde oben bereits beschrieben (Kap. 4.2). Die Geräteansicht ist funktional identisch mit der in Kap. 4.1.1.2 beschriebenen und wird hier nicht noch einmal erläutert.

Die wichtigste Änderung im Use Case "Anlage betrachten" ist jedoch, dass der Bildschirm "Gerätedetails" nicht nur nach der Erfassung eines Mangels oder über die Testview, d.h. während der Testmodus aktiv ist, aufgerufen werden kann, sondern jederzeit.

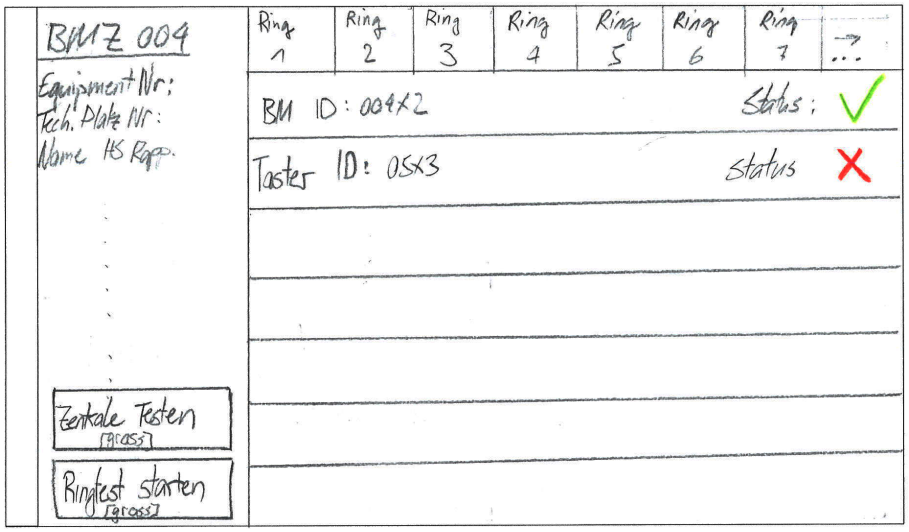


Abbildung 4.14: Zentrale Details

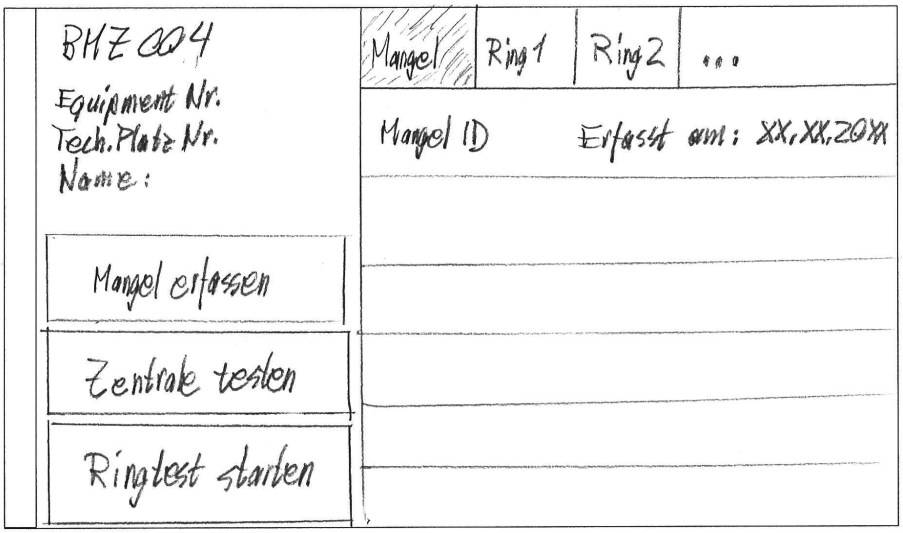


Abbildung 4.15: Mangel Tab in Zentrale Details

Beim Entwurf des Zentralenbildschirms (Abb. 4.14) wurde davon ausgegangen, dass die an der Zentrale angehängten Geräte am besten nach Ringlinien getrennt in verschiedenen Laschen angezeigt werden. Von BACnet spezifizierte Einteilungen wie Area, Section und Zone seien irrelevant. Wie sich herausstellen wird (s. dazu Kap. 4.3) verhält es sich genau anders herum. Als Ausweichmöglichkeit, falls sich Ringe nicht so einfach abbilden lassen, wurde angedacht, die nächsthöhere Hierarchiestufe über den einzelnen Meldern als Sortierkriterium zu verwenden, analog dazu, wie physische Melder durch die Zuteilung zu Ringen gruppiert werden.

In den Laschen für die verschiedenen Ringe werden alle Melder mit ihrem Teststatus angezeigt. Neben diesen gibt es eine einzelne Lasche für die auf dieser Zentrale erfassten Mängel (Abb. 4.15). Ansonsten ist der Bildschirm mit drei Knöpfen ausgestattet.

Mangel erfassen Eröffnet einen neuen Mangel für diese Zentrale.

Zentrale testen Führt den Use Case "Zentrale Funktionsprüfung" aus (s.a. Kap. 4.2.1).

Ringtest starten Versetzt alle Geräte in der gerade aktiven Ringlasche in den Testmodus und wechselt zum Bildschirm "Testview", um auf Ereignisse vom Gateway zu warten.

4.2.5 Testablauf

Beim Entwurf der Testprozedur wurde davon ausgegangen, dass die Zentrale ringweise in den Testmodus versetzt werden kann. Nachdem auf dem Bildschirm "Zentrale Details" (Abb. 4.14) der Benutzer den Knopf "Ringtest starten" gedrückt hat, werden die Melder in der aktuell angezeigten Lasche in den Testmodus gesetzt und zum Bildschirm "Testview" gewechselt.
Beim ersten Entwurf der Navigation (Abb. 4.2, S. 34) wurde angenommen, dass ein Ereignis früher oder später in der Testview eintritt. Es wurde nicht berücksichtigt was passiert, falls der Benutzer einen Testalarm mittels Prüfpflücker auslöst, dieser aber nicht beim Tablet ankommt. Gründe für eine Zeitüberschreitung gibt es diverse, wesentlich ist aber

für diesen Fall, dass ein erwartetes Ereignis nicht angezeigt wird.

Wie in Abb. 4.12 zu sehen ist, wurde das Testverfahren wie folgt erweitert:

- Falls ein ausgelöstes Ereignis auch auf dem Tablet eintrifft und ein Alarmpopup (Abb. 4.10) ausgelöst wird: Überprüfe Korrektheit der Angaben
  - Angaben sind korrekt: Alarm durch Drücken von "OK" bestätigen. Der Ringtest wird mittels Bildschirm "Testview" fortgesetzt
  - Angaben nicht korrekt: Zu diesem Gerät einen Mangel erfassen. Es wird in die Ansicht "Mangel Details" gewechselt. Nach dem Abschluss der Mangelerfassung kehrt der Benutzer auf den Bildschirm "Testview" zurück
- Kommt es zu einem Timeout der Verbindung mit dem Gateway:
  - Id des geprüften Geräts manuell eingeben. Es erscheint das gewohnte Alarmpopup für dieses Gerät.
  - Mittels Prüfpflücker testen, ob der Melder eine Verbindung zur Zentrale hat
    - Verbindung ok: Bestätige manuelles Ereignis durch Drücken von "OK".
    - Verbindung nicht ok: Zu diesem Gerät einen Mangel erfassen.

Da im Fall eines Netzwerkausfalls auf die Informationen des Prüfpflückers zurückgegriffen werden muss, wäre der nächste logische Schritt für ein Folgeprodukt, den Prüfpflücker mit z.B. Bluetooth zu versehen, so dass das Tablet direkt auf dessen Informationen zugreifen kann.

4.2.6 Mangelerfassung

Beim ersten Entwurf wurde davon ausgegangen, dass sich der Umfang der Applikation auf das Durchführen der Brandmeldertests beschränkt. Daher wurde das GUI und auch dessen Navigation darauf ausgelegt, dass ein Mangel nur für einen Brandmelder erfasst wird. Da jedoch laut Siemens wie in Kap. 4.2.1 erwähnt der Servicetechniker bei der gesamten Revision unterstützt werden soll, muss es auch möglich sein, bei anderen Elementen wie z.B. einer Zentrale oder sogar einer Anlage einen Mangel zu erfassen. Da die Zentrale und direkt an sie angehängte Geräte wie Signalhörner oder Batterien im Rahmen des Use Cases "Zentrale Funktionsprüfung" auf ihre Funktionstüchtigkeit überprüft wird, kann es sein, dass ein Mangel für die Zentrale zu erfassen ist.

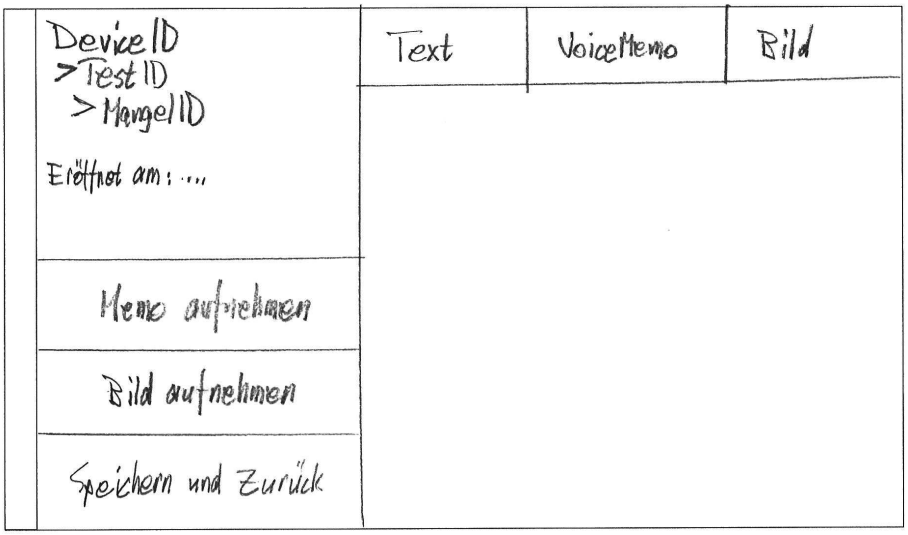


Abbildung 4.16: Mangel Details

VoiceMemos und Fotos werden nicht mehr über ein umständliches Kontextmenu aufgenommen, sondern direkt durch Drücken der Knöpfe "Memo aufnehmen" bzw. "Bild aufnehmen" auf dem Bildschirm "Mangel Details" (Abb. 4.16). Der rechte Teil des Bildschirms besteht aus drei Laschen, in denen die Textbeschreibung des Mangels angezeigt und verändert werden kann sowie die erfassten Voice Memos und Fotos in Listen dargestellt werden (s. Abb. 4.17). Durch Klick auf ein Listenelement wird es abgespielt bzw. angezeigt, durch einen langen Klick wird es nach einem Bestätigungsdialog gelöscht.

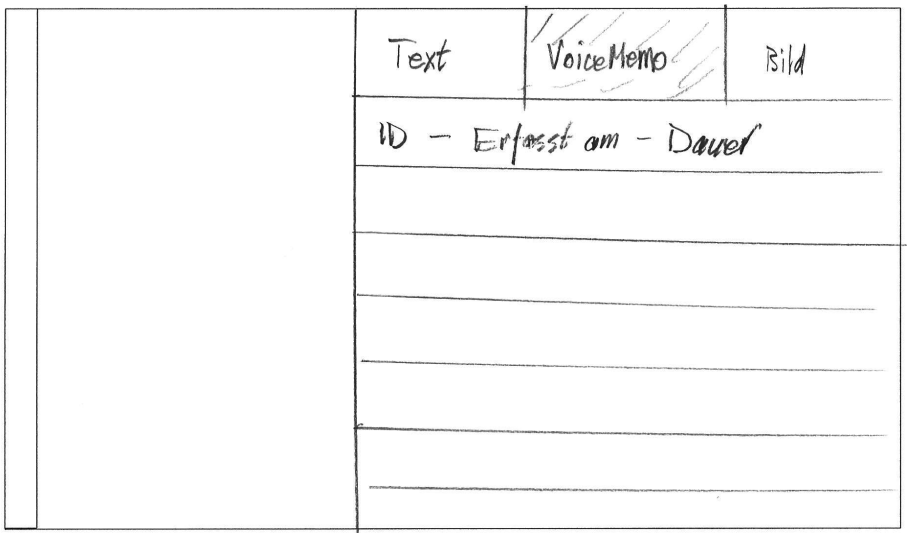


Abbildung 4.17: Mangel Details - Voice Memo Lasche

Es kann sein, dass der Brandmelder selbst funktionstüchtig ist, aber ein Problem in der unmittelbaren Nähe des Melders behoben werden muss. Beispielsweise könnte der vorgeschriebene Mindestabstand zum Brandmelder durch abgestelltes Material verletzt sein. [VdS Richtlinien, Kap.9.3, S.61] In solch einem Fall erfasst der Servicetechniker einen

Mangel mit entsprechender Textbeschreibung für den betroffenen Brandmelder.

Als erleichterndes Feature wäre denkbar, dass je nach betroffenem Gerät ein unterschiedlicher Standardtext für die Beschreibung des Mangels verwendet werden könnte. Beispielsweise würde sich ein Text je für Geräte-, Zentralen- und Anlagenmangel anbieten.

4.3 Finaler Entwurf

Betrachten wir zuerst die grössten Änderungen im Bereich der Applikation, der die Anzeige der Zentrale sowie das Testen von Melderlinien betrifft. Der "obere" Teil der Navigation vom Hauptmenu aus ist prinzipiell der selbe wie im zweiten Entwurf. Wo sich aus Zeitgründen Unterschiede ergeben haben ist in der Umsetzung der entworfenen Features (siehe Kap. 4.3.5).

Während der Implementation erkannten wir, dass besonders die Umsetzung des Testmodus sowie die Darstellung der Zentrale noch effizienter gestaltet werden kann. Im Vergleich zum zweiten Entwurf wird die Struktur in einem einzigen Bildschirm namens "FacilityInfo" dargestellt, und nicht in getrennten Ansichten wie in Kap. 4.2.4 beschrieben. Wird von einem Gerät zu einem untergeordneten Gerät navigiert, so werden die Angaben auf dem selben Bildschirm ersetzt. Der Vorteil ist, dass unabhängig vom Gerätetyp ein einheitliches Bildschirmlayout verwendet wird.

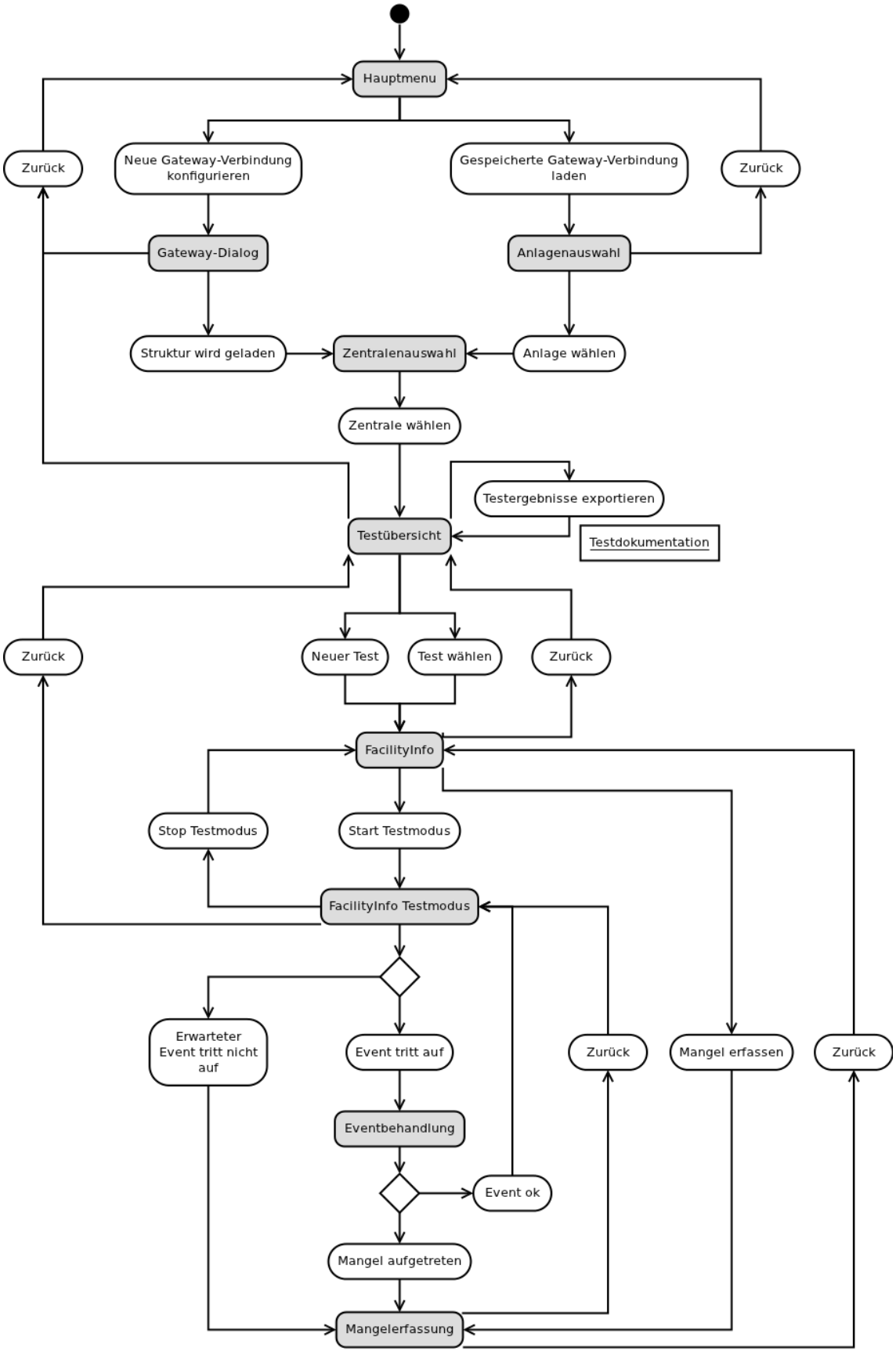


Abbildung 4.18: Flussdiagramm der GUI-Navigation

Die zweite grosse Änderung ist, dass der Testmodus nicht einen separaten Bildschirm benötigt, sondern ebenfalls in die Ansicht "FacilityInfo" integriert wird. Dies vereinfacht die Visualisierung von Gerätezuständen enorm. Es ist keine separate Liste in einer "Testview" mehr nötig, und es ist, sowohl in- als auch ausserhalb des Testmodus, direkt in der "FacilityInfo" ersichtlich, ob ein Gerät schon getestet wurde und falls ja, ob der Test erfolgreich war oder ob Mängel vorliegen. Die Konzentration auf einen einzigen Bildschirm, wenn auch mit mehreren Laschen, macht die Applikation einheitlicher, übersichtlicher und einfacher bedienbar.

Eine Wartung wird nun wieder, wie im ersten Entwurf (Kap. 4.1), von der "Testübersicht" und nicht der "Anlagenübersicht" heraus abgeschlossen. Die Testübersicht stellt alle Testreihen zu einer gewissen Zentrale einer Anlage dar. Das heisst allerdings auch, dass im Gegensatz zum zweiten Entwurf (Kap. 4.2) nicht mehr alle Prüfdokumente einer Anlage auf einmal generiert werden können.

