Mobile Servicetechniker App

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2011

Autor(en):	Sascha Bauer, Alexander Klee
Betreuer:	Patrick Dietschweiler
Projektpartner:	Büchi Labortechnik AG, Flawil SG
	Noser Engineering AG, Winterthur ZH
Experte:	Prof. Hansjörg Huser
Gegenleser:	Ivan Bütler

TABLE OF CONTENTS

PROJEKTPLAN

ANFORDERUNGSSPEZIFIKATION

SOFTWARE ARCHITECTURAL DOCUMENT

1 ABSTRACT

AUSGANGSLAGE

Abbildung 1 Kjeldahl Messgerät (K370)

Die Firma Büchi Labortechnik AG entwickelt und vertreibt Laborgeräte, die für Forschung und Entwicklung in verschiedenen Branchen eingesetzt werden. Die sogenannten Kjeldahl- Geräte werden in der Lebensmittel oder Pharmaindustrie angewandt um den Stickstoffgehalt einer Probe zu ermitteln. Daraus kann der Proteingehalt bestimmt werden, der z.B. auf einer Lebensmittelverpackung zu sehen ist. Nebst der Hardware und der Gerätesoftware entwickelt Büchi ein eigenes Qualitätsmanagement- Tool. Damit können Servicemitarbeiter per Ethernet eine Verbindung zu einem Gerät herstellen und Qualitätskontrollen durchführen. Die Software setzt auf einem firmeninternen Framework auf, welches die Ausführung der einzelnen Funktionstests steuert.

VORGEHEN/ TECHNOLOGIEN

Abbildung 2 Büchi Mobile App während eines Tests

Unter der technischen Leitung der Firma Noser Engineering sollten wir die Machbarkeit einer mobilen Anwendung abklären, die es dem Benutzer erlaubt über WLAN sämtliche Funktionstests am Kjeldahl-Gerät vom Smartphone aus durchzuführen. Diese Machbarkeitsstudie sollte in Form einer Windows Phone 7 (WP7) Applikation durchgeführt werden. Folgende drei Schritte waren dabei zentral. Bestehenden Programmcode analysieren UI für das Smartphone konzipieren Abläufe der Testvorgänge implementieren Als erster Schritt wurde die Portabilität des Codes abgeklärt. Es wurde sichergestellt, dass die zentralen Elemente des Frameworks auf dem WP7 lauffähig sind. Weiter strebten wir eine hohe Entkopplung von UI und Framework an, um so die Programmlogik des Frameworks beizubehalten. Neben UI- und View- Model- Layer besteht unsere Architektur aus einem Wrapper-, Framework- und einem Kommunikations-Layer. Der Wrapper-Layer trennt unsere Entwicklung vollständig vom Büchi-Testframework ab. Daten werden über die Wrapper aus dem Framework in die View- Models geladen und via Binding in den Views angezeigt.

Abbildung 3 Büchi Mobile App nach einem Test

ERGEBNIS

Um die Kommunikation mit dem Endgerät zu gewährleisten mussten wir die bestehende synchrone Socket- Implementierung auf die WP7-kompatiblen asynchronen Sockets umschreiben. Unser Ziel war es, das Framework der Firma Büchi unangetastet zu lassen um die bestehende Businesslogik für unsere Applikation zu nutzen. Bis auf vereinzelte Änderungen an den Schnittstellen zum Phone konnte dies umgesetzt werden und wir können eine stabile Lösung mit einem vergleichbaren Funktionsumfang zur PC-Software präsentieren.

Kjeldahl Servicetechniker Mobile App

Machbarkeitsstudie

Management Summary

2 AUSGANGSLAGE

Folgende Arbeit wurde im Rahmen einer Bachelorarbeit in Zusammenarbeit mit der Firma Noser Engineering Winterthur und dem Institut for Networked Solutions (INS) an der Hochschule für Technik in Rapperswil (HSR) durchgeführt. Die Firma Büchi Labortechnik AG mit Sitz in Flawil arbeitet im Bereich der Software-Entwicklung eng mit der Firma Noser Engineering zusammen. Als neueste Innovation soll auch der Smartphone Sektor erschlossen werden. Dabei wurden als Initialvorgang drei Arbeiten an Hochschulen ausgeschrieben, zwei davon in Rapperswil und eine in Bangkok. Alle Arbeiten befassen sich mit einem Kjeldahl-Laborgerät, das in der Industrie für die Messung des Proteingehaltes verschiedener Materien eingesetzt wird. So entstehen beispielsweise die Proteinwerte der Nährmitteltabelle auf Nahrungsmittel.

2.1 SERVICETECHNIKER APP

Für einen Funktionstest in Form eines Services und für die Qualitätsprüfung vor Verkauf existiert eine Applikation mit der die Qualitätsprüfung automatisch durchgeführt werden kann. Es ist eine Kombination aus Steuerbefehlen und Beobachtungen der Spezialisten vor Ort, die beim Gerät Aktionen auslösen und schlussendlich einen Bericht über den Qualitätsstand der Technik ergeben. Die Applikation wird in dieser Arbeit als Machbarkeitsstudie behandelt und soll so gut wie möglich auf der Windows Phone 7 Plattform umgesetzt werden.

2.2 KJEL MASTER

Während der produktiven Arbeit durchlaufen Kjeldahl Geräte verschiedene Stadien. Zum einen muss eine Vorbereitung durchgeführt werden, z.B. Vorheizen, es gibt einen StandBy Modus, das Gerät kann beschäftigt oder bereit sein oder sich in vielen anderen Zuständen befinden. Damit man nicht immer vor Ort sein muss, um dies vom Display des Gerätes abzulesen, soll parallel zu der bereits existierenden PC App ebenfalls auf der Windows Phone 7 Plattform eine ebenbürtige Applikation entwickelt werden.

2.3 KJELDAHL CALCULATOR

Diese Arbeit wird an der Universität von Bangkok durchgeführt auf der Plattform des iPad. Jeder Vorgang einer Proteinmessung beinhaltet das Setzen von diversen Parametern. Diese hängen von der jeweiligen Materie ab, von der der Gehalt bestimmt werden soll, und sind wichtig, um zu einem korrekten Resultat zu kommen. Es sind Kenntnisse im Fachbereich der Chemie erforderlich, um damit korrekt umgehen zu können. Bis anhin existiert ein Excel Sheet mit welchem diese Berechnungen durchgeführt werden.

3 INTERESSE AM PROJEKT

Wie erwähnt ist unser Teil dieser Arbeiten die Machbarkeitsstudie der Servicetechniker App. Wir haben diese Arbeit gewählt, da unser Interesse in der IT Branche vor allem der Entwicklung im Bereich von Microsoft Technologien liegt. Ausserdem bietet die Firma Noser Engineering, Gewinner des ICT Award 2011, einen hohen Level an Know How in diesem Bereich, woraus wir uns ebenfalls einen Profit zur Steigerung unseres Fachwissens erhofften.

3.1 ZIELE

Als Grundziel wurde definiert, alle technischen Risiken der mobilen Umgebung, wie die Kommunikation mit dem Gerät über Netzwerk oder die Anzeige auf dem eher kleinen Display, zu beheben. Es soll eine geeignete Applikation zur Präsentation

für die Geschäftsleitung von Büchi AG und Noser Engineering AG erstellt werden, die einen Beweis der Funktionalität vor Ort liefern kann.

Zudem soll ein End Test oder Service Test durchgeführt werden können und ein Report File erzeugt werden. Ausserdem sollen Tests in einer Sequenz oder einzeln gestartet und durchgeführt werden können.

4 UNSERE ARBEIT

Wir haben ein User Interface konzipiert, das den Anforderungen der bestehenden PC-Software entspricht und die nötige Funktionalität mittels Programmcode entwickelt.

PC-Software	Windows Phone 7 App

Abbildung 4 Portierung der PC-Test-Software

4.1 VORGEHEN

Zuerst galt es herauszufinden, wie die PC-Software angewendet wird, welche Szenarien damit bearbeitet werden können und was für Spezialfälle auftreten können. Als Vorlage bekamen wir einen Auszug sämtlicher Projektdateien.

Mit den gesammelten Informationen unter Berücksichtigung der Zeit und der Form der Machbarkeitsstudie wurde ein Konzept für die Umsetzung aufgezogen.

Technologische Aspekte, die durch die Umgebung von Windows Phone 7 gegeben waren, haben wir kennengelernt und gezielt eingesetzt.

Schnell galt es einen Prototyp zu entwickeln, der ein Grundgerüst bot, um den Beweis der Machbarkeit in primitiver Form erbringen zu können.

4.2 PROJEKTTEAM

HSR	Patrick Dietschweiler, Projektbetreuer	Alexander Klee, Diplomand
	Jürg Jucker Vize-Projektbetreuer	Sascha Bauer, Diplomand
Noser	Martin Straumann, Stv Head Business Unit	Hans Peter Bornhauser, Dipl. Ing ETH / STV
		Oliver Voll, Software Developer
Büchi AG		Claus Elsner, Head Software & NIR Developer
		Ruedi Hartmann, Product Manager
		Jürgen Traunig, Development Chemist
		Martin Uhland, Software Developer

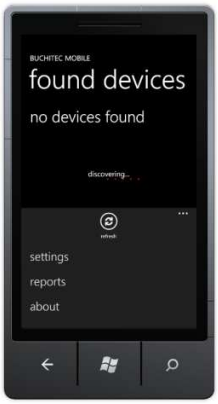
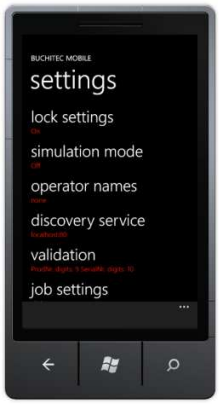
Technische Unterstützung wurde uns von Seiten der HSR und Noser Engineering AG geboten. Die Spezifizierung der Anforderungen an das Resultat wurde mit der Büchi Labortechnik AG ausgehandelt.

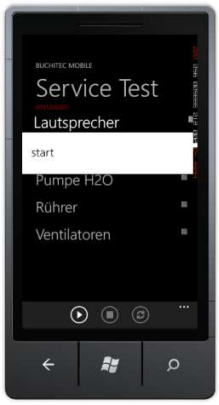
Meist fand wöchentlich ein Meeting an der HSR mit den Projektbetreuern statt. Ausserdem gab besuchten wir einige Mals die Industriepartner zum Testen der Software oder für den Wissensaustausch.

5 ERGEBNISSE

Eine anspruchsvolle Applikation ist entstanden, die mehr Funktionalität bietet als erwartet. So wurde nicht nur die Durchführung der verschiedenen Testmodus realisiert sondern auch die Konfiguration für geeignete Qualitätswerte einzelner Test. Verpackt in ein modernes User Interface mit einfacher Handhabung zeigte die Geschäftsleitung von beiden Industriepartnern Begeisterung für das Projekt. Nachfolgend eine Illustrierung der wichtigsten Funktionalitäten.

Bei unserer Entwicklung wurde stark auf eine saubere Architektur geachtet, die möglichst Abhängigkeiten der einzelnen Softwarekomponenten vermeidet.

Suchen von Geräten im Netzwerk, Navigationsmöglichkeit zu den Einstellungen	Generelle Einstellungen
	

Spezifische Einstellungen für Test - Qualitätswerte	Ausführen eines einzelnen Tests
	

Während der Ausführung	Stand nach Ausführung einer Testsequenz mit Abbruch bei Ventile-Test
	

5.1 WIEDERVERWENDETE ELEMENTE

Diverse Programmteile der PC-Software konnten wiederverwendet werden. Durch minimale Eingriffe konnte die Infrastruktur erfolgreich adaptiert werden. Dunkelblau symbolisiert unsere Eigenentwicklungen und hellblau die Bestehende. Für Büchi AG zeichnet sich dies als Erfolg, da so kein grosser Mehraufwand für die Wartung des Programmiercodes entsteht.