4.3.1 Anlage betrachten

Abbildung 4.19: Hauptbildschirm mit untergeordneten Geräten

Wie in der Abbildung 4.19 zu sehen ist, werden Geräte nicht mehr nach Melderlinienring getrennt angezeigt. Stattdessen betrachtet der Benutzer eines der Geräte in der Zentralenstruktur und bekommt in der Lasche "Elemente" dessen direkt untergeordnete Elemente angezeigt. Das oberste Element in der Hierarchie ist entsprechend der BACnet Spezifikation der Detection Domain (Kap. 2.1.2, S. 10) die Zentrale ("Panel") selbst. Gruppierungen der physischen Geräte in Areale, Sektionen und Zonen werden analog zu den Geräten behandelt. Wie auf dem Bild zu sehen ist, handelt es sich beim Objekt "Erdgeschoss" um eine Sektion und bei den untergeordneten Objekten "Raum 1.206 BM" und "Raum 1.206 HT" um Zonen in dieser Sektion. Wird eine der Zonen ausgewählt, wird zu dieser

gewechselt. Mit dem "Zurück"-Knopf am Tablet kann dann wieder eine Ebene höher zum abgebildeten Bildschirm gewechselt werden.

Über den Knopf "Fehler für . . . erfassen" kann ein Mangel für das aktuell betrachtete Gerät erfasst werden. Aus Zweckmässigkeit kann durch langen Klick auf ein Kindelement über ein Kontextmenu für dieses Kind ein Mangel erfasst werden. Dies erspart dem Benutzer, dass er, nur um einen Fehler zu erfassen, zum Kind wechseln, den entsprechenden Knopf drücken, und wieder hoch wechseln muss (Abb. 4.20).

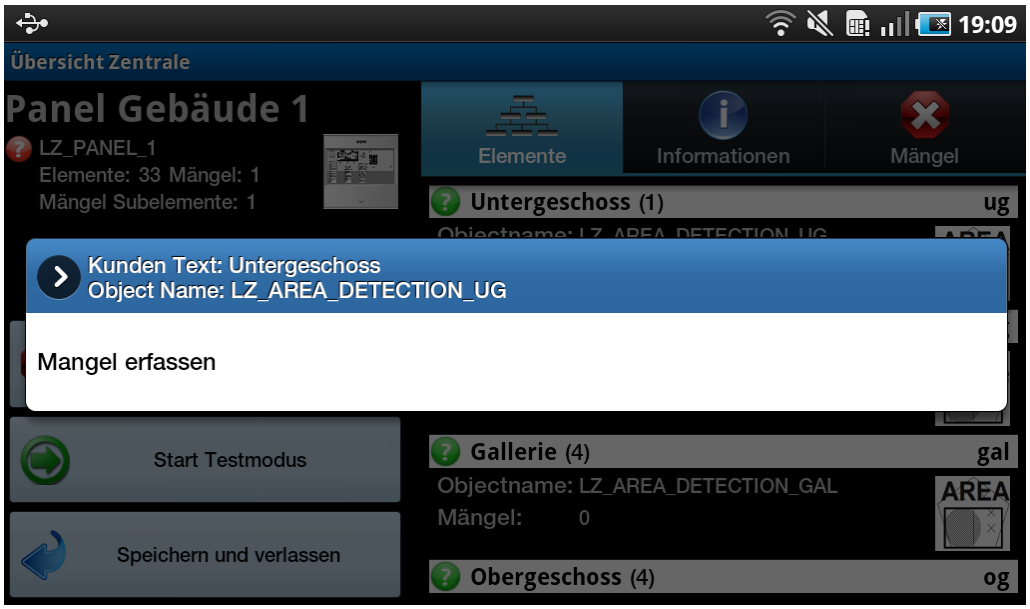


Abbildung 4.20: Kontextmenu für ein Kindelement

Durch Drücken des Knopfs "Start Testmodus" wird offensichtlich der Testmodus gestartet. Im Test inbegriffen sind das aktuell betrachtete Element sowie alle Kindelemente. Siehe Kap. 4.3.3 für eine Beschreibung des Testablaufs.
Mit dem Knopf links unten namens "Zurück zum Panel" kann zur Wurzel des Baums, der Zentrale, gewechselt werden. Für die Zentrale wird der Knopf durch "Speichern und verlassen" ersetzt (Abb. 4.21). Wird dieser betätigt, gelangt der Benutzer nach einem Bestätigungsdialog, ob er den Test verlassen möchte, zurück zum Bildschirm "Testübersicht" (Abb. 4.22.

Abbildung 4.21: Ansicht der Zentrale im Hauptbildschirm

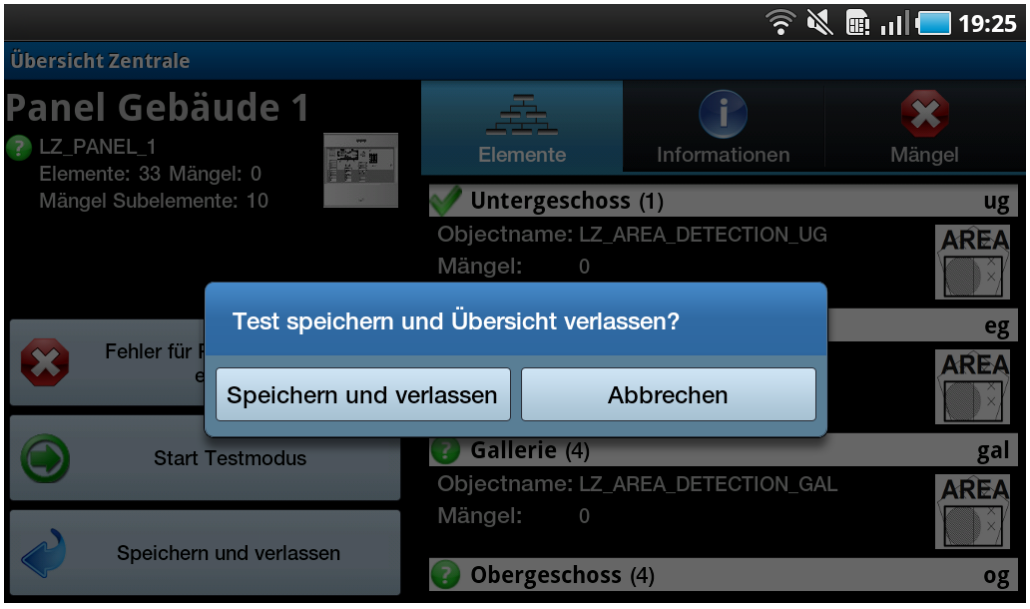


Abbildung 4.22: Bestätigungsdialog zum Verlassen der Zentralenansicht

In der Lasche "Informationen" werden alle Angaben zum betrachteten Objekt angezeigt, die von der Zentrale geladen wurden. (Abb. 4.23)

Abbildung 4.23: Alle Informationen zu einem Objekt

Schliesslich werden in der Lasche "Mängel" alle für dieses Objekt erfassten Mängel angezeigt. (Abb. 4.24) Durch einen Klick auf einen Mangel wird dieser zum Bearbeiten geöffnet (s.a. Abb. 4.34). Durch einen langen Klick lässt sich ein erfasster Mangel löschen (Abb. 4.25). Ist die Mangelliste leer, sieht die Lasche wie in Abbildung 4.26 aus.

Abbildung 4.24: Mängelliste für ein Objekt

Abbildung 4.25: Bestätigung für Löschen eines Mangels

Abbildung 4.26: Leere Mangelliste im Hauptbildschirm

4.3.2 Gerätezustände

Ein Gerät kann mit einer Menge von Icons versehen werden, um dessen Zustand betreffend der aktuell betrachteten Wartung zu bezeichnen.
Vor einem Test besitzt ein Gerät allgemein den Zustand "unknown". Während einem laufenden Test wird der temporäre Status "testing" angezeigt. Nach einem Ereignis befindet sich das Gerät entweder im Zustand "testedok" oder "failed", je nachdem, ob nach der Testauslösung ein korrektes Ereignis gemeldet wurde oder nicht.

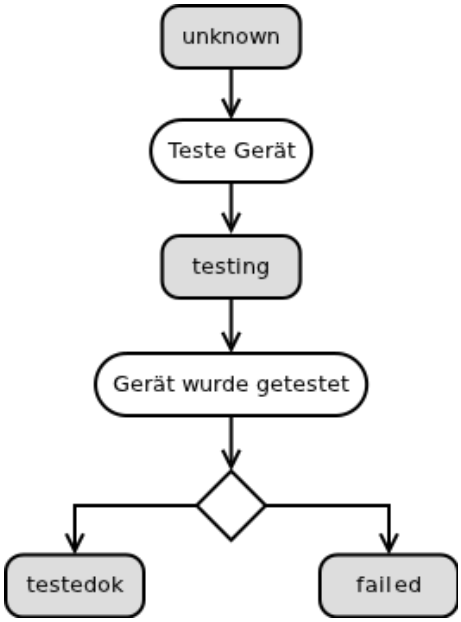


Abbildung 4.27: Mögliche Zustände eines Geräts

In der untenstehenden Tabelle sind alle möglichen Zustände aufgelistet, in denen sich ein Gerät befinden kann.

Gerätestatus \ Mangel	ja	nein
unknown	unknownfaulty	unknown
testing	testing	
ok	n/a	testedok
failed	failed	n/a

Tabelle 4.2: Visuelle Repräsentation des Gerätestatus

Es ist offensichtlich, dass es den Zustand "failed ohne Mangel" nicht gibt. Wir definieren

ausserdem, dass der Status "ok mit Mangel" nicht existiert. Wir unterscheiden nicht, ob der Mangel vom Testen des Melders kommt oder ob sonst etwas in der Umgebung mangelhaft ist. Ist einem Gerät ein Mangel angefügt, so hat es den Test nicht bestanden. Oder anders gesagt: Will bei der Auswertung unterschieden werden, ob der Mangel durch den Test mit dem Prüfpflücker oder andersweitig entstanden ist, so kann die Beschreibung des Mangels betrachtet werden.

Der Status "testing" ist ein Übergangszustand, der gültig ist, während ein Gerät sich unter Test befindet. Bevor ein Gerät in den Testmodus versetzt wird, besitzt es den Status "unknown". Nachdem der Testmodus gestoppt wird, ist das Gerät entweder in Ordnung ("testedok") oder nicht ('failed"). Daher werden auf einem Gerät nur die Zustände "unknown", "testedok" und "failed" gespeichert.

Als Präzisierung der in 4.27 definierten allgemeinen Zustände zeigt das folgende Flussdiagramm (Abb. 4.28) alle gültigen Zustandsübergänge, die vor, während und nach einem Test auftreten können.
Bevor ein Test auf einem Gerät gestartet wird, befindet es sich entweder im Zustand "unknown" oder "unknownfaulty", je nachdem, ob vom Benutzer bereits abgesehen vom Test mittels Prüfpflücker ein Mangel erfasst wurde. Wird der Test gestartet, bekommt das Gerät den Status "testing". Tritt im Testmodus ein Ereignis auf, ist dieses korrekt und ist das Gerät andersweitig mangelfrei, wird ihm der Zustand "testedok" vergeben. In allen anderen Fällen wird das Gerät in den Zustand "failed" überführt. Dieser Zustand kann bedeuten, dass ein fehlerhaftes Ereignis auftrat, ein erwartetes Ereignis nicht auftrat oder der Test erfolgreich war, das Gerät aber sonstige Mängel aufweist. Zu beachten ist, dass sowohl beim Übergang "Erwartetes Ereignis tritt nicht auf" als auch bei "Stop Testmodus" für dieses Gerät kein Ereignis eintraf. Der Unterschied ist, dass bei "Stop Testmodus" ein Mangel automatisch erfasst wird, während bei ersterem der Benutzer selbst diesen erfasst.

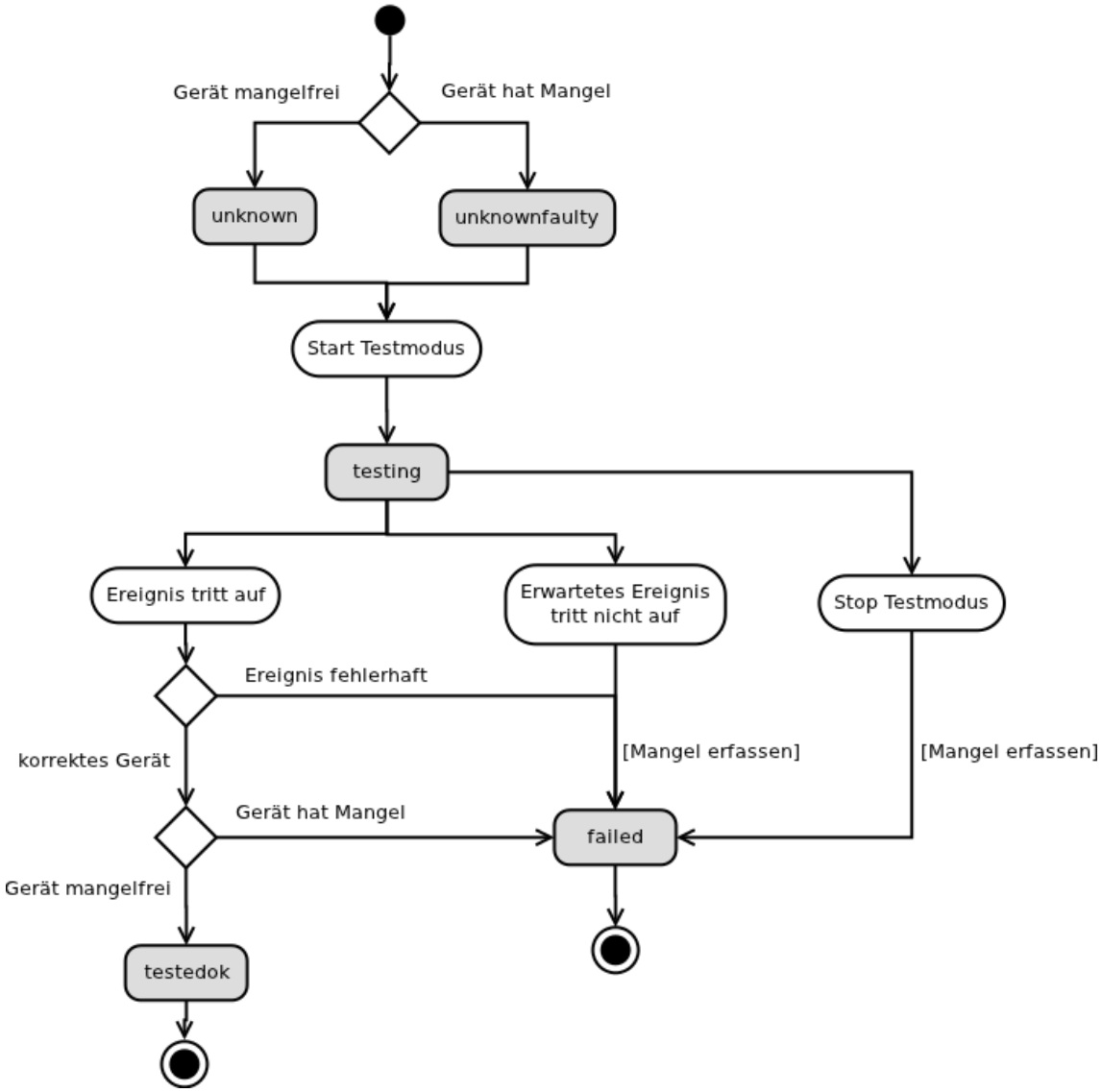


Abbildung 4.28: Angezeigte Zustände eines Geräts während eines Tests

4.3.3 Testen von Geräten

Befindet sich das Tablet im Testmodus, d.h. hört es auf vom Gateway kommende Ereignisse für die unter Test stehenden Melder, wird dies wie in Abb. 4.29 angezeigt. Die animierten Icons (in Tabelle 4.2 "testing" genannt) in der Liste der Kindelemente signalisieren, dass sie auf ein Ereignis warten. Ausserdem wird mit einer roten Box und entsprechendem Text angezeigt, dass der Testmodus aktiv ist.

Abbildung 4.29: Zentralenansicht im Testmodus

Empfängt das Tablet ein Ereignis für einen der unter Test stehenden Melder, wird dies in einem Popup angezeigt (Abb. 4.30). Der Benutzer kann aufgrund der Information im Popup entscheiden, ob der korrekte Melder ausgelöst hat. Ist dies der Fall, bestätigt der Benutzer den Dialog mittels des Knopfs "Bestätigen". Darauf wird das Gerät mit dem in Tabelle 4.2 aufgeführten Icon "testedok" versehen (Abb. 4.31).

Abbildung 4.30: Eintreffen eines Ereignisses

Abbildung 4.31: Zustand eines Objekts nach Bestätigung eines korrekten Ereignisses

Wie in dieser Abbildung ebenfalls zu sehen ist, wird ein nicht-physisches Element wie ein Areal, eine Sektion oder eine Zone ebenfalls mit dem Status "testedok" versehen, wenn alle untergeordneten Melder den Status "testedok" besitzen.

Ist mit dem eintreffenden Ereignis etwas nicht in Ordnung, kann der Benutzer im Popup durch Klicken auf "Fehler erfassen" sogleich einen neuen Mangel eröffnen. Kehrt er danach aus der Mangelerfassung zum Bildschirm "FacilityInfo" zurück, ist der Melder und somit die Zone, in der sich der Melder befindet, mit dem Icon für den Status "failed" versehen (Abb. 4.32).

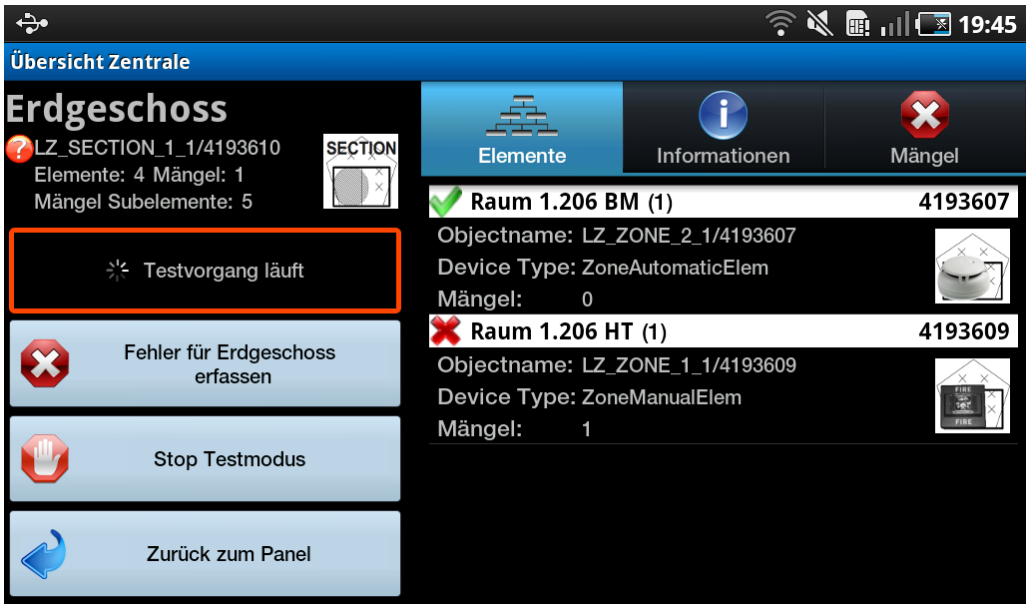


Abbildung 4.32: Zustand eines Objekts nach einem fehlerhaften Ereignis

Was passiert, wenn ein Ereignis mittels Prüfpflücker ausgelöst wurde, aber nicht eintritt? Wir entschieden uns dagegen, nach einer gewissen Wartezeit ein Timeout auftreten zu lassen. Stattdessen wartet das Tablet auf unbestimmte Zeit auf das nächste Ereignis, solange es sich im Testmodus befindet. Dem Benutzer stehen zwei Handlungsmöglichkeiten offen:

- Der Benutzer kann zum betroffenen Gerät navigieren und für dieses einen Mangel erfassen.
- Er kann mit dem Testen des nächsten Melders fortfahren. Beendet der Benutzer den Testmodus, werden für alle Geräte, die sich noch im Zustand "testing" befinden, automatisch ein Mangel mit einem Standardtext erfasst ("Keine Antwort beim Sensortest empfangen.") und somit das Gerät in den Zustand "failed" versetzt (s. Abb. 4.33).

Abbildung 4.33: Automatisch erfasster Mangel für einen nicht getesteten Melder

4.3.4 Mangel erfassen

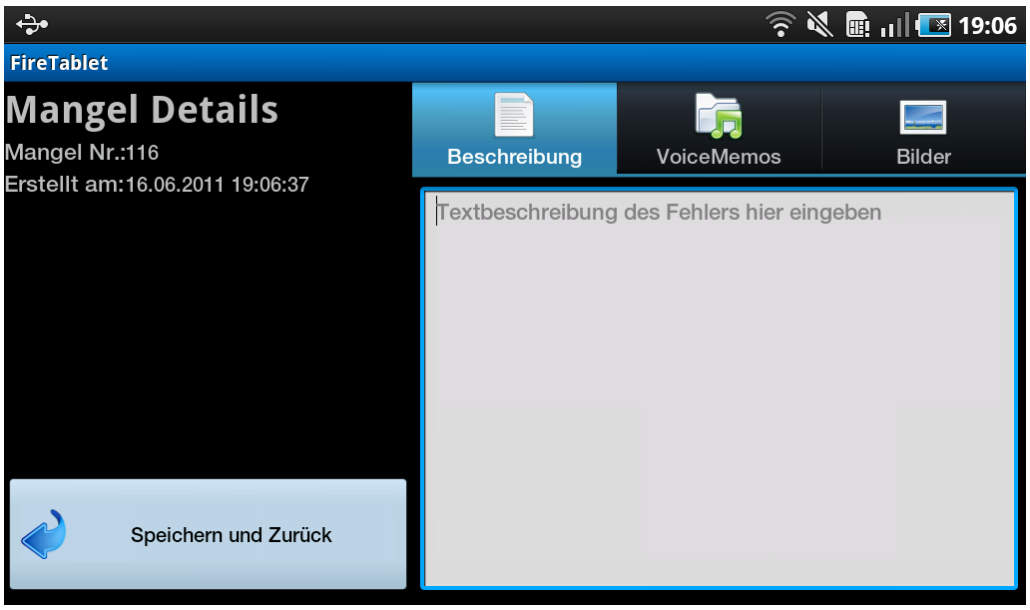


Abbildung 4.34: Mangelerfassung mit Lasche "Beschreibung"

Der Bildschirm "Mangelerfassung" besteht hauptsächlich aus den drei Laschen "Beschreibung", "VoiceMemos" und "Bilder", in denen entsprechende Inhalte erfasst werden können (Abb. 4.34). Die Lasche "Beschreibung" besteht aus einem Textfeld, in dem eine beliebig lange Beschreibung des Mangels eingetippt werden kann.

In der Lasche "VoiceMemos" (Abb. 4.35) kann durch Drücken des Knopfs "VoiceMemo aufnehmen" das im Tablet eingebaute Mikrofon aktiviert und eine Aufnahme gestartet werden. Soll die Aufnahme gestoppt werden, kann dies durch Betätigen des gleichen Knopfs (nun genannt "Stop") bewirkt werden (Abb. 4.36). Danach wird die Datei auf den Datenträger gespeichert und der Speicherort in unserer Datenbank zu diesem Fehler zugeordnet (Abb. 4.37).

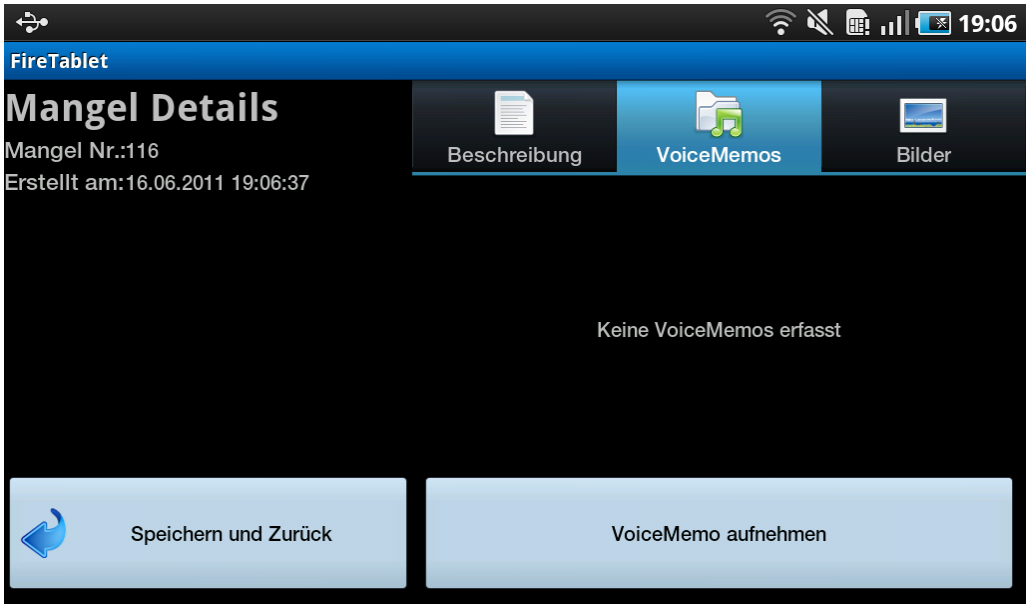


Abbildung 4.35: Leere Liste in der Lasche "VoiceMemo"

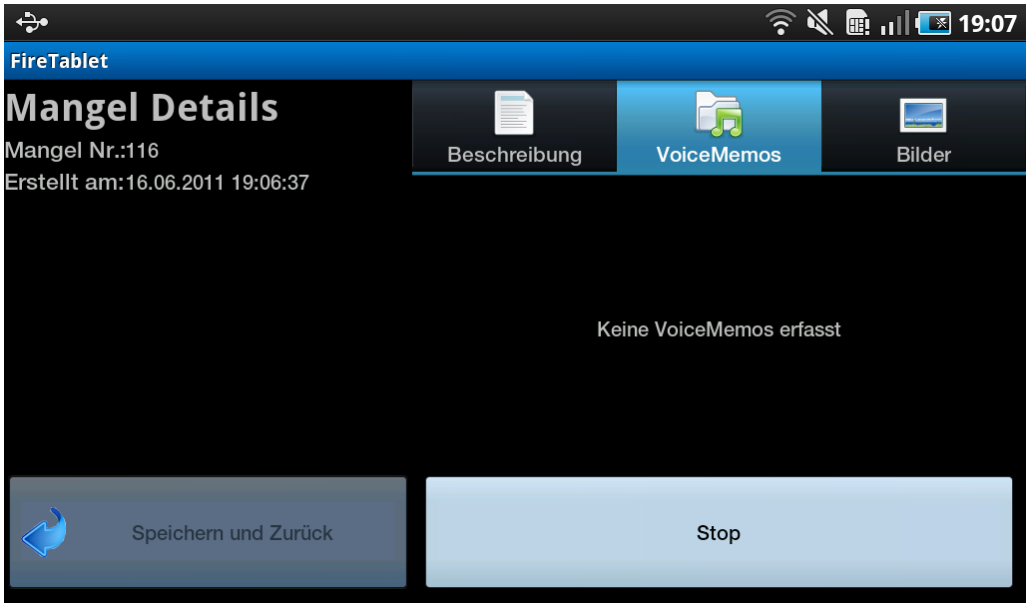


Abbildung 4.36: Während laufender Aufnahme

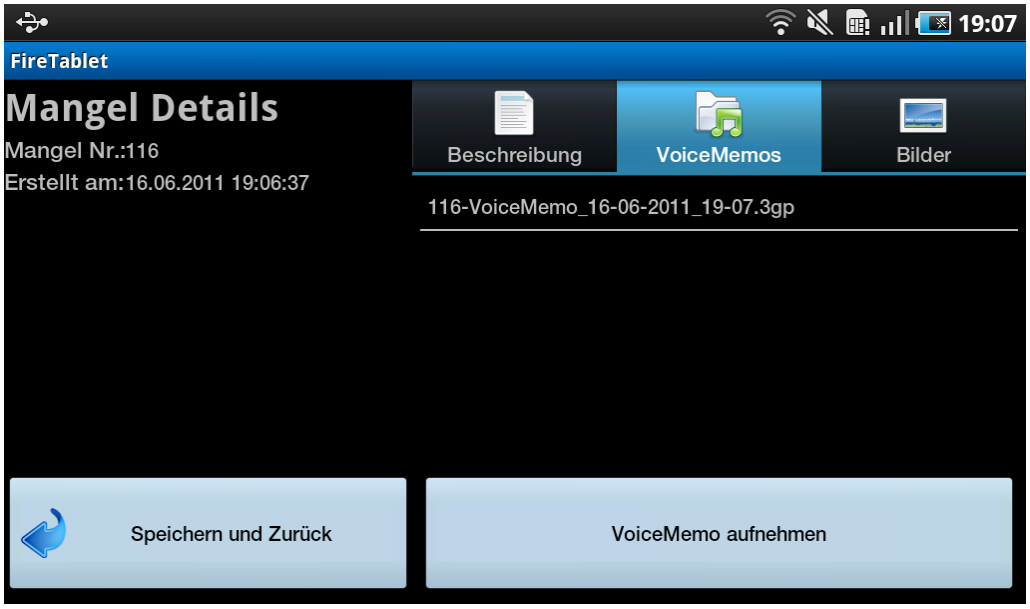


Abbildung 4.37: Lasche "VoiceMemo" mit Aufnahme als Listenelement

Analog verhält sich die Erfassung von Bildern, nur dass dabei statt einem Aufnahmegerät die als Standard eingestellte Kamera-Applikation des Tablets gestartet wird (Abb. 4.38, 4.39).

Soll eine Ton- oder Bildaufnahme gelöscht werden, kann dies durch einen langen Klick auf ein Listenelement veranlasst werden. Nach einem Bestätigungsdialog (Abb. 4.40) wird der Verweis aus der Datenbank sowie die Datei vom Datenträger entfernt.

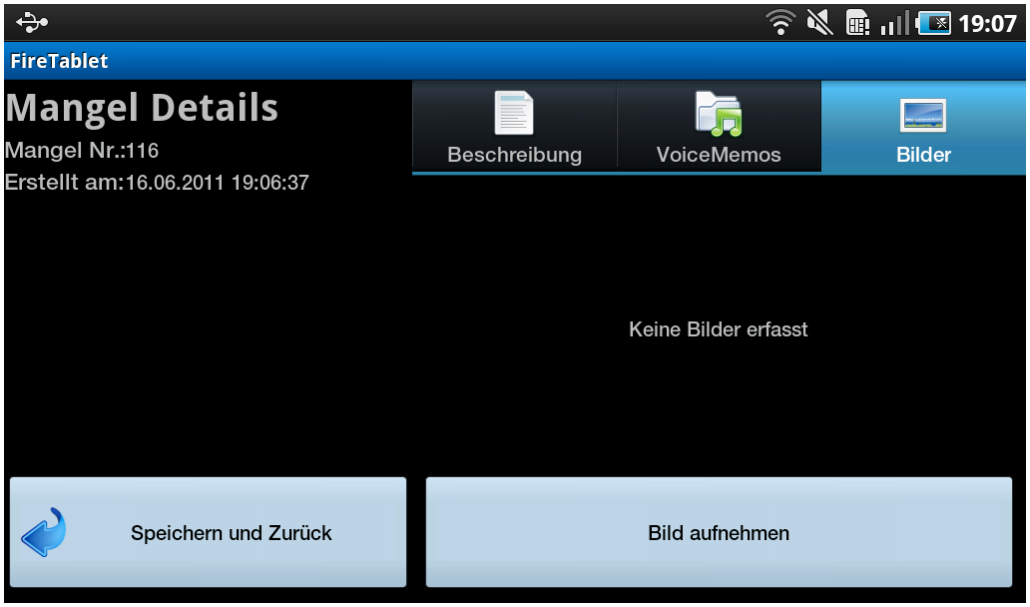


Abbildung 4.38: Leere Liste in der Lasche "Bilder"

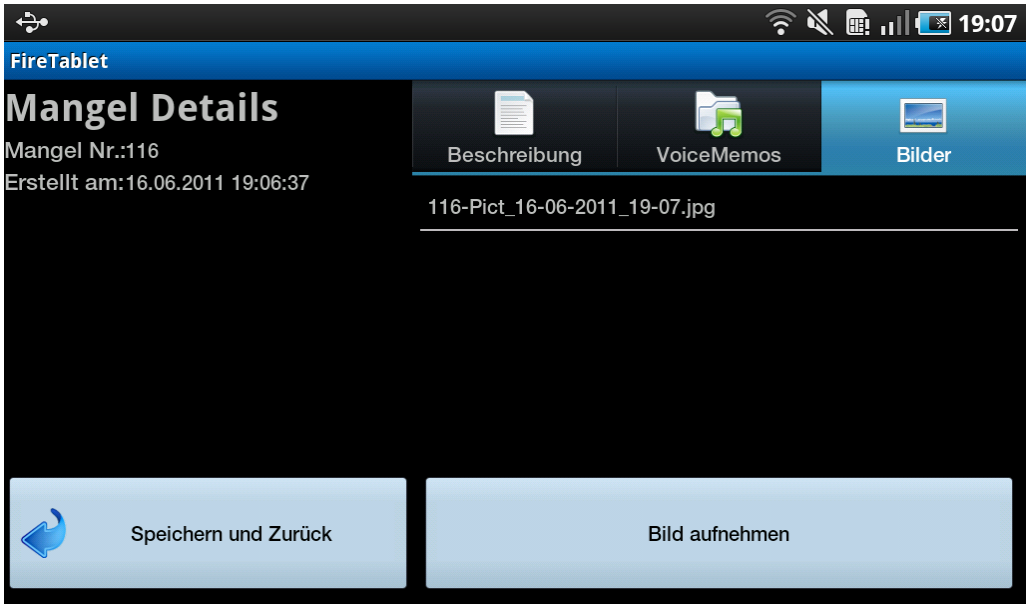


Abbildung 4.39: Lasche "Bilder" mit Aufnahme als Listenelement

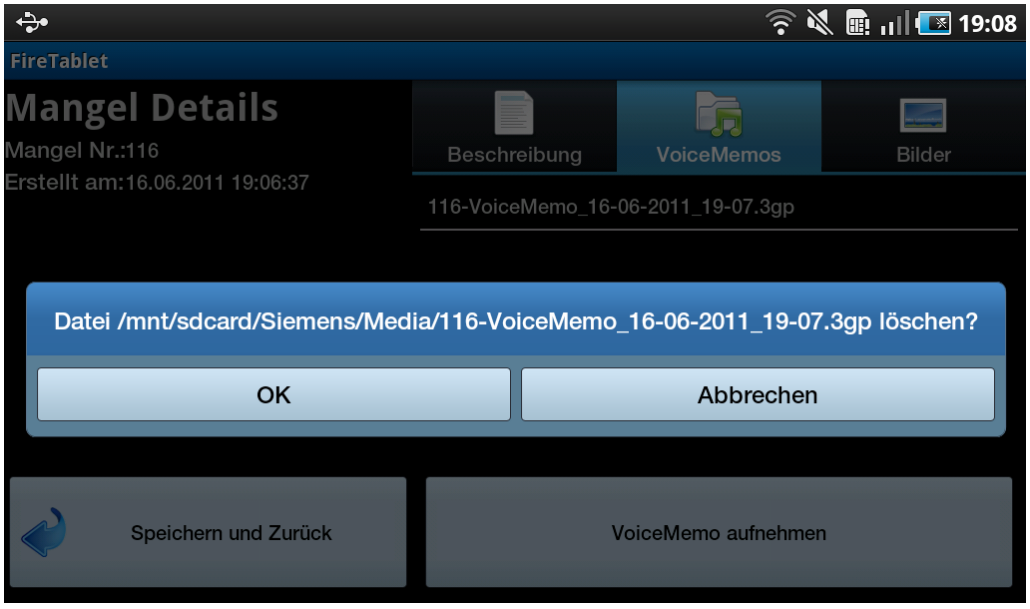


Abbildung 4.40: Bestätigungsdialog zum Löschen einer Ton- oder Bildaufnahme

4.3.5 Implementation - Differenzen gegenüber Entwurf

Aufgrund des Treffens mit einem Servicetechniker (siehe Kapitel 3.2) stellte sich heraus, dass bei einer Anlagenrevision jede Zentrale separat getestet wird. Dies spiegelt sich auch darin, dass die Dokumentation sowohl vor Ort als auch für Siemens für jede Zentrale getrennt geführt wird. Und nicht zuletzt befassten wir uns aufgrund unseres Testaufbaus vor allem mit einer einzelnen Zentrale und ihren untergeordneten Elementen. Aus diesen Gründen haben wir auch entschieden, dass für Demonstrationszwecke unsere Implemen-

tation keine Anlagen abbildet, sondern einzelne Zentralen.