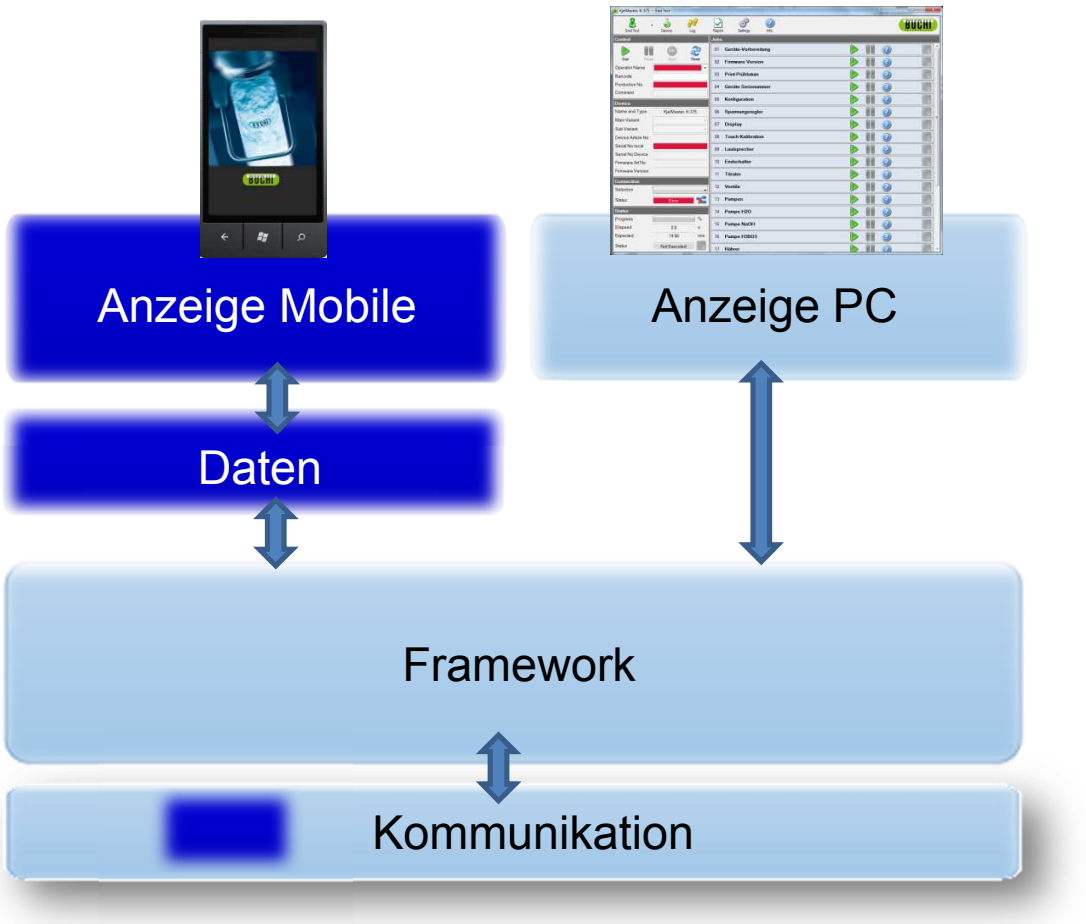


Abbildung 5 Kompatibilität der Windows Phone 7 App mit bestehendem Framework

5.2 VERBLEIBENDE PROBLEME

Da es sich um eine Machbarkeitsstudie handelt wurden gewisse Aspekte nicht berücksichtigt. Es können selten Stabilitätsprobleme bei der Bedienung auftreten. Auch funktioniert die Datenvalidierung nicht immer korrekt, da nicht alle Datenquellen (Resources) in unser Projekt eingefügt wurden.

5.3 MÖGLICHE WEITERENTWICKLUNGEN

Es wäre interessant diese Innovation auch auf ähnlichen Plattformen wie IPhone oder Android benutzen zu können. Ein einfacher Schritt um dies zu erreichen wäre die Portierung auf das .NET basierte MonoTouch Framework.
Reports sollten nach Abschluss der Arbeiten direkt auf einen Server geladen werden können. Weiter könnten daraus in der Masse Rückschlüsse auf einzelne Verbrauchsteile, Herkunftsländer oder die Verarbeitungsqualität geführt werden.

Kjeldahl Servicetechniker Mobile App

Machbarkeitsstudie

Technischer Bericht

6 EINLEITUNG UND ÜBERSICHT

6.1 KENNENLERNEN DER PC-TEST SOFTWARE

Die von Büchi eingesetzte PC-Test Software befindet sich seit über drei Jahren in Entwicklung. In dieser Zeit ist die Applikation stetig gewachsen und immer komplexer geworden. Darum galt es in einem ersten Schritt die Funktionalität der Software kennenzulernen.

6.1.1 TESTMODI

Ein Servicetechniker führt unterschiedliche Wartungs- und Servicearbeiten an einem Gerät durch. Für jede Art von Arbeit gibt es einen spezifischen Testmodus. Je nach Testmodus werden andere Testschritte, die sogenannten Jobs, geladen.

- Bevor ein Gerät in den Verkauf geht, wird der End-Test durchgeführt. Dieser Test überprüft sämtliche Funktionen des Geräts. Vom Display bis zur internen Gerätespannung wird alles überprüft.
- Eine andere Art von Test ist der Service Test. Dieser überprüft lediglich die Verschleissteile eines Geräts, wie die Ventile oder die Flüssigkeitspumpen.

Nachdem der Techniker einen Testmodus ausgewählt hat, beginnt der Ablauf des Tests.