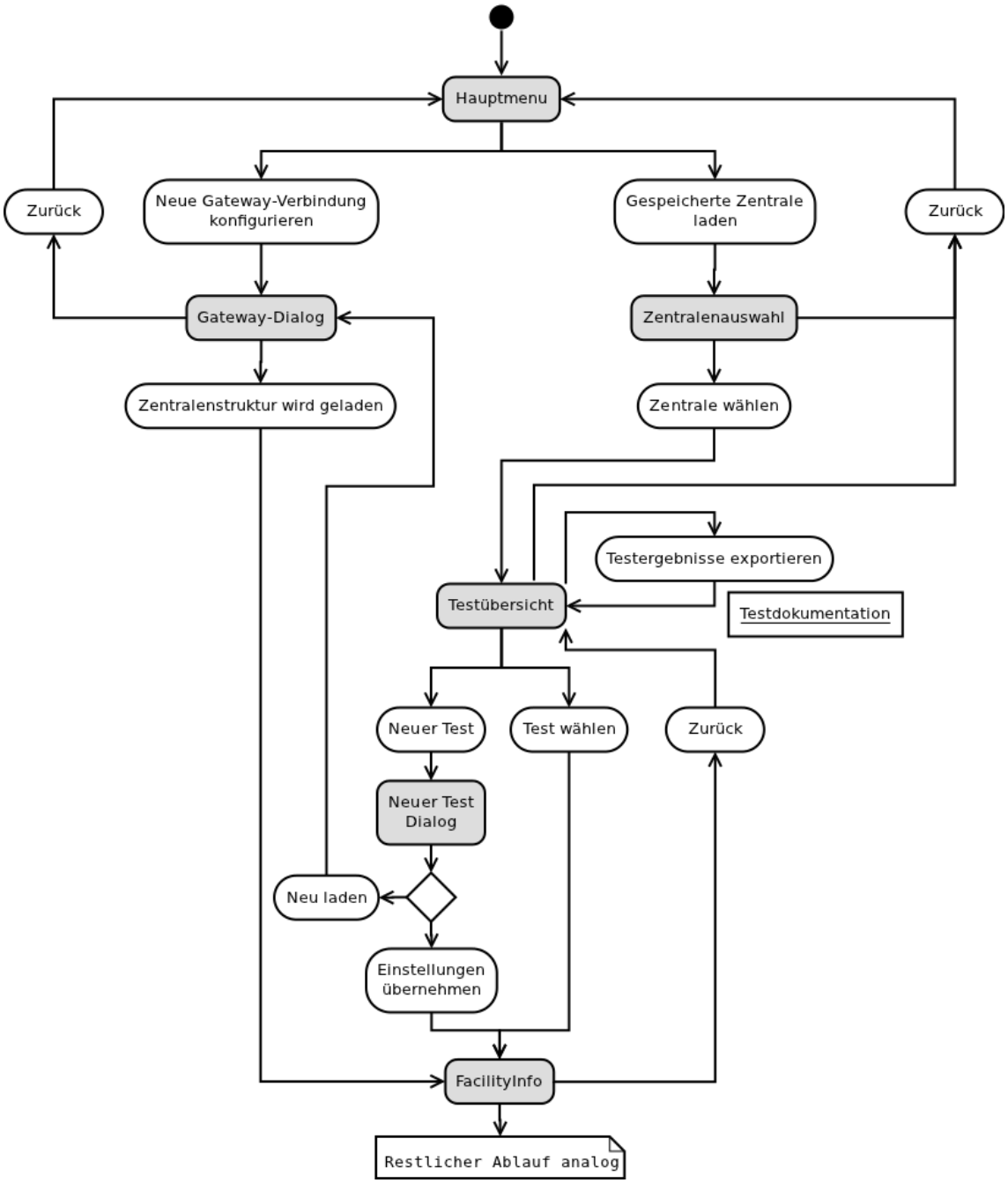


Abbildung 4.41: Flussdiagramm der im Prototyp implementierten GUI-Navigation

Wie in der obigen Abbildung zu sehen ist, unterscheidet sich die implementierte Variante vor allem dadurch von der entworfenen Lösung, dass zwischen dem Hauptmenu und der Zentralenübersicht ("FacilityInfo") nicht eine komplette Anlage geladen wird. Ein weiterer Unterschied ist, dass eine neue Zentrale ohne unnötigen Umweg über den Bildschirm "Testübersicht" (Abb. 4.45) erfasst werden kann und sogleich im Bildschirm "FacilityInfo" angezeigt wird. Dies macht Sinn, weil im Bildschirm "Testübersicht" für eine neue Zentrale

sowieso noch kein Test vorhanden ist und die einzige mögliche Handlung wäre, einen Test zu eröffnen. Also kann dies gleich in einem Schritt erledigt werden.

Wird aus dem Hauptmenu (Abb. 4.42) ausgewählt, eine neue Zentrale zu erfassen, so muss neben dem zu speichernden Namen, der IP- und der Port-Adresse des Gateways auch die BACnet-Id der zu ladenden Zentrale angegeben werden (Abb. 4.43).

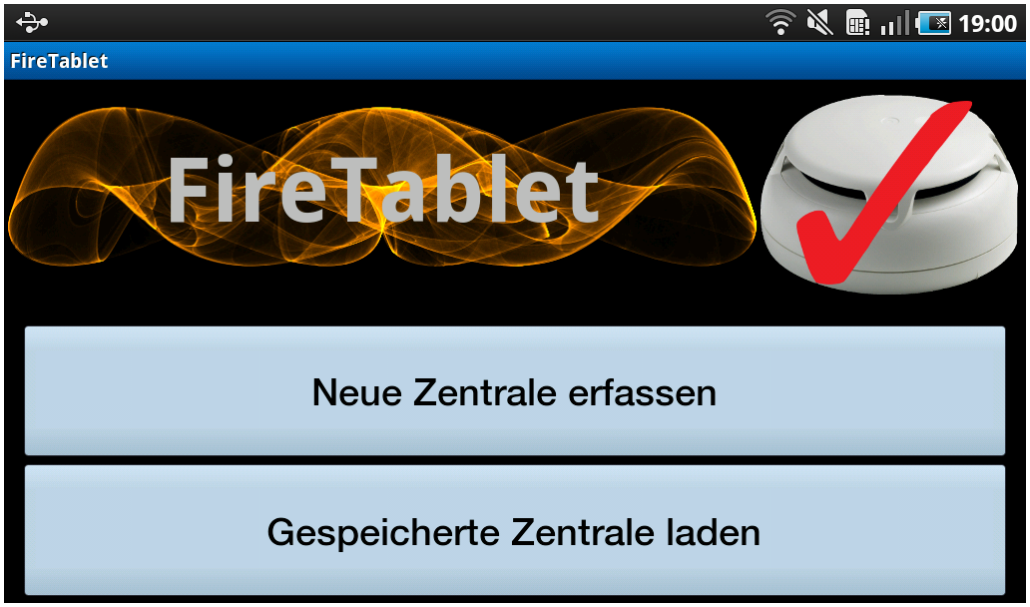


Abbildung 4.42: Hauptmenu

Abbildung 4.43: Eingabemaske für Verbindungseinstellungen zum Gateway

Möchte der Benutzer eine bereits gespeicherte Zentrale laden, erscheint nach Klick auf

den entsprechenden Knopf im Hauptmenu eine Auswahlliste (Abb. 4.44). Darin sind alle bekannten Zentralen namentlich aufgeführt. Nach der Auswahl einer Zentrale wird zum Bildschirm "Testübersicht" gewechselt und darauf alle bereits zu dieser Zentrale erfassten Tests mit Erfassungsdatum und Anzahl enthaltener Geräte angezeigt (Abb. 4.45).

Abbildung 4.44: Zentralenauswahl

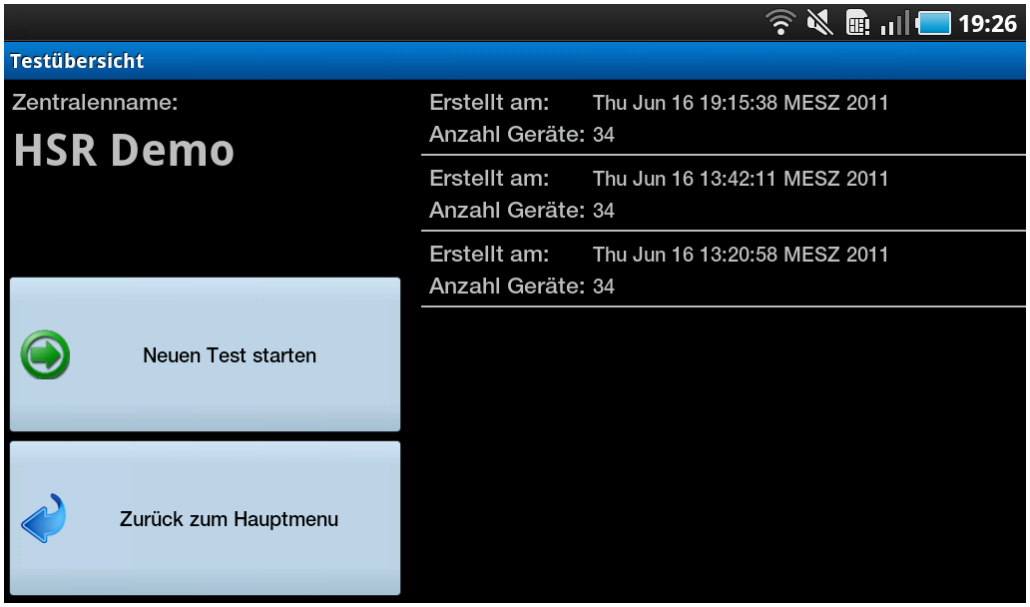


Abbildung 4.45: Bildschirm Testübersicht

Klickt der Benutzer auf den Knopf "Neuer Test", wird er zuerst gefragt, ob er die entsprechende Zentralenstruktur vom Gateway neu laden möchte oder ob sie vom letzten vorhandenen Test kopiert werden soll (Abb. 4.46). Wählt er ersteres, erscheint der bere-

its vom Hauptmenu bekannte Dialog mit den Einstellungen des Gateways (Abb. 4.47). Allerdings sind diesmal die Felder anhand des letzten Zentralentests bereits ausgefüllt, der Dialog dient zur Kontrolle und allfälligen Korrektur der Angaben.
Wählt der Benutzer, dass der letzte vorhandene Test kopiert werden soll, erstellt die Applikation eine Kopie jener Testinstanz, löscht alle darauf erfassten Mängel, setzt den Status aller Geräte wieder auf "unknown" und wechselt in die Ansicht "FacilityInfo" für diesen Test.

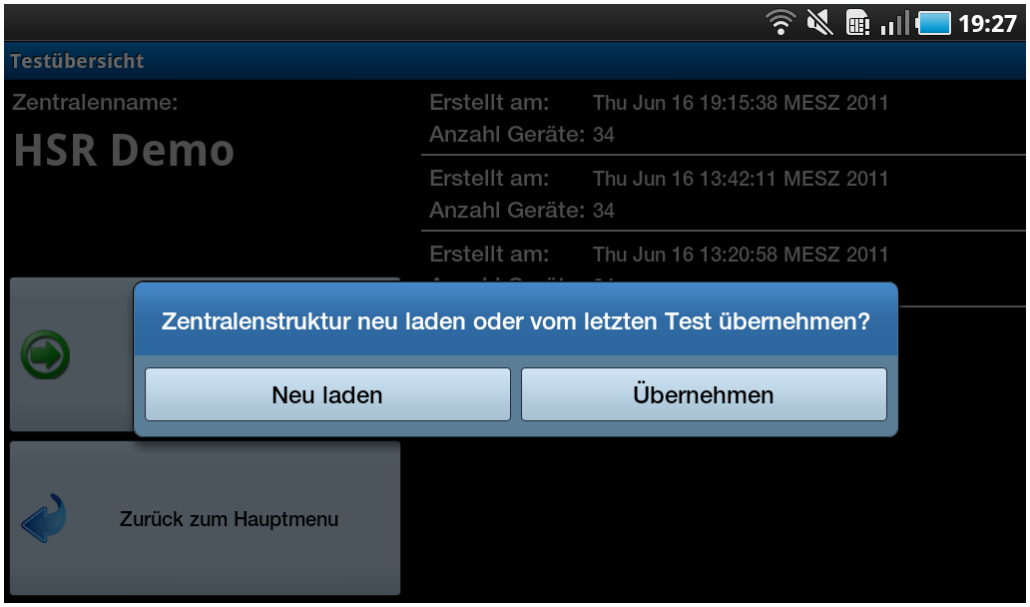


Abbildung 4.46: Dialog nach Klick auf "Neuer Test"

Abbildung 4.47: Eingabemaske nach Klick auf "Neu laden"

Soll ein alter Test betrachtet werden, kann dieser durch einen Klick auf den entsprechen-

den Listeneintrag aufgerufen werden. Es erscheint die im Kapitel 4.3.1 bereits beschriebene Zentralenübersicht. Der weitere Arbeitsablauf ab diesem Bildschirm ist identisch mit dem dort aufgeführten.
Durch einen langen Klick wird ein Kontextmenu angezeigt mit den Einträgen "Test löschen" und "Daten exportieren" (Abb. 4.48). Wählt der Benutzer "Test löschen" aus, wird nach einem Bestätigungsdialog wie erwartet der angewählte Test gelöscht. Das bedeutet auch, dass alle mit diesem Test verbundenen Mängel gelöscht werden.
Durch den Befehl "Daten exportieren" werden wie in Kapitel 5.1.3 die Informationen des ausgewählten Tests in zwei Dateien geschrieben, damit sie ausserhalb der Applikation weiterverarbeitet werden können.

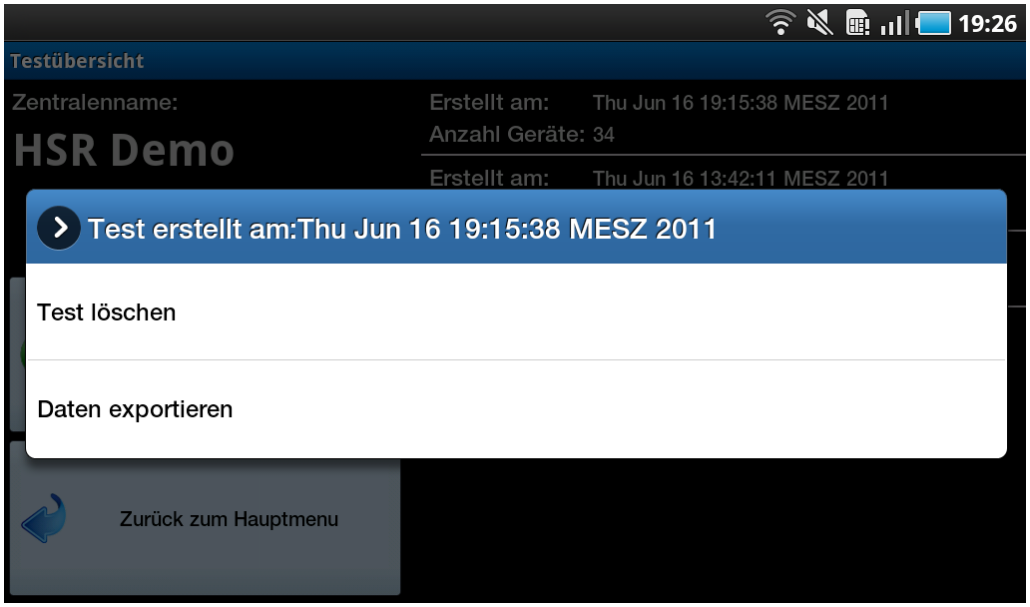


Abbildung 4.48: Kontextmenu nach Klick auf einen Test

4.3.5.1 Nicht implementierte Features

Aus Zeitgründen wurden die folgenden Use Cases nicht implementiert:

- Use Case Zentrale Funktionsprüfung: Messwerte u.Ä. erfassen
- Use Case Wartung abschliessen: Checkliste für den Servicetechniker

5 Software Entwicklung

In diesem Kapitel wird aufgezeigt, wie die Applikation softwaretechnisch funktioniert. Es wird beleuchtet, welche Klassen voneinander abhängen, und welche Designentscheide dahinter standen. Weiter wird hier auf die genauen Abläufe der einzelnen Use Cases eingegangen und diese vom technischen Standpunkt aus beschrieben. Da auf die physische Architektur und das Zusammenspiel der verschiedenen Geräte schon im Kapitel 2 (Technische Rahmenbedingungen) eingegangen wurde, beschäftigt sich dieses Kapitel nur noch mit der Software, welche auf dem Tablet abläuft.

5.1 Design & Architektur

Autor: Daniel Bobst, Samuel Hüppi

Das Design und die Architektur der Software haben sich vor allem aus der Struktur der Brandmeldezentrale ergeben. Es musste ein Konzept sein, mit dem man die Baumstruktur der Zentrale abbilden, sowie Fehler zu jedem Gerät erfassen konnte. Im Folgenden wird erklärt welche Objekte zu welchem Zweck eingeführt wurden, und wie die Software modular aufgebaut ist. Das Domainmodell dient dazu, die Objekte, mit denen im Programm gearbeitet wird, in sinnvolle Zusammenhänge zu setzen.

5.1.1 Domainmodell

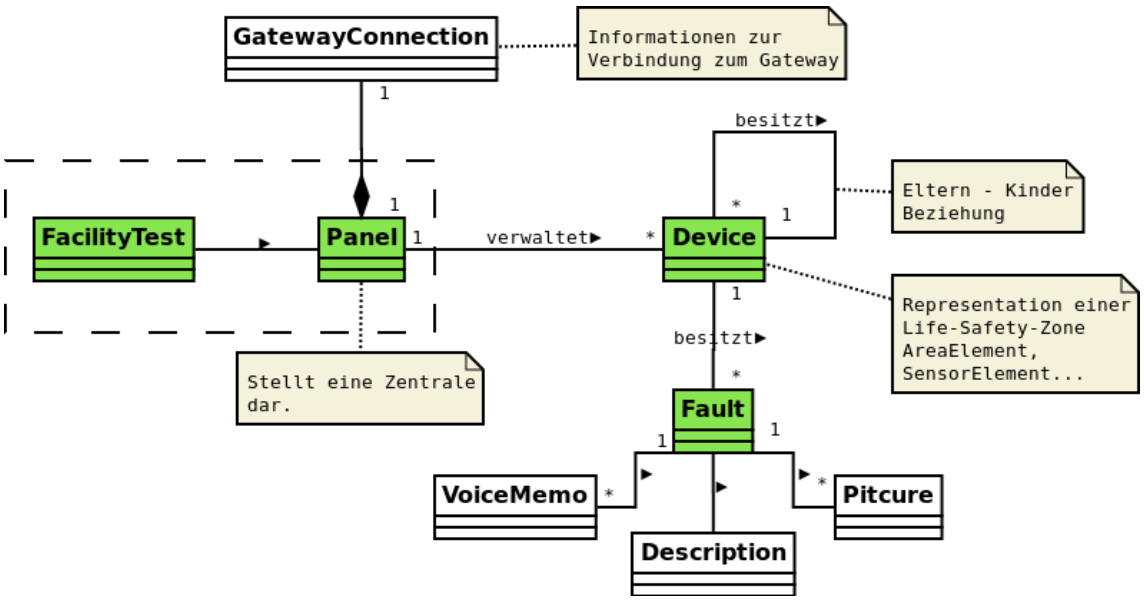


Abbildung 5.1: Das Domainmodell zeigt die grundlegende Struktur der Software

5.1.1.1 Objekte

- **FacilityTest** stellt einen Testdurchgang für die gesamte Zentrale dar. Es könnte z.B. eine jährliche Wartung oder eine Inbetriebnahme sein.
- **Panel** bildet die gesamte Baumstruktur einer Brandmeldezentrale ab und kennt die einzelnen *Devices*. Allerdings weiss das *Panel* nicht, welches *Device* welche Kind- oder Elternelemente hat. Für jeden neuen Test wird die gesamte Anlagestruktur vom Gateway geladen. Dies bedeutet, dass jedes *Panel* auch einen Zentralen-Test darstellt, der mit einem Erstellungsdatum unterschieden wird. Für das Domainmodell ist die Trennung von *Panel* und *FacilityTest* aber relevant, da hier die Realität abgebildet wird.
- **Device** repräsentiert ein Objekt in der Baumstruktur einer Zentrale, sogenannte Life-Safety-Zones. Jedes Gerät, jede Zone, oder sonst ein Objekt in einer Brandmeldezentrale wird auf einer Life-Safety-Zone abgebildet. Die für FireTablet relevanten Life-Safety-Zones sind:
  - PanelFc2020Elem
  - AreaElem
  - SectionElem
  - ZoneAutomationElem, ZoneManualElem
  - Alle SensorElemente

  Jedes *Device* weiss, ob es fehlerhaft ist oder bereits getestet wurde.
- **Fault** speichert alle möglichen Arten der Fehlererfassung. Es können Voicememos, Bilder und Textbeschreibungen hinzugefügt werden.