6.1.2 TESTABLÄUFE

Ein Testablauf verfolgt folgendes Schema:

1. Der Servicetechniker wählt das Gerät aus, mit dem er den ausgewählten Test durchführen möchte.
2. Als nächstes hat er die Möglichkeit einen einzelnen oder gleich sämtliche Jobs des Tests zu starten.
3. Nun führt der Servicetechniker die Arbeiten am Gerät durch, die ihm von den Jobs aufgetragen werden.
    a. Die meisten Jobs benötigen irgendeine Form von Benutzereingabe – einen Klick, oder die Eingabe eines Messwertes – um erfolgreich abzuschliessen.
    b. Es gibt auch Jobs die ohne Interaktion des Technikers abgeschlossen werden. Diese automatischen Jobs überprüfen beispielsweise die Seriennummer des Geräts oder die Version der installierten Firmware.
    c. Ein Job kann auch pausiert oder abgebrochen werden. Letzteres führt dazu dass der Job als „nicht erfolgreich“ im User Interface markiert wird (mit einem roten X).
4. Schliesst ein Job erfolgreich ab, wird dies mit einem grünen Häckchen signalisiert.

6.1.3 EINSTELLUNGEN AM FRAMEWORK

Die Einstellungen die am Framework gemacht werden können sind sehr umfangreich und reichen vom Simulationsmodus ein-, ausschalten bis hin zur Konfiguration einzelner Jobs. Es können zwei Arten von Einstellungen unterschieden werden.

1. Temporäre Einstellungen. Bei einem Neustart gehen die Änderungen verloren.
2. Persistente Einstellungen. Bei einem Neustart bleiben die Änderungen bestehen.

Alle Einstellungen können auch wieder auf ihre Standard-Einstellungen zurückgesetzt werden.

6.2 ABGRENZEN DER MACHBARKEITSSTUDIE

6.2.1 FUNKTIONSUMFANG

Die vorliegende Arbeit ist eine Machbarkeitsstudie die untersucht, ob die Laborgeräte der Firma Büchi mit einem Windows Smartphone kommunizieren können. Das oberste Ziel einer Machbarkeitsstudie muss es sein, die Machbarkeit der in der Anforderungsspezifikation definierten Features beweisen oder widerlegen zu können. Optionale Features werden im Kontext einer Machbarkeitsstudie erst dann interessant, nachdem die benötigten Features implementiert sind und noch genügend Zeit für eine Umsetzung bleibt.

7 ANALYSE DES BESTEHENDEN UI

Siehe SoftwareArchitectureDocument.docx

8 DEFINITIVES UI

Das UI-Design war ein wichtiger Aspekt der Arbeit. Obwohl es sich lediglich um eine Machbarkeitsstudie handelte, sollte ein vergleichbarer Funktionsumfang wie bei der PC-Applikation erreicht werden.

Folgende Abbildung zeigt die finale GUI-Map wie sie auch im der Applikation umgesetzt wurde.

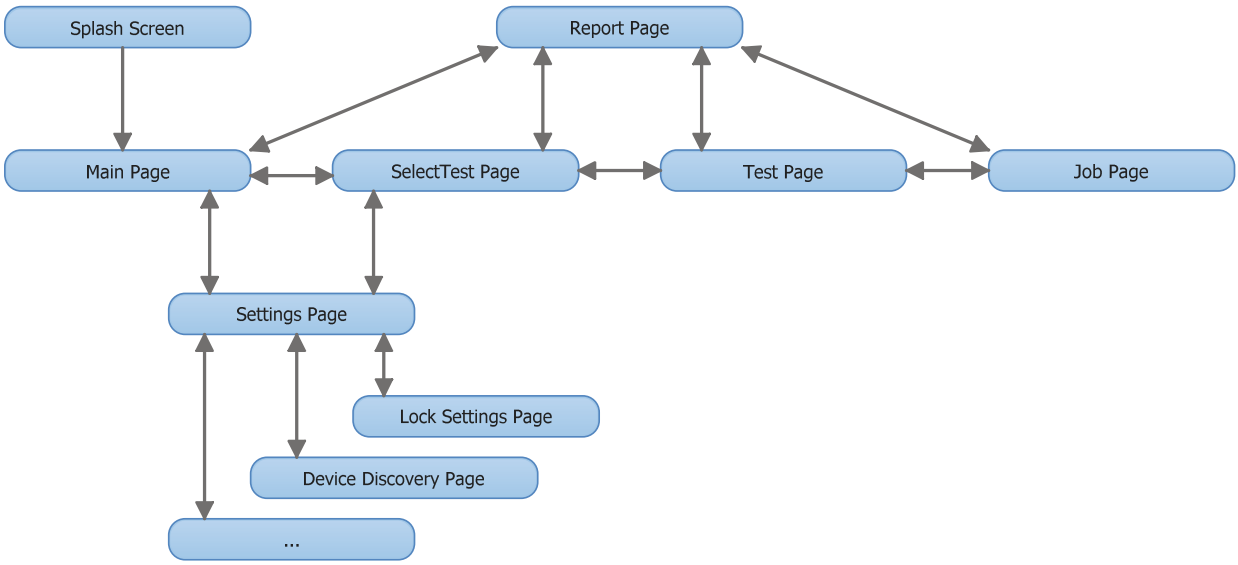


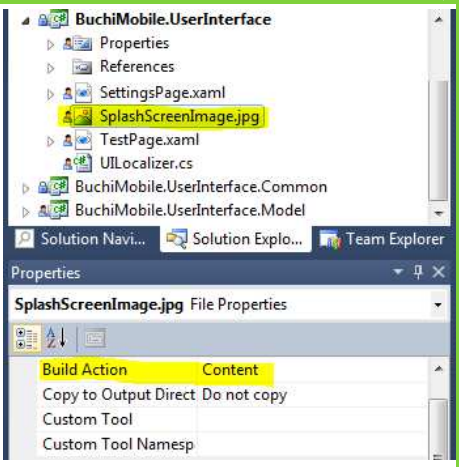
Abbildung 6 Definitive UI-Map

Jede Seite hat ihre Besonderheiten und individuellen Aufgaben zu erfüllen. Es werden zahlreiche Technologien eingesetzt, um das hoch komplexe Test-Framework so benutzerfreundlich wie möglich anzubieten.