5.1.2 Architekturübersicht

Dieser Abschnitt erklärt die Verwendungszwecke der verschiedenen Pakete und zeigt die Abhängigkeiten auf. Jeder Paketname beginnt eigentlich mit hsr.ifs.firetablet. Die Einteilung in die einzelnen Pakete wurde nicht von Anfang an fixiert, sondern entstand erst durch den Entwicklungsprozess. Gewisse Klassen wurden erst am Schluss bei der Analyse mit Structure 101 in das definitive Paket eingeteilt. Structure 101 ist eine Codeanalyse Software, die z.B. zirkuläre Abhängigkeiten von Paketen aufzeichnet. [Structure101] Wie unten im Bild gezeigt wird, waren zu viele Abhängigkeiten ein Problem. Durch das Analysewerkzeug wurden wir auf die Klasse *GatewayConnection* aufmerksam, die im Paket Network war. Rein von der Logik her würde die Klasse aber ins Domainpackage gehören, weil es eigentlich eine reine Datenklasse ist. Als positiver Nebeneffekt hat das Verschieben geholfen, die Struktur der Paketverknüpfungen zu entflechten.

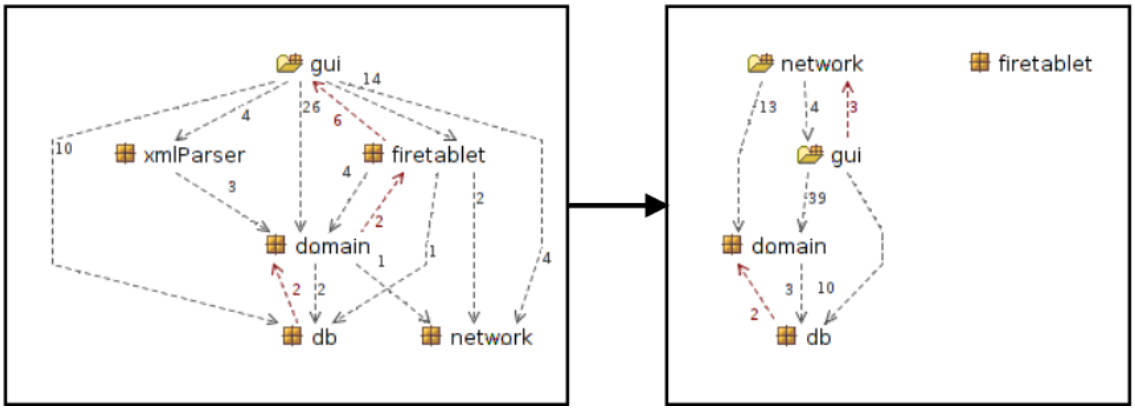


Abbildung 5.2: Optimierung durch Structure101

Im Bild unten ist die finale Struktur mit den Abhängigkeiten unter den Paketen gezeigt.

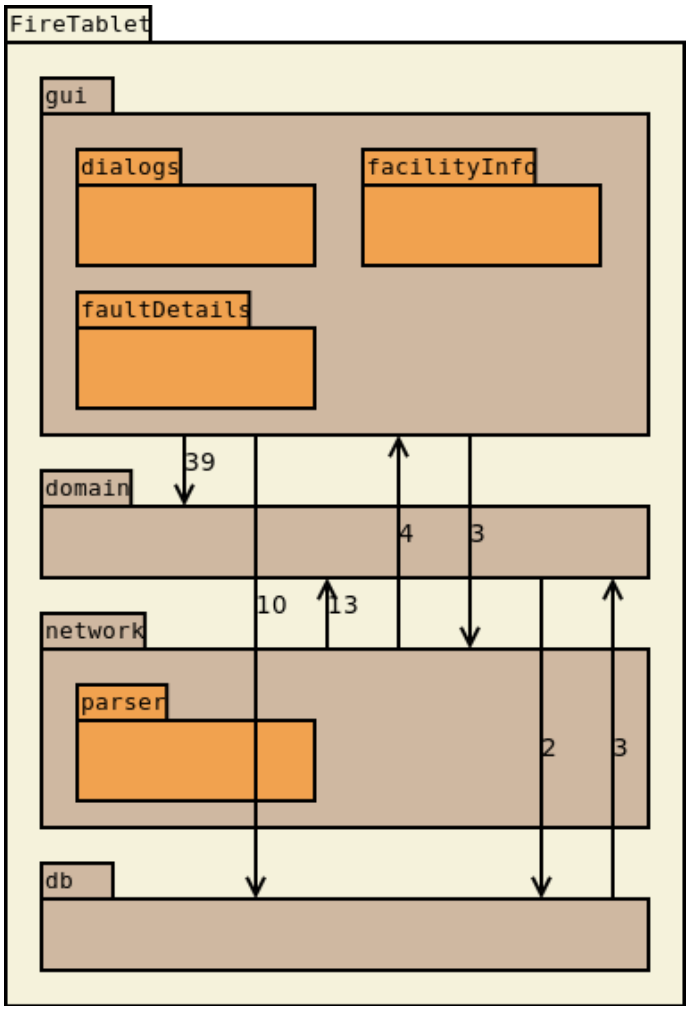


Abbildung 5.3: Paketübersicht mit eingezeichneten Abhängigkeiten

Die Aufgaben der in den Paketen gruppierten Klassen unterteilen sich wie folgt:

- **Gui**
  Das Paket beinhaltet vor allem Activities und jene Klassen, die zur Unterstützung benötigt werden. Weil Activities viel mehr als reine Gui-Klassen sind, und z.B. in der onPause() Methode eigenständig Daten persistieren müssen, haben die Activities mit allen Paketen Abhängigkeiten. Im Paket sind drei weitere Unterpakete enthalten. Diese gruppieren die Klassen des Gui weiter entsprechend ihrer speziellen Funktion. Alles bezüglich der Mangelerfassung befindet sich im Paket "faultDetails". Diverse Dialoge sind im entsprechenden Paket untergebracht, und das Paket "facilityInfo" macht den Rest des Gui aus.

- **Domain**
  Die Brandmeldezentrale wird mit den Klassen in diesem Paket abgebildet.

- **Network**
  Das Paket Network enthält die Netzwerkschicht. Da alle Klassen in diesem Paket *AsyncTasks* sind, besteht eine Abhängigkeit mit dem Gui-Paket über Domain hinweg. *AsyncTasks* führen viel Funktionalität direkt auf der *Activity* aus, beziehungsweise im gleichen Thread wie die Activity läuft. Damit greift jeder *AsyncTasks* auch wieder auf die *Activity* zu, die ihn gestartet hat. Das Paket Parser beinhaltet alle für das Parsen von XML nötigen Teile.

- **Db** Diese Paketinhalte verwalten den Zugriff zur Datenbank.

5.1.3 Klassendiagramm

In diesem Abschnitt erhält der Leser einen Überblick über die verwendeten Klassen. Auf die wichtigsten Klassen wird speziell eingegangen und ihre Funktion erklärt. Alle grünen Klassen sind Activities, die einen Bildschirm darstellen, wie sie im Abschnitt 4.3 (Finaler Entwurf) mit Screenshots gezeigt werden. Weisse Klassen im selben Paket werden von den grünen verwendet. Da die Funktion der grün markierten Klassen schon im Kapitel 4 (4) eingehend erklärt wird, verzichten wir an dieser Stelle auf eine detaillierte Erklärung. Unten abgebildet ist aber ein Mapping der Activityklassen zu den entsprechend benannten Bildschirmen. Für die restlichen Klassen gibt es unterhalb der Übersicht detaillierte Beschriebe.

Bildschirm	**Activity**
Hauptmenu	MainMenu
Gateway-Dialog	IpInputDialog
Zentralenauswahl	LoadFacilityFromDB
Testübersicht	TestOverview
FacilityInfo	FacilityInfo
Eventbehandlung	GatewayEvent
Mangelerfassung	FaultDetails

Tabelle 5.1: Mapping der Activities auf die verschiedenen Bildschirme

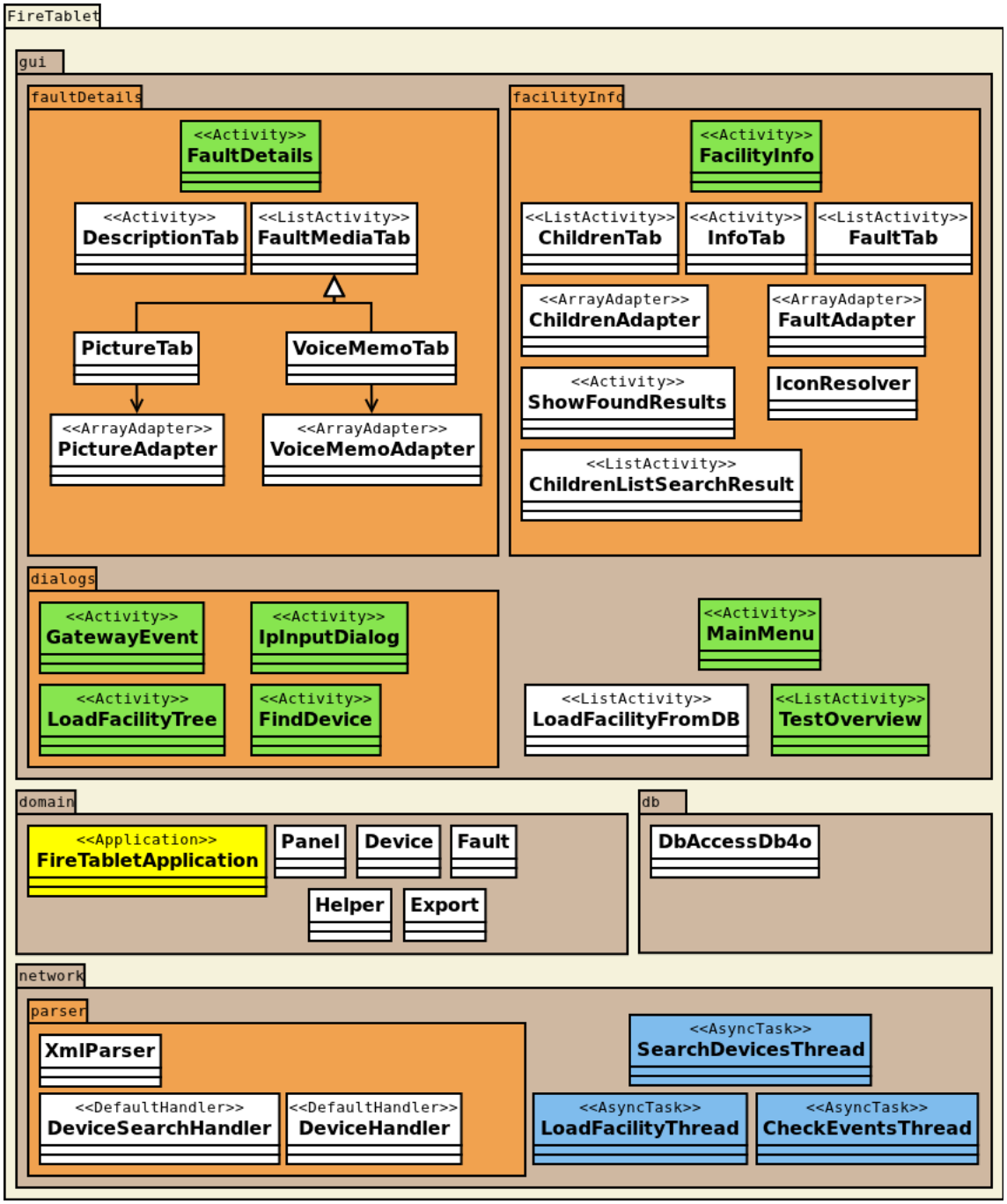


Abbildung 5.4: Klassendiagramm ohne Klassenabhängigkeiten

- **FireTabletApplication**
  Diese Klasse stellt den Application- Context für FireTablet dar. Die Klasse wird von Android bei jedem Start des Programms instantiiert und verwaltet. Sie dient als eine Art Singleton und kann von jeder Activity mit *getApplicationContext()* bezogen werden. Beim Application Context können folgende Artefakte abgerufen werden:
  - **Datenbank**
    Die Datenbankverbindung wird beim Erstellen des ApplicationContext erstellt

und kann mit *public DbAccessDb4o getDb()* vom Context bezogen werden.

– **Panel**
Das Panel muss manuell beim Context gesetzt werden, sobald ein Panel verfügbar ist. Dazu gibt es zwei Anlässe. Entweder wird ein Panel von der Datenbank geholt oder das Panel wird neu vom Gateway geladen.

- **Panel**
Ein Panel ist das Rootelement der gesamten Zentralenhierarchie und speichert alle Objekte in Form von Devices in eine *HashMap*, mit welcher auf alle Objekte der Zentrale zugegriffen werden kann. Als Keys dienen die OBJECT_IDENTIFIER jedes Objektes in der Zentrale. Weiter verwaltet das Panel die Verbindungsinformationen zum Gateway und besitzt einige Operationen um Devices zu verwalten.
Wie man im Domainmodell erkennt, ist vor dem *Panel* ein FacilityTest vorgeschaltet. Dieses Objekt hat in der Klassenstruktur keine Realisierung bekommen, weil ein Facilitytest mittels eines *createDate* im *Panel* abstrahiert wurde. Demzufolge muss bei jedem Facilitytest ein neues *Panel* erstellt und vom Gateway geladen werden. Verschiedene *Panels* werden durch ihr *createDate* unterschieden und repräsentieren auch unterschiedliche Tests. Der Vorteil neben dem einfacheren Klassendiagramm hat auch noch eine praktische Natur. Es ist nicht möglich auf veralteten Informationen einen Test durchzuführen, sondern die Zentrale muss immer neu geladen werden.

- **Device**
Wie bereits in Abschnitt 5.1.1 beschrieben ist, stellt ein *Device* ein Objekt im Baum der Zentrale dar. Jedes *Device* besitzt ein *Bundle* mit den im Listing ersichtlichen Keys. Ein *Bundle* ist eine Art HashMap, mit dem Vorteil, dass man ein *Bundle* mit einem Intent von einer zur nächsten Activity schicken kann. *Bundles* sind also serialisierbar.
Genauere Informationen zur Bedeutung dieser Schlüssel können dem Teil 2.2 Analyse Gateway entnommen werden.

```
private Bundle attributes;

 String OBJECT_IDENTIFIER = "object-identifier";
 String OBJECT_NAME = "object-name";
 String DESCRIPTION = "description";
 String DEVICE_TYPE = "device-type";
 String PROFILE_NAME = "profile-name";
 String MAINTENANCE_RQIRED =  "maintenance-required";
 String NOTIFICATION_CLASS = "notification-class";
 String NOTIFY_TYPE = "notify-type";
 String EVENT_MESSAGE_TEXT = "isa-event-message-texts";
```

Listing 5.1: Keys für Attribute von Device. Alle Key Strings sind public static final.

Ein *Device* ist aber keine reine Datenklasse, sondern kann seinen Zustand intern mit der Enumartion *status* verwalten. Wenn zum Beispiel ein Fehler erfasst werden soll, kann mittels *public void registerFault(Fault fault)* ein *Fault* übergeben werden. *Device* passt anschliessend seinen internen Status entsprechend dem jetzigen Status an und schreibt den *Fault* auf die Datenbank. Weitere Details zu den verschiedenen

Status befinden sich im Abschnitt 4.3 Gui Design.
Diese Architektur wurde nötig, weil *Faults* nicht in *Devices* gespeichert werden, wie das Domainmodell vermuten lässt, sondern nur in der Datenbank. Von dort können Fehler mittels OBJECT_IDENTIFIER und *createDate* gesucht werden. Um viele *Devices* in einer scrollbaren Liste anzuzeigen braucht es aber sehr schnelle Abfragen auf die *Devices*. Dies ist aber nicht möglich wenn für jedes *Device* mindestens ein Datenbankzugriff ausgeführt werden muss, um abzuklären welcher Status ein *Device* besitzt. [Turbo-charge your UI] Somit verwaltet ein *Device* seinen Status selber und ist gleichzeitig auch verantwortlich, die *Faults* auf der Datenbank zu erstellen oder zu löschen. Damit entsteht der Vorteil diese Funktionalität zu kapseln. Man muss nicht immer daran denken, wenn man den Status eines *Devices* ändert, auch die Datenbank zu informieren oder umgekehrt.

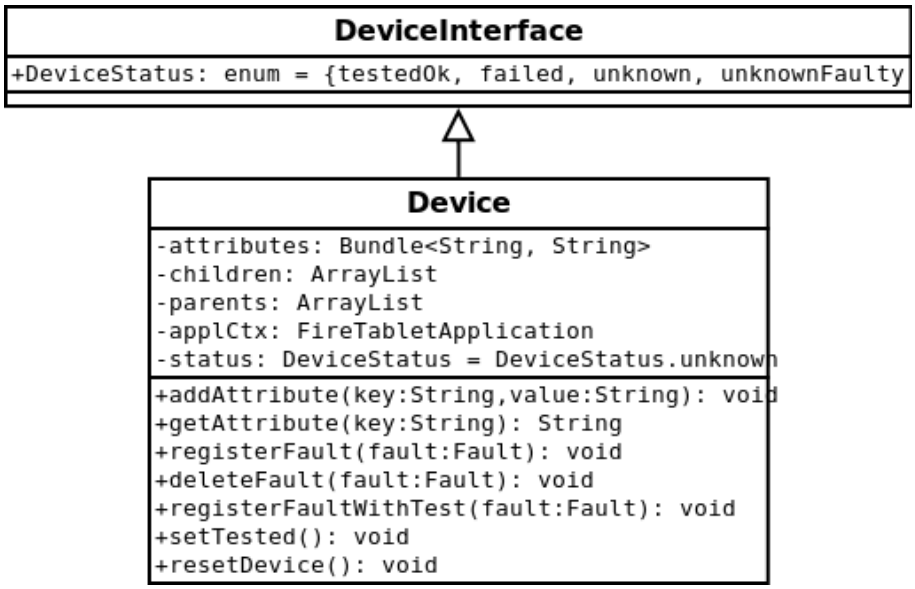


Abbildung 5.5: Die Klasse *Device* mit den wichtigen Methoden

Device implementiert das Interface *Parcelable*, damit ein *Device* serialisierbar wird und mit *Intents* verschickt werden kann.