Auf den folgenden Seiten wird auf die einzelnen Ansichten der Applikation eingegangen. Die folgende Tabelle dient als Einstiegspunkt in die technischen Bereiche der Dokumentation und als Illustrierung unserer gewonnenen Kenntnisse im Kontext der WP7 Entwicklung

8.1 SPLASH SCREEN

Screen	Bemerkungen

 Abbildung 7 Splash Screen	Der Splash-Screen ist das Bild das beim Start der Applikation angezeigt wird. Er gehört bereits zum Life-Cycle einer Windows Phone App und ist dementsprechend einfach einzubauen.
	
	Im WindowsPhoneApplication-Projekt erstellt VisualStudio ein SplashScreenImage.jpg. Dieses kann auf dem Filesystem ausgetauscht und wird beim nächsten Build in die Applikation integriert werden.
	
	 
	
	Die Build Action des Bildes muss im Properties-Fenster auf „Content“ gesetzt sein.
	
	Die Page, die nach dem SplashScreen angezeigt wird, kann im WMAppManifest.xml (im „Properties“ Ordner des UI-Projekts) definiert werden.
	
	`<Tasks>`
	`  <DefaultTask Name="_default" NavigationPage="MainPage.xaml" />`
	`</Tasks>`

8.2 MAINPAGE – DISCOVERING

Screen	Bemerkungen


Abbildung 8 Main Page – No Devices
	Als erste Seite wird die MainPage angezeigt. Unter anderem stehen der Page folgende Ressourcen zur Verfügung
	
	`<phone:PhoneApplicationPage.Resources>`
	`  <viewModels:MainViewModel x:Key="viewModel" />`
	`  <converters:NotConverter x:Key="not" />`
	`</phone:PhoneApplicationPage.Resources>`
	
	Durch das deklarieren des MainViewModel in den Ressourcen, wird beim Laden der Seite eine Instanz des ViewModels erzeugt. Diese sorgt dafür, dass die Applikation nach Geräten zu suchen beginnt. Die Suche der Geräte wird im Kapitel erklärt. Solange keine Geräte gefunden wurden, wird eine TextBlock-Control angezeigt, welcher den Benutzer über diesen Umstand informiert. Mittels VisibiltyConverter kann diese Control ein- und ausgeblendet werden. Ein Beispiel für einen solchen Visibilty Converter kann auf *Jeff Wilcox' Blog*[1] gefunden werden.
	
	`<TextBlock Text="{Binding Strings.NoDevices}"`
	`  Visibility="{Binding Devices.Count, Converter={StaticResource not}}" />`
	
	Während der Gerätesuche wird ein ProgressBar angezeigt. Wir verwenden den PerformanceProgressBar aus dem OpenSource-Framework „MyToolkit“ von Rico Suter.
	
	`<toolkit:PerformanceProgressBar IsIndeterminate="{Binding IsDiscoveringDevices}" />`

[1] http://www.jeff.wilcox.name/2008/07/visibility-type-converter/

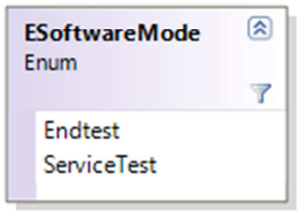
Screen	Bemerkungen


Abbildung 9 MainPage - Found Devices
	Alle gefundenen Geräte werden in einer NavigationList angezeigt. Die NavigationList-Komponente stammt, wie auch der PerformanceProgressBar, aus der Library „MyToolkit“.
	
	`<UI:NavigationList ItemsSource="{Binding Devices}" Margin="0" Navigation="SelectDevice">`
	`  <UI:NavigationList.ItemTemplate>`
	`  <DataTemplate>`
	`  ...`
	`  </DataTemplate>`
	`  </UI:NavigationList.ItemTemplate>`
	`</UI:NavigationList>`
	
	Obiges Listing zeigt den NavigationList-Container. Besonders zu beachten ist das DependencyProperty „Navigation“. Dieses erlaubt es, im Code Behind einen Event Handler zu definieren, welcher das geklickte Element als Parameter erhält. Anschliessend wird gezeigt, wie man das geklickte Element aus dem Event-Argument erhält. Dieses Vorgehen entspricht nicht dem MVVM-Standard und dient nur zur Verdeutlichung der Möglichkeiten der NavigationList.
	
	`private void SelectDevice(object sender, NavigationListEventArgs e)`
	`{`
	`  BuchiDevice device = (BuchiDevice) e.Item;`
	`  device.DoSomething();`
	`}`

	Unter dem Hostnamen wird die IP-Adresse des Geräts in der aktuellen PhoneAcce
	
	`<TextBlock Text="{Binding HostIp}" TextWrapping="Wrap"`
	`  Style="{StaticResource PhoneTextSmallStyle}"`
	`  <TextBlock.Foreground>`
	`  <SolidColorBrush Color="{StaticResource PhoneAccent`
	`  </TextBlock.Foreground>`
	`</TextBlock>`
	
	Die AccentColor-Einstellungen sind Applikationsunabhängig und können in den Sy WindowsPhones vorgenommen werden.