- **Export**
  Die Klasse dient dazu, die in einem Test zusammengetragenen Daten zu sammeln und daraus Dokumente zu generieren, damit die Testergebnisse ausserhalb der Applikation weiterverarbeitet werden können. In der vorliegenden Implementation handelt es sich bei den Dokumenten der Einfachheit halber um zwei csv Dateien. Die eine enthält den Status der verschiedenen Geräte der Zentrale, die andere Angaben zu erfassten Mängeln.

```
object-identifier;objectname;description;device-type;status;
    nbrOfFaults
```

Listing 5.2: Exportierte Werte von Geräten

```
object-identifier;faultId;faultCreatedDate;description;
   nbrofVoiceMemos;voiceMemoFilenames;nbrOfPictures;
   pictureFilenames
```

Listing 5.3: Exportierte Werte zu Mängeln

- **Fault**
  Diese Klasse ist eine reine Datenklasse, die die verschiedenen Attribute eines Fehlers speichern kann. *Faults* können anschliessend auf der Datenbank persistiert werden.

- **DbAccessDb4o**
  Wie es der Name schon sagt, wird mittels dieser Klasse auf die Datenbank zugegriffen. Dazu stehen einige Methoden zur Verfügung, um zum Beispiel ein *Panel, Faults*, oder *VoiceMemos* abzulegen und wieder zu finden. Als Datenbank wird Db4o verwendet. [Db4o]

- **AsyncTask**
  Die blau markierten Klassen mit dem Stereotyp AsyncTask im Package "network" sind die drei Threads die FireTablet benötigt, um das Benutzerinterface ansprechbar zu halten, während länger dauernde Aktionen ausgeführt werden. Dies ist der Fall, wenn die Zentralenstruktur vom Gateway abgerufen wird oder falls das Tablet auf ankommende Events vom Simulationserver wartet. *AsyncTasks* sind sehr eng mit den Activities gekoppelt. Es wird nur die *doInBackground()* Methode in einem seperaten Thread ausgeführt. Alle anderen Methoden laufen im gleichen Thread wie das Benutzerinterface. Weitere Informationen zu AsyncTasks können dem Artikel Painless Threading entnommen werden. [Painless Threading]

5.1.4 Sequenzdiagramme

Nachdem ein Überblick über die Klassen von FireTablet vermittelt wurde, wird anschliessend auf ausgewählte Abläufe innerhalb der Applikation eingegangen. Auch dabei werden die *Acivities* verwendet, dessen Funktionalität und Zweck im Abschnitt 4.3 Design Gui erklärt wurde.

5.1.4.1 Use Case 01 Gateway verbinden

Ein Panel kann aus zwei Quellen erstellt werden. Entweder es wird aus der Datenbank wieder erstellt oder FireTablet lädt die Anlagestruktur neu vom Gateway und erstellt einen neuen Test. Auf den nächsten zwei Bildern sind diese beiden Abläufe dargestellt.

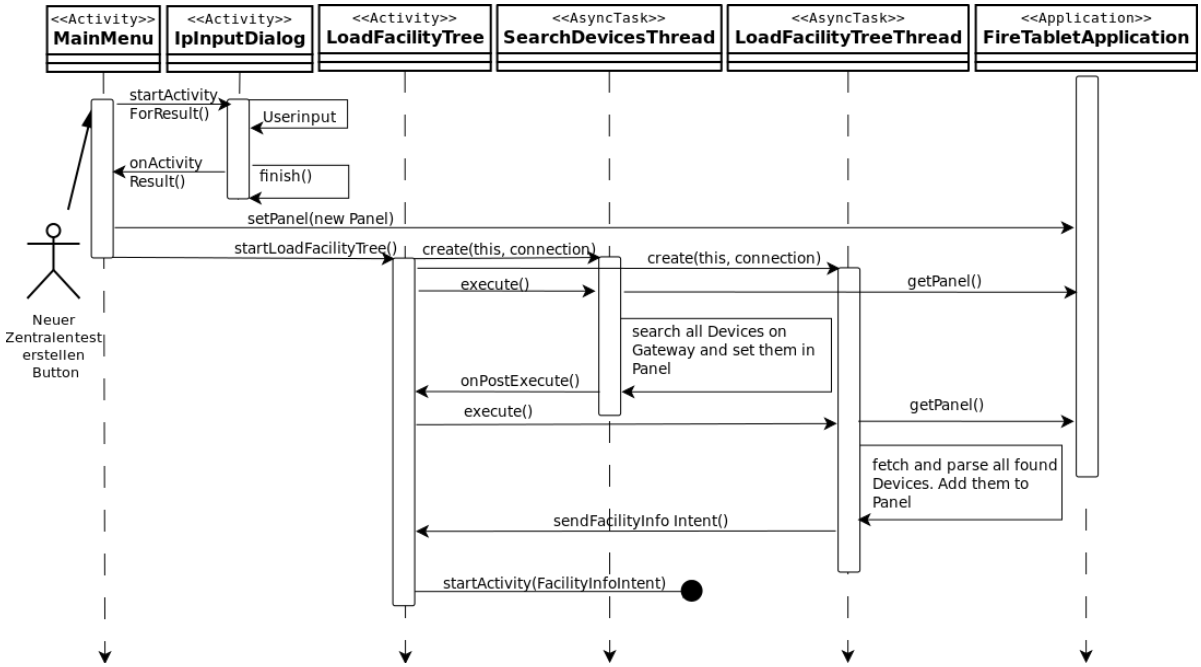


Abbildung 5.6: Zentralenstruktur wird vom Gateway abgerufen, geparsed und als *Panel* im *ApplicationContext* gespeichert.

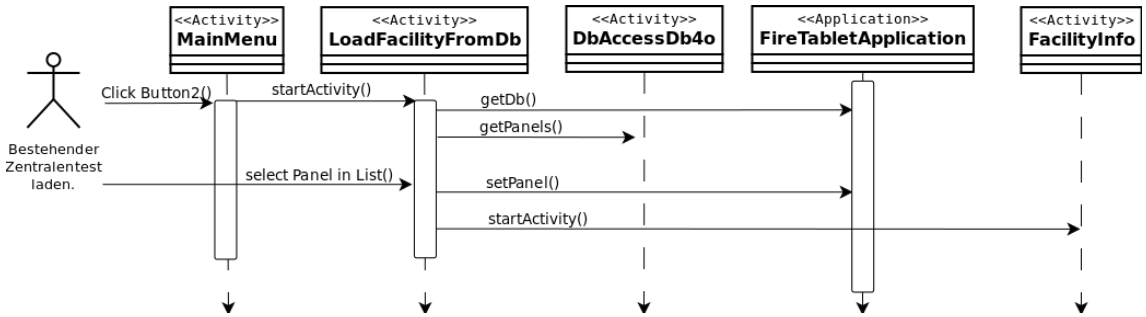


Abbildung 5.7: Zentralenstruktur wird von der Datenbank geholt und als *Panel* im *ApplicationContext* gespeichert.

Wenn man diese beiden Sequenzdiagramme betrachtet, stellt sich die Frage, warum das Panel im *ApplicationContext* gespeichert wird und nicht bei der Erstellung als Parameter an die neuen Activities übergeben wird. Der Kritikpunkt dieser Lösung ist die Klasse *FireTabletApplication*, die ein Singleton mit all seinen Nachteilen ist. [APF Foliensatz 14, S. 17] Tatsächlich gibt es wegen dem *ApplicationContext* auch Probleme, zum Beispiel beim Erstellen von UnitTests, siehe Abschnitt 5.2.3 Unit Testing. Folgende Gründe haben aber trotzdem zur Verwendung des Singleton geführt:

1. Bei Android wird der Singleton *ApplicationContext* genannt und vom Android System verwaltet.

2. Man hat nicht von überall Zugriff auf diesen *Context*, sondern nur aus *Activities* heraus.

3. Eine Applikation ist eine Gruppe von Activities, die nur lose miteinander verbunden sind. Will man Daten von einer Activity zur anderen schicken, wird dies mit *Intents* getan. *Intents* erlauben aber nur primitive Datentypen oder jene die das Interface *Parcelable* implementieren.

5.1.4.2 Use Case 02 Sensortest durchführen

Das nächste Sequenzdiagramm beschreibt den Vorgang nachdem in *FacilityInfo* der Testmodus gestartet wurde. Zuerst wird ein *AsyncTask* erstellt, der wiederum einen Thread startet, der anschliessend auf eingehende Events vom Event Simulationsserver hört. Falls der Server ein Event schickt, wird dies mit *publishProgress*() dem *AsyncTask* mitgeteilt, der danach wieder im Userinterfacethread die *GatewayEvent Activity* aufruft.

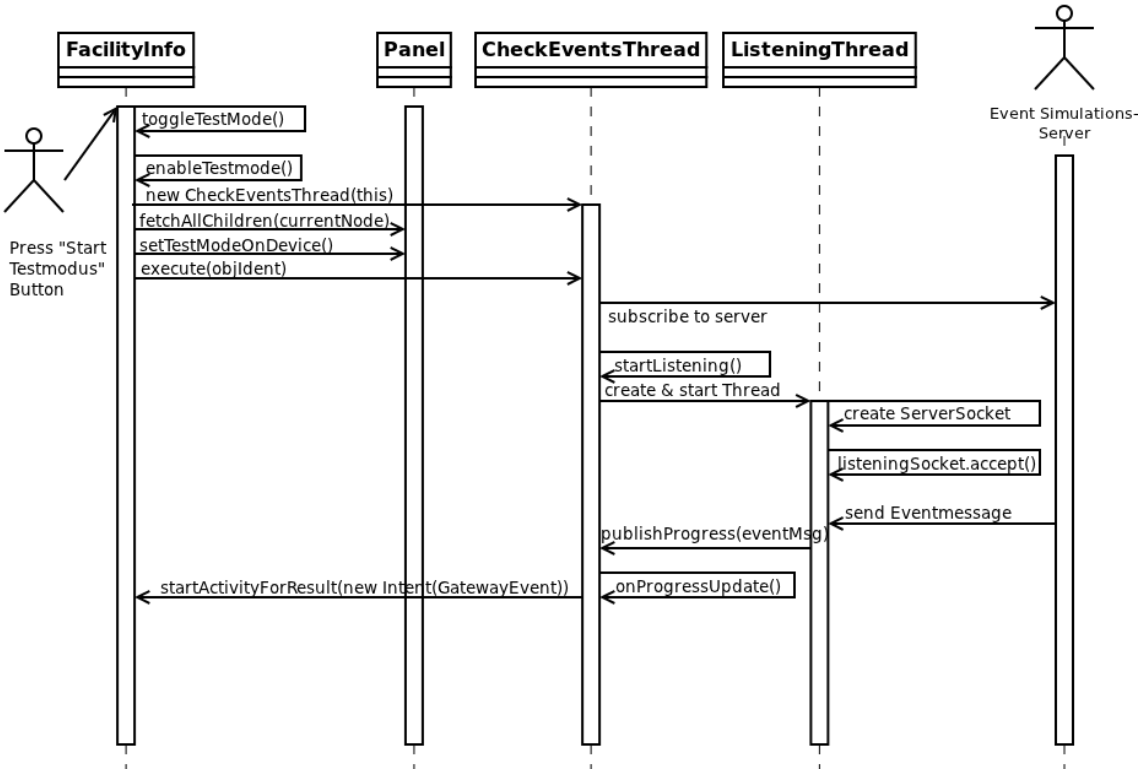


Abbildung 5.8: Prozess, nachdem "Start Testmodus" geklickt wurde.

5.2 Unit Testing

Autor: Samuel Hüppi

In der heutigen Zeit gehört Test Driven Development (TDD) nicht mehr nur zum guten Ton bei einem Softwareprojekt, sondern ist absolute Pflicht, wenn man qualitativ hochstehende Software entwickeln will. Aus diesem Grund stellt auch das Android Framework eine vollständige Testumgebung basierend auf JUnit bereit. [JUnit] Wie sich aber im Verlauf der Arbeit gezeigt hat, sind die Testklassen von Android noch nicht über alle Zweifel erhaben (Siehe Kap. 5.2.3, S. 85). Dafür gibt es einige Frameworks, die zum Teil Abhilfe schaffen, wie z.B. RoboGuice und Robotium. Auf diese wird weiter unten nochmals einge-

gangen. Am Anfang dieses Kapitels geht es darum, einen Überblick zu den verschiedenen Android Testklassen, welche eingesetzt wurden, zu erhalten. Im Kap. 5.2.3 Anwendung und Beispiele werden anschliessend ein paar Beispiele von Tests erklärt, die für das Projekt geschrieben wurden.

5.2.1 Android Test Klassen

Android stellt eine Auswahl von verschiedenen Testsuperklassen zur Verfügung. Je nachdem was genau getestet werden soll und wie stark die Tests mit androidspezifischen Komponenten interagieren, eignet sich eine Testsuperklasse besser als die andere. Im Buch Android 2 werden drei verschiedene Arten beschrieben um eine Applikation zu testen. [Android 2, S.373]

- **Test der Gesamtanwendung**
  Um komplexe Vorgänge, bei denen mehrere Activities zusammen spielen, zu testen, verwendet man häufig keine automatischen Tests, sondern Menschen. Der Aufwand wäre zu gross um Applikationen gesamtheitlich zu testen, denn die Tests müssten einerseits auf wichtige Systemereignisse oder Multitasking reagieren, andrerseits trotzdem so flexibel sein, dass man die Tests nicht bei jeder Änderung der Zielanwendung anpassen muss. Falls man doch mehrere Actitivities zusammen testen will, sollte man *ActivityInstrumentationTestCase2* verwenden.

- **Oberflächentests**
  Diese Tests überprüfen einen Handlungsablauf auf dem Bildschirm. Meistens beschränkt sich ein solcher Test auf eine Activity, so dass man auch Activity-Test sagen könnte. Beispielsweise wird für den UseCase "01 Mit Gateway verbinden" ein Oberflächentest verwendet. Für Oberflächentests eignet sich *ActivityUnitTestCase* am besten.

- **Modultest**
  Modultests sind von der Granularität her die feinsten Tests. Bei diesen Tests geht man nicht auf den Lebenszyklus von Androidkomponenten ein sondern testet normale Javaklassen, die zum Beispiel von Activities gebraucht werden. Für Modultests verwendet man vor allem *ActivityUnitTestCase* und *AndroidTestCase*.

Die folgenden Testklassen sind im Androidframework vorhanden. Je nachdem, was man testen will, haben die Klassen ihre Vor- und Nachteile. Am Schluss dieses Teils wird Robotium genauer vorgestellt, da dieses Framework eine grosse Vereinfachung für Oberflächentests darstellt.

5.2.1.1 AndroidTestCase

AndroidTestCase ist die einfachste Form eines Tests. Die Klasse eignet sich um Zugriff auf die Attribute und Funktionen einer Activity zu bekommen. Obwohl diese Klasse keine Androidkomponente ist, benötigt sie doch einen Context, damit sie auf Informationen des Systems und der Applikation zugreifen kann.

5.2.1.2 InstrumentationTestCase

Wie in der Android-Referenz im Developer Guide unter Activity Testing beschrieben wird, ist *InstrumentationTestCase* die Grundklasse, um auf Activities während einem Test Einfluss zu nehmen, und stellt drei grundsätzliche Funktionen zur Verfügung [Activity Testing].

- Es kann auf den Lebenszyklus von Activities Einfluss genommen werden. Activities können gestartet, pausiert oder zerstört werden.
- Mit Dependency Injection ist es möglich Mockobjekte wie z.B. *Contexts* oder *Application* zu injizieren.
- Benutzeraktionen können simuliert werden. Beispielsweise können Tastatureingaben gesendet oder direkt das User Interface bedient werden. Im Vergleich zu Robotium ist die Kontrolle des User Interface aber weniger benutzerfreundlich. *Instrumentation* ist die zentrale Klasse welche die Fernbedienung einer Activity kapselt.

5.2.1.3 ActivityUnitTestCase

Mit dieser Klasse ist es möglich eine Activity isoliert zu testen ohne viele Verbindungen zum Androidsystem zu haben. Durch die Isolation gibt es viele Methoden einer Activity, die nicht in einer Testumgebung aufgerufen werden sollten, da diese sonst Exceptions werfen. Genauere Details können der Android-Referenz entnommen werden. *ActivityUnitTestCase* bietet zudem die Möglichkeit einfache Mocks einzuführen. [Android Reference]

5.2.1.4 ActivityInstrumentationTestCase2

Falls mehrere Activities und das gegenseitige Zusammenspiel getestet werden wollen, eignet sich diese Testklasse. Genau wie *InstrumentationTestCase* stellt sie sämtliche Möglichkeiten um auf eine Activity Einfluss zu nehmen, zur Verfügung. *ActivityInstrumentationTestCase2* erlaubt es aber nicht Mocks einzuführen. Die Tests können also nicht von einem produktiven System getrennt werden.

5.2.1.5 Klassenübersicht

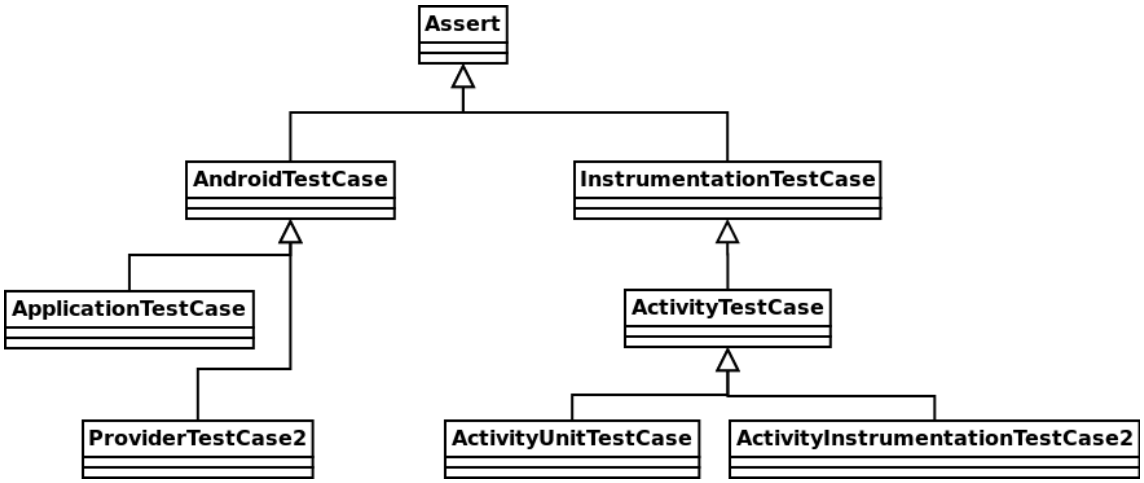


Abbildung 5.9: Klassendiagramm der Android Testklassen.

5.2.2 Robotium

Die Robotium Libraries stellen eine einfache Möglichkeit dar, um Oberflächentests oder Tests der Gesamtanwendung durchzuführen. [Robotium] Dabei bietet Robotium im wesentlichen einen Simulator an, der Eingaben eines Benutzers simuliert, zugleich aber sehr einfach zu verwenden ist. Es können beispielsweise Buttons geklickt werden, indem der identifizierende Textinhalt des Buttons der entsprechenden Simulatorfunktion übergeben wird. Weiter kann Robotium auch dazu eingesetzt werden, um Oberflächentests über mehrere Activities zu starten, was ein einfaches Testen der Gesamtanwendung ermöglicht. Um mit Robotium zu arbeiten, wird immer ein Simulator, meistens solo genannt, instantiiert indem man dem Konstruktor die Activity und die Instrumentation übergibt. Danach kann beispielsweise mit solo.clickButton(0); der erste Button im Layout geklickt werden oder mit solo.enterText(0, "Text"); ein Text in ein Textfeld eingegeben werden. Dies stellt eine enorme Vereinfachung im Vergleich zum Androidframework dar.

```
public void setUp() throws Exception {

      activity = getActivity();
        solo = new Solo(getInstrumentation(),activity );
  }
   }
```

Listing 5.4: Erstellung von solo, dem Simulator. Falls die Activity nicht schon gestartet wurde, wird dies durch die Methode getActivity() erledigt.