8.3 SELECT TEST PAGE

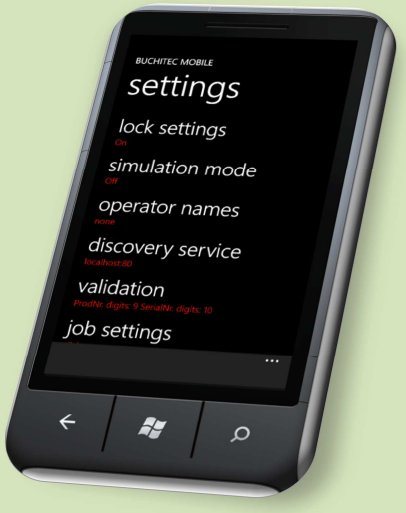
Screen	Bemerkungen

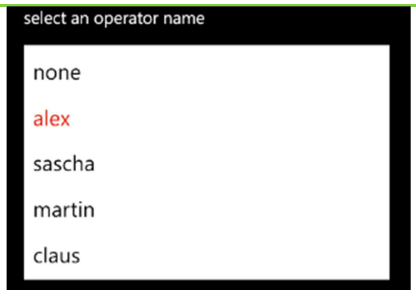

Abbildung 10 SelectTestPage
	Sobald ein Gerät ausgewählt wurde, wird eine Verbindung hergestellt und die verfügbaren Tests werden angezeigt. Dies wird durch die Instanziierung eines TestExecutionWrappers erreicht. Alle benötigten Daten werden aus dem Framework geladen und dem ViewModel bereitgestellt.
	
	In der NavigationList werden die verfügbaren Tests angezeigt. Die verfügbaren Tests entsprechen den unterschiedlichen SoftwareModes des Frameworks.
	
	 
	`//SelectTestViewModel`
	
	`ParsedTests = new List<ESoftwareMod`
	`{`
	`  ESoftwareMode.Endtest,`
	`  ESoftwareMode.ServiceTest`
	`};`
	
	Zu beachten ist, dass der Enum „ServiceTest“ in der View mit einem Leerzeichen angezeigt wird. In der NavigationList befinden sich jedoch keine Strings, sondern Referenzen auf die beiden ESoftwareModes. Hier zeigt sich die Nützlichkeit von ValueConvertern erneut. „Auf dem Weg zum UI“ fängt der ValueConverter den Enum ab, und wandelt ihn in einen Anzeige-Freundlicheren String um.

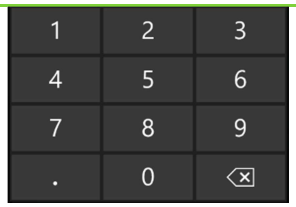
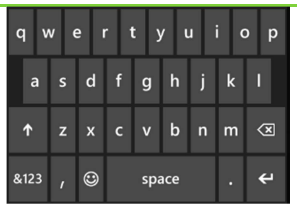
8.4 TEST PAGE & JOB PAGE

Screen	Bemerkungen
 Abbildung 11 TestPage und JobPage	Nachdem eine Testart (ESoftwareMode) ausgewählt wurde, werden die entsprechenden Jobs geladen. Das Laden der Jobs übernimmt das Framework. Ein JobWrapper sorgt dafür, dass die geladenen Jobs auf dem UI angezeigt werden können. Kapitel Job Wrapper zeigt wie dies im Job Wrapper gehandhabt wird. Die Job-Ansicht bietet dasselbe Look & Feel wie die PC-Applikation. Auch diese View benutzt den JobWrapper und den TestExecutionWrapper, um mit dem Framework zu kommunizieren.

8.5 SETTINGS

Screen	Bemerkungen
 Abbildung 12 Settings Page	Klickt man in der ApplicationBar auf „Settings“ gelangt man in die Einstellungen der BüchiMobile-App. Es gibt sowohl persistente als auch nicht persistente Einstellungen. Die persistenten Einstellungen werden auf dem IsolatedStorage des Phones gespeichert und beim Start der Applikation geladen. Informationen über die Verwendung des IsolatedStorage sind im Kapitel AppSettingsWrapper zu finden.Je nach Datentyp einer Einstellung werden andere Controls verwendet.

Type (Control)	Bild	Bemerkungen zum UserControl
Boolean	 **Abbildung 13 ToggleSwitch**	Anstelle von „On/Off“ zeigt ein ToggleSwitch „True/False“ an. Mittels ValueConverter können solche Feinheiten behoben werden.
List	 **Abbildung 14 ListPicker**	Wie alle ItemControls in .Net funktioniert der ListPicker am besten, wenn eine ICollectionView zugrunde liegt. Da die Operator-Namen vom Framework organisiert werden, wurde hier bewusst darauf verzichtet. Stattdessen wird eine einfache List<string> verwendet und die Bindings werden vom ViewModel der OperatorPage verwaltet.

Integer	 Abbildung 15 Number-Tastatur	Validierungen von Benutzereingaben können in Silverlight und WPF in einer ersten Stufe bereits **auf dem UI** vorgenommen werden. `<TextBox InputScope="Number"/>`
String	 Abbildung 16 Text-Tastatur	Jede Textbox bietet über das „InputScope“-Property die Möglichkeit den Benutzer in der Eingabe von Werten einzuschränken. `<TextBox InputScope="Text" />`

8.6 PERSISTENTE SETTINGS

Screen	Bemerkungen
 Abbildung 17 LockSettingsPage	Persistente Settings werden auf den Isolated Storage des Phones geschrieben. So können sie bei einem Neustart der Applikation wieder gelesen werden. Den Zugriff auf die Settings der Applikation regelt eine Instanz von AppSettingsWrapper. Folgende Abbildung zeigt am Beispiel der „LockSettings“-Page welche Klassen beim Speichern einer Einstellung involviert sind.

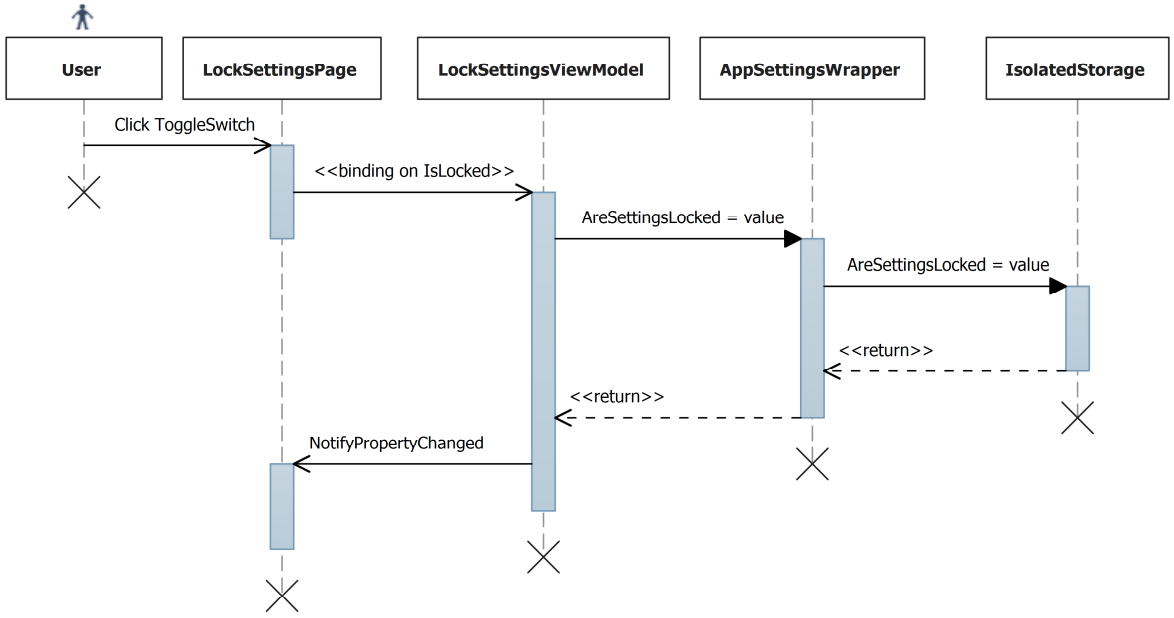


Abbildung 18 Speichern von Settings auf dem IsolatedStorage

8.7 JOB SETTINGS PAGE

Screen	Bemerkungen
 Abbildung 19 JobSettingsPage	Bei den Job Settings handelt es sich um spezifische Qualitätswerte-Einstellungen pro Funktionstest . Um die grosse Anzahl von Einstellungen zu verwalten, wird die Klasse zur Laufzeit mittels Reflection analysiert und im UI dargestellt. Folgende Abbildung zeigt welche Klassen am JobSettings-UI beteiligt sind. Eine genaue Beschreibung des JobSettingsWrapper findet sich im Kapitel Job Settings Wrapper.

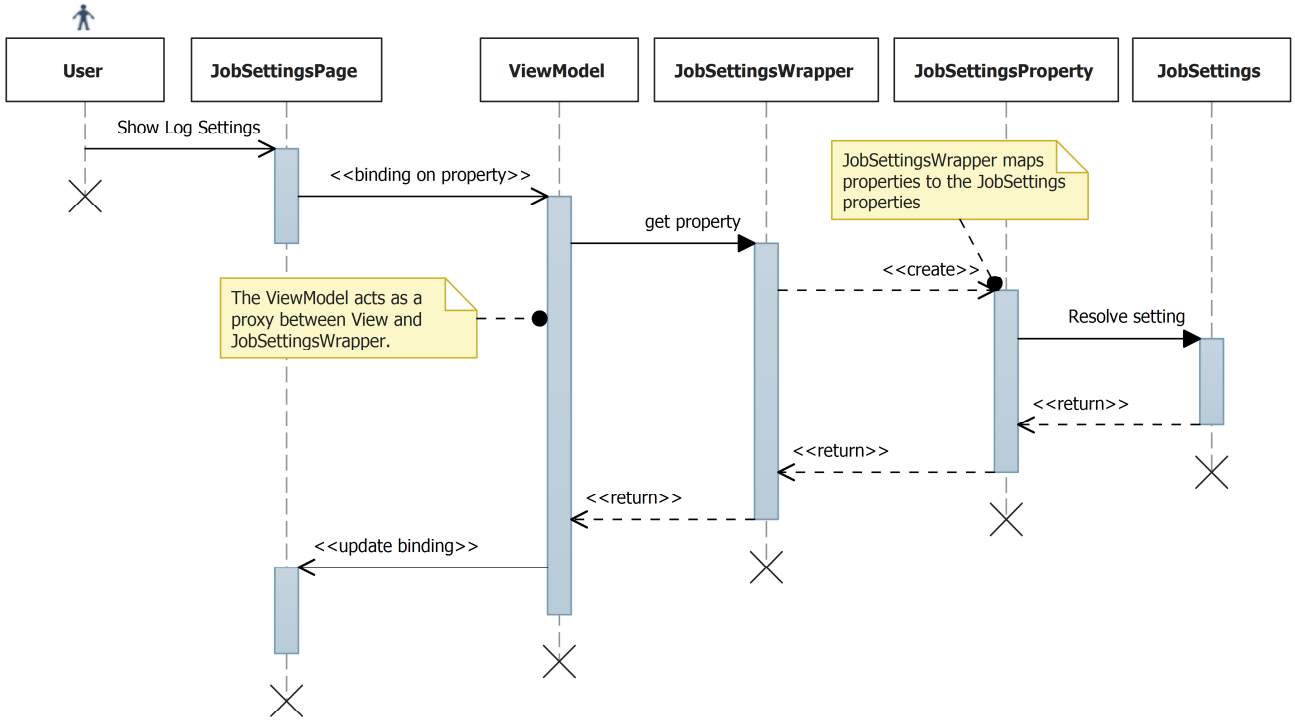
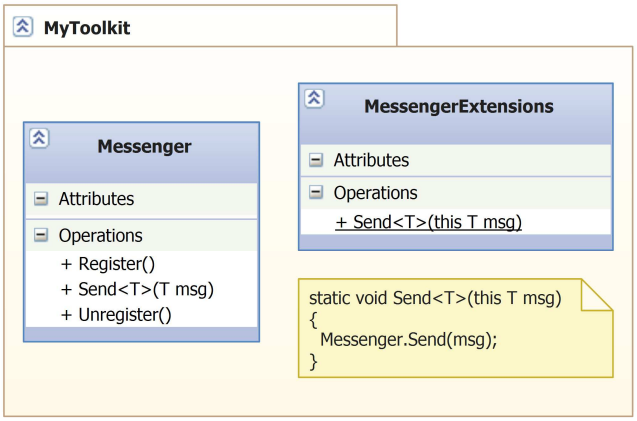