5.2.3 Anwendung und Beispiele

In der Praxis stellt sich das Unittesting für Android dann aber als wesentlich unkomfortabler heraus, als die Tutorials und Dev Guides von Google meinen. [Testing Tutorial]

Im wesentlichen wird das Thema durch Activities erschwert, die nicht einfach als normale Klassen instantiiert, sondern von Android aufgerufen werden. Wirklich schwierig wird es, falls man Objekte vom Application Context benötigt, welche zuerst erstellt werden müssen. So haben wir es nicht geschafft, zuverlässig zuerst den Context aufzusetzen und erst danach die *onCreate()* Methode auszuführen. Es scheint, als wäre die *Activity* mit ihrer *onCreate()* Methode, durch parallele Ausführung ständig schneller, was in *NullPointerExceptions* endet. Oft wird folgende Situationen angetroffen: Die Tests schlagen bei normaler Ausführung fehl, laufen aber im Debugging Modus erfolgreich ab. Aus diesem Grund und dem, dass Activity-Tests sehr viel Zeit benötigen, haben wir uns vor allem auf Modultests konzentriert, deren Funktionalität losgelöst von Activities getestet werden kann.
Im Folgenden werden Beispiele aus dem Testprojekt vorgestellt, die wichtige Teile des Projekts abdecken oder wo spezielle Methoden zum Einsatz kamen.

5.2.3.1 XML Handler Test

Das korrekte Verhalten der beiden SAX-Parser Handler *DeviceHandler* und *DeviceSearchHandler* wird mit den Testklassen *DeviceHandlerTest* und *SearchDeviceHandlerTest* sichergestellt. Es wird als erstes eine SAX-Parserinstanz erstellt und mittels der parse() Methode ein *InputStream* der XML enthält dem entsprechende Handler übergeben. Anschliessend kann das Resultat bei den Handlern abgeholt werden.
Das Testen der XML Handler ist sehr wichtig, weil die Übersicht in den Handlerklassen schnell verloren geht. Man will aber bei Änderungen am Handler garantieren können, dass sich der Handler noch gleich verhält.

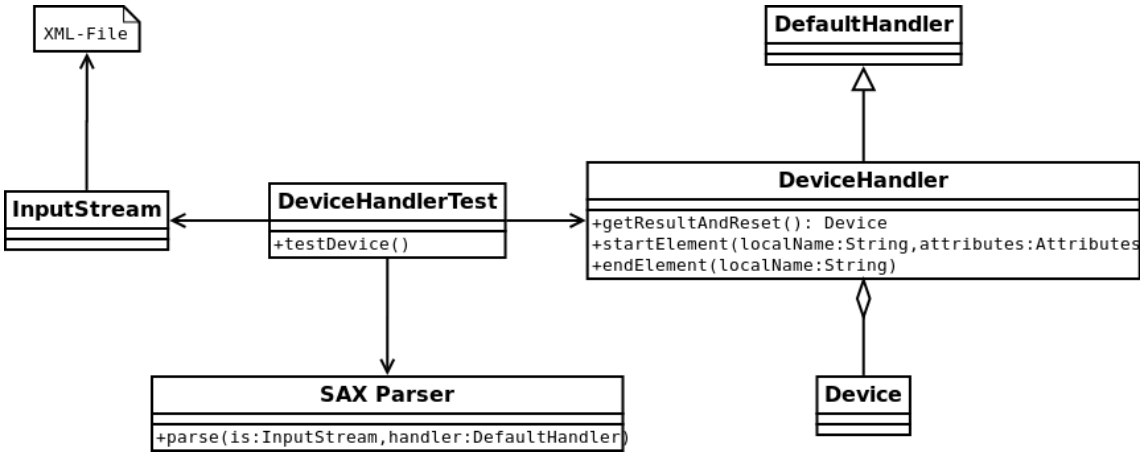


Abbildung 5.10: Testcase Übersicht von *DeviceHandlerTest*. Der *SearchDeviceHandlerTest* funktioniert analog. *Device* wird dabei durch *Panel* ersetzt.

5.2.3.2 Load Facility Test

Die Schwierigkeit beim Testen der *LoadFacilityTree* Klasse liegt bei den zwei *AsyncTasks* welche gestartet werden. Die beiden Tasks werden eingesetzt um die Kommunikation mit dem Gateway in separate Threads auszulagern und das GUI dadurch flüssig zu halten. Beim Testen entsteht dabei aber die Schwierigkeit, dass die Tests nicht auf die Threads warten und schon getestet wird bevor sie ihre Arbeit verrichtet haben. Das Resultat sind

NullPointerExceptions, da das *Panel* noch initialisiert wird.
Wir haben während der Entwicklung versucht mit Roboguice das Problem zu beheben. [roboguice] Roboguice ist ein Framework, das viele Dinge von Android vereinfachen will. Unter anderem soll es auch eine Möglichkeit geben *AsyncTasks* richtig zu testen. In der Praxis war dann aber der zusätzliche Aufwand und die Einarbeitungszeit zu gross um nur zwei *AsyncTasks* zu testen. Hinzu kam, dass wir eine andere Lösung gefunden haben auf die anschliessend eingegangen wird.
Der Lösungsansatz ist, die *doInBackground()* Methode selber aufzurufen und nicht wie üblich den *AsyncTask* mit *execute()* zu starten. Somit wird kein zweiter Thread gestartet und die Tests werden mit dem richtig initialisiertem *Panel* ausgeführt. Damit dies möglich ist, muss von *LoadFacilityThread* und *SearchDevicesThread* abgeleitet werden, um das Überschreiben von gewissen Methoden zu ermöglichen. Speziell bei dieser Lösung ist auch die *ThreadXmlParser2* Klasse. Sie imitiert einen Parser, der je nachdem welche URL übergeben wird, eine andere *InputSource* der *parse()* Funktion übergibt, ohne eine Netzwerkabfrage auszulösen. Somit kann man den Parser einfach seinen Bedürfnissen anpassen und mehrere verschiedene *Devices* parsen. Mehr zum Aufbau kann aus dem Klassendiagramm entnommen werden.

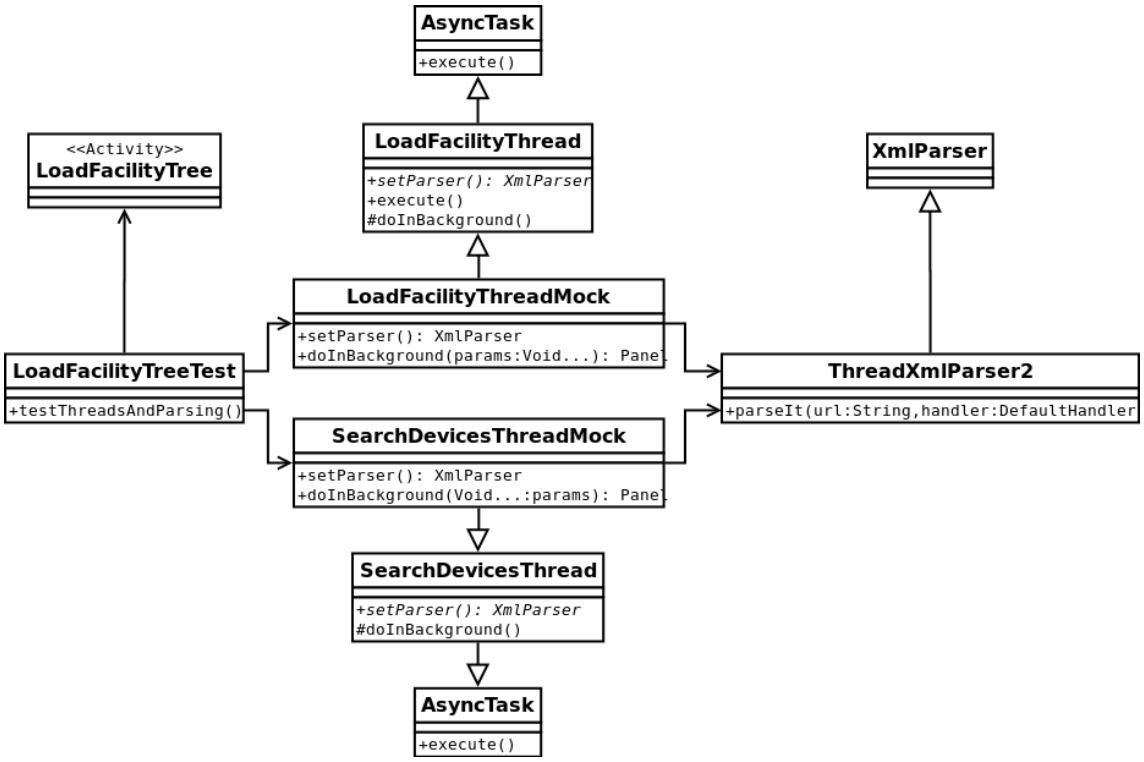


Abbildung 5.11: Testcase Übersicht
von *LoadFacilityTreeTest*. Die Thread Mocks werden in der Testklasse instanziert und durch den Aufruf von *doInBackGround() manuell gestartet*.

5.2.3.3 Oberflächentests mit Robotium

Robotium wurde nur für wenige Activity-Klassen verwendet, obwohl Robotium in der Tat die Handhabung von Oberflächentests sehr vereinfacht. Probleme traten auch hier beim

Laden des ApplicationContext auf, da dieser nicht richtig initialisiert werden konnte, bevor die Activity durch den Aufruf von *onCreate()* ins Leben gerufen wird. Oberflächentests mit Robotium sind aber auch vom Microtestingstandpunkt aus sehr kritisch zu betrachten. [Microtesting] Bei Laufzeiten von mehreren Sekunden pro Test kann kaum mehr von "Micro" gesprochen werden. Diese Art von Test eignet sich aber gut zur Sicherstellung eines Oberflächentests der ganzen Applikation in Bezug auf die Lauffähigkeit aller Funktionen. Das wäre aber ein Vorgang der nicht so häufig ausgeführt wird wie normale feingranulare Unittests. Mit Robotium wurden die Activities *MainMenu* und *FaultDetails* getestet. Anschliessend findet sich ein Beispiel eines Robotiumtests von *MainMenu*. "Solo" ist der Simulator von Robotium und wird in der *setUp()* Methode erstellt.

```
public void testInputIp() throws Exception{
      solo.clickOnButton(0);
      solo.assertCurrentActivity("IpInputDialog
            Activity erwartet", IpInputDialog.class);

      solo.enterText(1, "192.168.1.300");
      assertTrue(solo.getButton(0).isEnabled());

      solo.clickOnButton(0);
      solo.assertCurrentActivity("IpInputDialog
            Activity erwartet", IpInputDialog.class);
      assertTrue(solo.searchText(activity.getString(
            R.string.inputIp_dialog_wrong_ip_toast)));
      solo.clearEditText(1);

      solo.enterText(1, "192.168.1.1");
      solo.clickOnButton(0);
      solo.assertCurrentActivity("LoadFacilityTree
            Activity erwartet", LoadFacilityTree.class);
   }
```

Listing 5.5: Auszug aus einen Robotiumtest für *MainMenu*

Die Simulatorfunktion *assertCurrentActivity()* überprüft, ob die derzeit aktive Activity dieselbe ist, wie ihr über den zweiten Parameter angegeben wird. So kann sichergestellt werden, dass nach gewissen Vorgängen die richtigen Activities gestartet werden.

5.2.3.4 Datenbank Tests

Als Datenbank kommt Db4o zum Einsatz, was einen sehr einfachen Umgang mit dem Thema Persistenz ermöglicht. [Db4o] Um die Datenbank zu kapseln wird die Klasse *DbAccessDb4o* verwendet. Sie besitzt einige Methoden, um einfach auf bestimmte Objekte, wie z.B. *Fault* oder *Panel*, Zugriff zu haben. Da die Klasse unabhängig von Android ist, kann sie gut getestet werden. Die Testklasse braucht lediglich von Android die Möglichkeit, auf das Dateisystem zu schreiben, um eine Testdatenbank anzulegen, weshalb die Testklasse *InstrumentationTestCase* zum Einsatz kam. Die Testdatenbank wird für jeden Testcase in *setUp()* neu erstellt, damit alle Tests unabhängig voneinander ablaufen.

```
protected void setUp() throws Exception {
      super.setUp();
      filesdir = getInstrumentation()
               .getTargetContext().getFilesDir();
      dbpath = filesdir + "/testdb.db";

      dbaccess = new DbAccessDb4o(dbpath);
   }
```

Listing 5.6: Erstellen einer Testdatenbank mithilfe der Testklasse *InstrumentationTestCase*

6 Zusammenfassung

Autor: Daniel Bobst, Samuel Hüppi

In diesem Kapitel wird auf die erreichten Ziele eingegangen sowie Ideen für zukünftige Erweiterungen vorgestellt.

6.1 Resultat

Wenn man FireTablet mit den Zielen in Abschnitt 1.4 vergleicht, können folgende Aussagen gemacht werden:

6.1.1 Sortiert nach Zielen

1. Es können zu allen relevanten Objekten der Zentralenstruktur Fehler erfasst werden. Namentlich sind dies:
   - Areal
   - Sektion
   - Zone
   - Panel

   Fehler können in Form von Text, Bild und Ton erfasst werden. Eine Zusammenfassung der erfassten Fehler kann zur Weiterverarbeitung mit einem geeigneten Programm exportiert werden. Ton- und Bildaufnahmen selbst liegen direkt zugreifbar auf dem externen Speicher des Tablets.
   Neben der Fehlerliste kann zusätzlich eine Zusammenfassung über den Zustand der oben erwähnten Objekte exportiert werden.

2. FireTablet kann Testalarme empfangen. Allerdings kommen diese nicht direkt von der Zentrale, sondern von einem Simulationsserver, der auf dem gleichen Computer läuft wie das Gateway. Geräte können in den Testmodus versetzt werden. Für Geräte im Testmodus generiert der Simulationsserver anschliessend automatisch oder auf Knopfdruck Testalarme.

3. Das Tablet ist selbstverständlich portabel und kann überall hin mitgenommen werden. Das gesamte Userinterface wurde im Querformat erstellt, so dass es an den Arm des Monteurs befestigt werden könnte. Es wurde ein Augenmerk auf grosse Buttons für gute Bedienbarkeit gelegt.

4. Die Führung des Servicetechnikers durch den gesamten Testablauf, wie von Siemens angedacht, ist nur bedingt vorhanden. Es fehlen z.B. Checklisten, die beim Testen von Zentralen helfen.

Folgende Use Cases konnten im Rahmen der Arbeit realisiert werden:

6.1.2 Sortiert nach Use Cases

Use Case	Implementations-Status
01 Mit Gateway verbinden	Erfüllt
02 Sensortest durchführen	Erfüllt
03 Anlage betrachten	Erfüllt
04 Zentrale Funktionsprüfung	Funktion wurde aus Zeitgründen weggelassen
05 Mangel erfassen	Erfüllt
06 Prüfdokumente generieren	Erfüllt
07 Bild aufnehmen	Erfüllt
08 Voicememo aufnehmen	Erfüllt
09 Licht an	Funktion nicht vorhanden. Es wurde keine Möglichkeit gefunden die LED des Samsung Tablets anzusteuern. [Flashlight Stackoverflow]

Tabelle 6.1: Implementations-Status der geplanten Use Cases

6.2 Ausblick

FireTablet muss noch an sehr vielen Stellen verbessert werden um es produktiv einsetzbar zu machen. So muss sicher an der Darstellung des Userinterfaces gearbeitet sowie dessen Performance gesteigert werden. Hierzu wären Usability-Tests mit Servicetechnikern interessant. Zur Zeit basiert die Datendarstellung auf Annahmen aufgrund von verschiedenen Dokumentationen.
Viel gewünschte Funktionalität bezüglich Verarbeitung von Ereignissen hängt von den Fähigkeiten des Gateways ab, weshalb auch dessen Entwicklung parallel zur Weiterentwicklung von FireTablet hilfreich wäre.
Im Verlauf der Arbeit tauchten folgende weiterführende Ideen auf:

- **Gebäudeplan integrieren** Auf dem Tablet könnte der gesamte Gebäudeplan gespeichert sein, mit den Standorten aller Geräte. Eventuell könnte sogar Indoornavigation miteinbezogen werden, ähnlich dem Indoor WPS Projekt. [Indoor WPS Projekt]
- **Testartefakte zentral auf einem Server speichern** Die Testresultate befinden sich nach abgeschlossenem Test auf dem Tablet. Dort nützen sie allerdings nicht viel da die Resultate nur vom Servicetechniker betrachtet werden können. Es wäre nützlich, wenn bei einem Abschluss des Tests die Resultate umgehend auf einen zentralen Server der Siemens geladen würden, inklusive eine Benachrichtigung an die nach bearbeitende Stelle.
- **Synchronisation der Daten mit Gateway.** Daten wie Gebäudepläne, Zentralen und Testlogs sollten direkt auf der Zentrale oder einem zentralen Server der Siemens abgerufen werden können. Auf dem Tablet könnten auch die Manuals zu den Anlagen von Siemens angezeigt werden.
- **Sprachsteuerung** FireTablet sollte mit Sprachbefehlen bedient werden können um möglichst hohe Bewegungsfreiheit zu garantieren und den Bedienkomfort zu erhöhen.

- **Prüfpflücker mit Bluetooth** Um jeder Zeit Informationen über erfolgreiche Tests zu erhalten ist es unumgänglich, dass das Tablet mit dem Prüfpflücker kommunizieren kann. Es ist unmöglich zu garantieren, dass das Tablet jederzeit Verbindung mit dem Gateway hat. Darum sollte direkt mit dem Prüfpflücker interagiert werden.

7 Projektmanagement

Autor: Daniel Bobst, Samuel Hüppi

Dieses Kapitel befasst sich mit der Planung des Projektes sowie der eingesetzten Software. Die Planung wird anhand des ursprünglichen und überarbeiteten Projektplans und der Übersicht zu den aufgewendeten Stunden pro Use Case dargestellt. Am Schluss befinden sich die beiden Erfahrungsberichte.

7.1 Zeitplan

Im Verlauf des Projekts wurden zwei Projektpläne erstellt. Version 1 wurde ganz am Anfang erstellt anhand von Schätzungen wie viel Zeit wohl für die einzelnen Use Cases benötigt werden würde. Im Verlauf der Zeit geriet die Entwicklung ins Hintertreffen und es musste für die letzten fünf Wochen eine Version 2 des Projektplans erstellt werden. Dabei wurde auch entschieden die Use Cases "05 Wartung abschliessen" und "07 Zentrale testen" nicht zu realisieren, damit man sich auf die grösseren Use Cases konzentrieren konnte. Anschliessend sind beide Projektpläne abgedruckt.

7.1.1 Projektplan Version 1

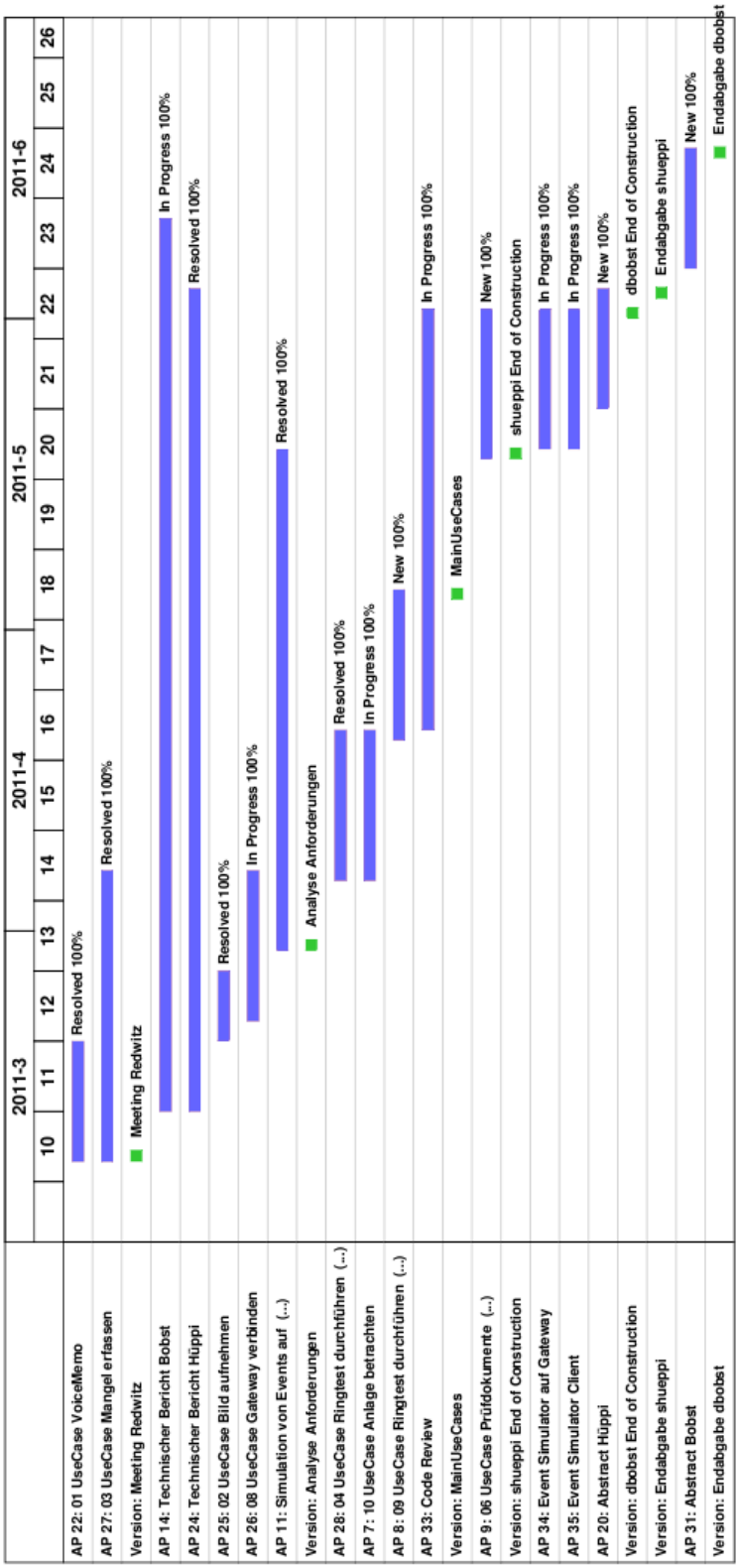


Abbildung 7.1: Gantt-Chart generiert von Redmine

7.1.2 Projektplan Version 2

Arbeitspaket	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16
09 UseCase Ringtest durchführen - Backend	Analyse GW	Backend Tab					
		Backend GW					
04 UseCase Ringtest durchführen - Frontend	GUI						
07 UseCase Zentrale testen	Nicht implementiert						
05 UseCase Wartung abschliessen	Nicht implementiert						
06 UseCase Prüfdokumente generieren							
Technischer Bericht Hüppi				Doku			
Technischer Bericht Bobst					Doku	Doku	Doku

Bobst
Hüppi

Abbildung 7.2: Überarbeiteter Projektplan für die letzten fünf Wochen

7.2 Stundenübersicht

Die Grafik zeigt die eingesetzten Stunden aller Mitarbeiter.

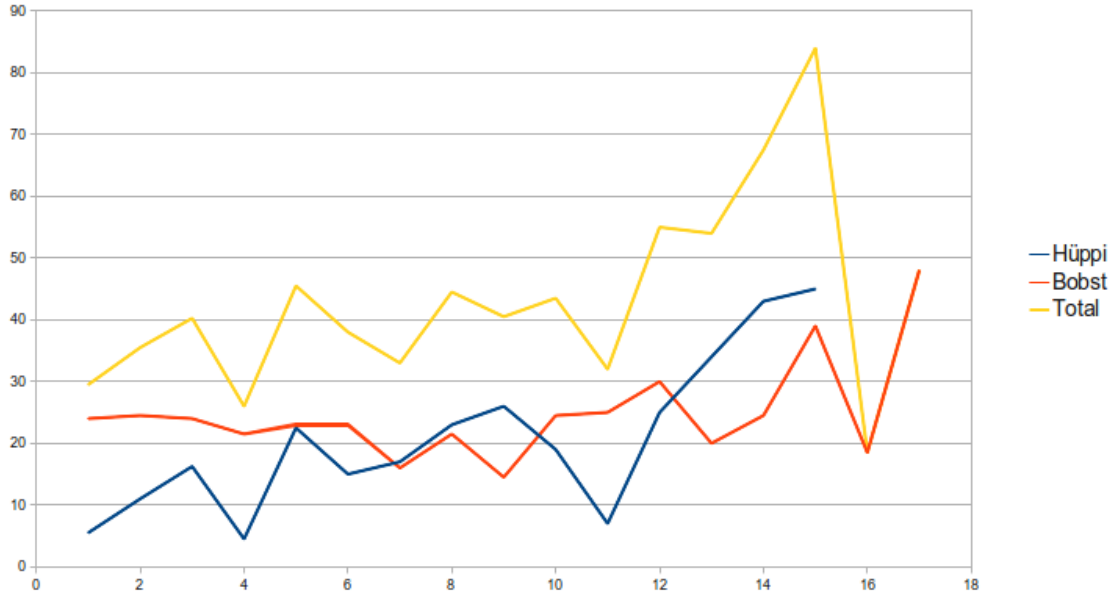


Abbildung 7.3: Aufgewendete Zeit in Stunden verteilt über 17 Wochen Projektdauer

7.3 Infrastruktur

Während der Entwicklung wurden die folgenden Programme eingesetzt:

Software	Einsatzbereich
Eclipse Helios 3.6	glside
Android Development Toolkit 10.0.1	Eclipse Plugin für Android Entwicklung
GIT	Version Control System
Hudson 1.396	Continuous Integration Server
Texmaker 2.0	LaTeX Editor für die Dokumentation
Dia 0.97.1	Grafikwerkzeug zur Erstellung von Diagrammen
Gimp 2.6.10	Bildbearbeitung
Inkscape 0.48	Vektorbasierte Bildbearbeitung
Ubuntu 10.10/11.04	Betriebssystem Arbeitsrechner
Redmine	Projektmanagement Server

Tabelle 7.1: Bei der Entwicklung von FireTablet verwendete Software

7.4 Erfahrungsbericht

Autor: Samuel Hüppi

Wie immer am Anfang eines Projekts ist der Rahmen sehr undefiniert und es scheint als könnte man sich in hundert Richtungen verrennen. Was mir in dieser Situation jedoch sehr geholfen hat, war die Anweisung des Dozenten, einfach einmal Dinge zu definieren und Annahmen zu treffen. Bei diesem Prozess setzt man sich viel intensiver mit der Materie auseinander und stösst schnell auf Punkte, die man nochmals überdenken muss. Nach und nach kommen weitere Inputs zum Gesamtbild dazu und es zeichnen sich langsam konkrete Umrisse ab, die man wiederum festhält.
Beim praktischen Teil der Arbeit hatten wir anfangs vor allem mit dem Android-Framework zu kämpfen. Viele der Konzepte waren neu und es ging nur langsam aber stetig vorwärts. Mit der Zeit war dann vieles bekannter und die Entwicklungsgeschwindigkeit nahm etwas an Fahrt auf.
Die Zusammenarbeit im Team empfand ich als sehr angenehm. Wir konnten uns gut ergänzen, da Daniel Bobst sehr genau arbeitet und ich eher auf ein Resultat zu renne. So konnte ich viel von seiner exakten Arbeitsweise profitieren und er vielleicht auch etwas von meinen raschen Lösungen. Die Situation mit den verschiedenen Arbeiten (SA/BA) empfand ich in keiner Hinsicht negativ. Natürlich entsteht minimal mehr Arbeit, dies ist aber eigentlich vernachlässigbar.

Während der Arbeit hatte ich die Möglichkeit, mein Informatikwissen in sehr viele Bereiche auszudehnen. Sowohl was das Android-Framework betrifft als auch betreffend Dokumentation und Projektarbeit. Abschliessend kann man sagen, dass die Arbeit mit den vielen Open Source Tools absolut Freude macht und sich meine Einstellung gegenüber Open Source auch durch viel Praxiserfahrung nicht ins Negative gewandelt hat.

7.5 Erfahrungsbericht

Autor: Daniel Bobst

Für mich war diese Arbeit sowohl eine willkommene Abwechslung als auch eine Herausforderung. Als Abwechslung betrachtete ich sie, weil es ein völlig anderes Thema war als jenes meiner Studienarbeit. Schon damals hatte ich ein Android-Projekt ins Auge gefasst, aber schlussendlich wurde uns eine Arbeit im Bereich Webapplication Security vergeben. Zwar war jene Arbeit ebenfalls spannend, aber mein Interesse an Android war geweckt. Eine Herausforderung war die Arbeit, weil wirklich jeder Aspekt neuartig für mich war. Ausser einem kurzen Einstieg in die Welt von Android im Modul Betriebssystemkonzepte war mir das Framework unbekannt. Abgesehen von Eclipse als Entwicklungsumgebung waren für mich durchgehend alle eingesetzten Programme neu; sei dies *Git* statt *SVN* zur Versionsverwaltung, *Hudson* statt *Bamboo* als Build-Server, *Db4o* statt *MySQL* als Datenbank, *Dia* statt *Enterprise Architect* zum Zeichnen von Diagrammen oder LaTeX statt *OpenOffice* zur Dokumentation. Teilweise machte dies die Einarbeitungszeit sehr langwierig, aber es bedeutete auch die einmalige Gelegenheit, meinen Wissensstand umfangreich zu erweitern.

Die Zusammenarbeit mit Samuel war ein wahrer Glückstreffer. Tendenziell analysiere ich ein Problem viel zu lange, bevor ich mit der Umsetzung einer Lösung beginne. Samuel hingegen beginnt sogleich mit der Umsetzung, das Ziel kristallisiert sich durch den Prozess. Ich glaube, wir haben durch unsere völlig unterschiedlichen Ansätze beide gelernt, wie wir unser individuelles Arbeitsverhalten verbessern können.

Die Betreuung durch Siemens war ebenfalls sehr angenehm. Sowohl Dokumentation, Hardware als auch Ratschläge wurden uns nach Bedarf schnell und zuverlässig mitgegeben. Man könnte eher feststellen, dass von unserer Seite her zu wenig Feedback an Siemens floss. Besonders Mitte Semester, als einige hartnäckige Probleme viel Zeit in Anspruch nahmen, konzentrierten wir uns sehr auf die Umsetzung und weniger auf die Kommunikation mit unserem Industriepartner. Auch das ist eine wertvolle Lektion gewesen, die wir in unser Arbeitsleben mitnehmen können.

Literaturverzeichnis

[Hello, Android] Introducing Google's Mobile Development Platform
Third Edition
Ed Burnette
The Pragmatic Bookshelf
http://www.pragprog.com/titles/eband/hello-android
2010

[Android 2] Grundlagen und Programmierung
Arno Becker, Marcus Pant
dpunkt.verlag
http://www.dpunkt.de/buecher/3319.html
2010

[FS20 BACnet Spec] FS20 Sinteso Fire Detection System MP3.0
BACnet Interface Description Specification
Siemens Building Technologies / Fire Safety & Security Products
2010

[FS20 Maintenance] FS20 Fire Detection System
Commissioning, Maintenance, Troubleshooting
Siemens Building Technologies / Fire Safety & Security Products
MP3.0
2010

[Prüfprotokoll] PruefprotokollFuerBMA1.xls
Prüfprotokoll
Siemens Building Technologies
25.05.2010

[Checkliste] Checkliste "Revision FS20"
Siemens Building Technologies / Fire Safety
2010

[APF Foliensatz 14] Foliensatz Modul Advanced Patterns and Frameworks 2010
Prof. Peter Sommerlad
referenzdokumente/14_boxing_killing.pdf

[Android Reference] Referenz für das Androidframework
Google
http://developer.android.com/reference/packages.html
Abgerufen am: 23.5.2011

[Android Manifest Reference] Android Manifest in der Reference.
Google

http://developer.android.com/guide/topics/manifest/manifest-intro.html
Abgerufen am: 23.5.2011

[Parcelable Reference] Android Manifest in der Reference.
Google
http://developer.android.com/reference/android/os/Parcelable.html
Abgerufen am: 23.5.2011

[Activity Testing] Activities testen mit den verschiedenen Testklassen
Google
http://developer.android.com/guide/topics/testing/activity_testing.html
Abgerufen am: 25.5.2011

[Testing Tutorial] Ein Testprojekt um das Testing von Androidklassen zu lernen
Google
http://developer.android.com/resources/tutorials/testing/helloandroid_test.html
Abgerufen am: 30.5.2011

[Painless Threading] Verwendung von Threads zusammen mit Activities
Google
http://developer.android.com/resources/articles/painless-threading.html
Abgerufen am: 30.5.2011

[Turbo-charge your UI] Wie man Listadapter effizient programmiert
Google
http://www.google.com/events/io/2009/sessions/TurboChargeUiAndroidFast.html
Abgerufen am: 30.5.2011

[Activity Reference] Activity auf der Android Reference
Google
http://developer.android.com/reference/android/app/Activity.html
Abgerufen am: 30.5.2011

[Robotium] Testframework für Android
Jayway
http://code.google.com/p/robotium/
Abgerufen am: 23.5.2011

[roboguice] Framework zur Vereinfachung des Android Framework
http://code.google.com/p/roboguice/
Abgerufen am: 24.5.2011

[Db4o] Datenbank für Objekte
Versant

http://www.db4o.com/
Abgerufen am: 25.5.2011

[JUnit] Testing Framework für Java
Open Source Project
http://www.junit.org/home
Abgerufen am: 30.5.2011

[Dependency Injection] Wikieintrag zu Dependency Injection
Wikipedia
http://de.wikipedia.org/wiki/Dependency_Injection
Abgerufen am: 30.5.2011

[Microtesting] Microtesting ist ein Begriff von Joshua Kerievsky, Gründer von Industriallogic.
IndustrialLogic
https://elearning.industriallogic.com/gh/submit?Action=AlbumContentsAction&album=theBasics&devLanguage=Java
Abgerufen am: 30.5.2011

[Structure101] Codeanalyse Software
headwaysoftware
http://www.headwaysoftware.com/products/structure101/index.php
Abgerufen am: 30.5.2011

[BACnet Stack] Homepage des BACnet Stack.
http://bacnet.sourceforge.net/
Abgerufen am: 1.6.2011

[VdS Richtlinien] VdS-Richtlinien für automatische Brandmeldeanlagen - Planung und Einbau
VdS Schadenverhütung GmbH
2010

[Quelle] Beispiel eines Spider-Charts.
http://en.wikipedia.org/wiki/File:Spider_Chart.jpg
Abgerufen am: 2.6.2011

[Wikipedia Android] Wikipediaeintrag zum Thema Android.
http://de.wikipedia.org/wiki/Android_(Betriebssystem)
Abgerufen am: 2.6.2011

[Flashlight Stackoverflow] Hilfe Thread auf Stackoverflow betreffend dem Samsung Galaxy Tab und Flashlight
http://stackoverflow.com/questions/5017455/how-to-use-camera-flash-led-as-torch-on-a-samsung-galaxy-tab
Abgerufen am: 2.6.2011

[Indoor WPS Projekt] Projekt um Positionsbestimmung in einem Gebäude zu haben.
http://gis.hsr.ch/wiki/IndoorWPS
Abgerufen am: 2.6.2011

Glossar

Activity Activities sind die Grundbausteine, aus denen ein Android Userinterface besteht. Sie stellen eigentlich eine Bildschirmseite dar welche auf Aktionen des Benutzers oder des Systems reagieren kann. Zudem ist sie verantwortlich für die Darstellung des Inhalts. 83, 84

BMZ Brandmeldezentrale. 29, 31, 35

Context Der Context bietet Zugang zu globalen Informationen über eine Applikation. Es gibt verschiedene Klassen die von Context direkt oder indirekt ableiten zB *Activity* oder *Application*. Contexte werden an vielen Stellen verwendet um Activities zu starten oder Intents abzusetzen. 77, 83

csv Comma Separated Value. 46, 79

Dependency Injection "Dependency Injection ist ein Entwurfsmuster und dient in einem objektorientierten System dazu, die Abhängigkeiten zwischen Komponenten oder Objekten zu minimieren."
Zitat: `http://de.wikipedia.org/wiki/Dependency_Injection` (30.5.2011). 84

HashMap HashMaps sind abstrakte Datentypen. Sie funktionieren ähnlich wie Listen sind aber viel schneller bei vielen Einträgen da zu jedem Eintrag eine Hashfunktion ausgeführt wird und der Eintrag anschliessend an dieser Stelle gespeichert wird. 78

HSR Hochschule für Technik Rapperswil. 27

Intent Ein Intent ist eine Art Beschreibung einer Operation, welche man ausführen will. An einen Intent können zudem Daten angehängt werden, die von der aufzurufenden Anwendung verwendet werden können. In einem Intent kann zum Beispiel eine Activity der eigenen Applikation beschrieben sein oder auch eine fremde Applikation. Das Androidsystem sammelt die Intents und ruft gemäss diesen die neuen Activities auf. 78

Mock Mocks sind abgeleitete Klassen die so tun als wären sie die abgeleitete Klasse aber viele Funktionen vereinfachen. Ein Mock wird vor allem bei UnitTests eingesetzt um zeitaufwändige Klassen zu entschärfen. 84

Prüfpflücker Der Prüfpflücker ist ein Gerät welches an Brandmelder angedockt werden kann um die Brandmelder auf korrekte Funktionalität zu prüfen. Sobald der Prüfpflücker angedockt ist kann ein Testalarm ausgelöst werden. Er erinnert von seiner Bauweise und Handhabung her etwas an ein Gerät mit welchem man z.B. Birnen pflücken geht. 7, 25

VM Virtuelle Maschine. 21