Abbildung 20 Anzeigen von JobSettings - Überblick

8.8 FEHLERMELDUNGEN UND USER-PROMPTS

Screen	Bemerkungen
 Abbildung 21 User Prompt Beispiel	Folgende Klassen trennen die Anzeige einer Fehlermeldung oder eines UserDialogs vom ViewModel.  Sendet man eine TextMessage an die Send-Methode des Messengers, wird daraus auf der aktuellen Seite ein Overlay erstellt. Die Extension-Methode „Send()“ wird dazu benutzt, ein beliebiges Objekt dem Messenger zu übergeben. Somit führt folgender Syntax zur Anzeige eines Dialogs. `new TextMessage("message", "header", TextMessage.MessageButton.OKCancel).Send();`

8.9 BINDING DER APPLICATION BAR

Um ein Binding in der AppBar zu realisieren wird im Code-Behind der View eine Instanz auf das View Model benötigt. Diese kann über einen Zugriff auf die Resources erhalten werden:

```
<phone:PhoneApplicationPage.Resources>
  <viewModels:MainViewModel x:Key="viewModel" />
  ...
</phone:PhoneApplicationPage.Resources>
```

Listing 1 Define ViewModel in XAML-Resources

```
public JobViewModel ViewModel
{
  get
  {
    return (JobViewModel)Resources["viewModel"];
  }
}
```

Listing 2 Accessing XAML-Resources from Code behind

Mit der Verwendung des ViewModels im Code Behind ist Vorsicht geboten. Dies ist sozusagen ein Verstoss gegen die Trennung zwischen UI und ViewModel. Leider gibt es aber für das folgende Szenario keine Alternativen.

8.9.1 DIE KLASSE APPBARBINDINGS

Diese Klasse wurde erzeugt, da in zwei verschiedenen Views ein beinahe identisches Binding auf die Application Bar realisiert werden soll. (Die Buttons Start und Stop werden auf TestPage.xaml und JobPage.xaml verwendet und unterliegen in beiden Fällen demselben Prädikat). Die Bedingungen für Start und Stop sind ausserdem gegensätzlich.

Abbildung 22 ApplicationBar mit Start, Pause und Stop-Button

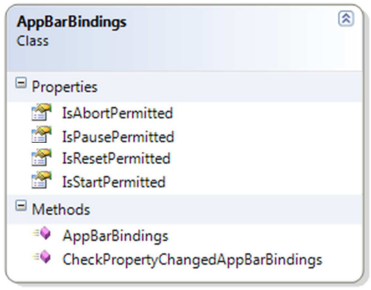


Abbildung 23 AppBarBindings-Klassendiagramm

VERWENDUNG IM CODE BEHIND

Das Ziel ist es, das IsEnabled Property bei Änderung der in AppBarBindings gespeicherten Prädikate auf den jeweiligen Buttons zu setzen. Um dies zu realisieren muss sich ein Event Handler im Code Behind auf die PropertyChanged Methode des ViewModels registrieren.

```
Private void ModelPropertyChanged(object sender, PropertyChangedEventArgs e)
{
  if (e.PropertyName == "AppBarBindings")
  {
    ApplicationBar.Buttons[0].IsEnabled = ViewModel.AppBarBindings.IsStartPermitted;
    ApplicationBar.Buttons[1].IsEnabled = ViewModel.AppBarBindings.IsPausePermitted;
    ApplicationBar.Buttons[2].IsEnabled = ViewModel.AppBarBindings.IsAbortPermitted;
  }
}
```

Listing 3 Binding der Application Bar Buttons

Leider weist der Code Unschönheiten auf. Der Zugriff auf die Application Bar Buttons kann nicht über einen Referenznamen (x:Name) gelöst werden. Die einzige Möglichkeit ist es in der Collection ApplicationBar.Buttons die Id zu kennen. Diese lässt sich an der Definitionsreihenfolge im XAML herauslesen. Hier der erste Vorteil der AppBarBindings-Klasse. Man muss nur auf einen Event reagieren und nicht auf Events jedes einzelnen Buttons. Es werden zwar immer alle Werte gesetzt, doch hat dies keinen Einfluss auf die Performance.

VERWENDUNG IM VIEWMODEL

Die gesamten AppBarBindings werden in Form eines einzigen Properties angeboten.
In den `PropertyChanged` Event Handler beider ViewModels (Job, Test), braucht man nicht mehr mit „If-Statements“ zu prüfen, bei Benachrichtigung welcher Properties der Event weitergeleitet werden soll.

Call in `ProertyChanged` EventHandler des ViewModel:

```
AppBarBindings.CheckPropertyChangedAppBarBindings(
  e.PropertyName,
  () => RaisePropertyChanged(
    myself => myself.AppBarBindings)
);
```

Listing 4 Binding der Application Bar, Verwendung im ViewModel

Hier entsteht derselbe Vorteil. Es muss sich nicht darum gekümmert werden, welches Property geändert hat und dann für dieses Explizit ein RaiseProperty

Dies übernimmt der Typ des betroffenen Properties folgendermassen selbst:

```
public void CheckPropertyChangedAppBarBindings(string propertyName, Action t)
{
  if (propertyName.Equals("CurrentExecutingJob") || propertyName.Equals("Actual    Status"))
  {
    t.Invoke();
  }
}
```

Listing 5 AppBarBinding Überprüfung beim PropertyChanged Event

8.10 WEAK REFRENCES

Event Handler die im Code Behind registriert werden, müssen beim Verlassen einer Page wieder de-registriert werden. Ansonsten wird die Instanz vom Garbage Collector nicht entfernt, weil der EventHandler noch auf sie referenziert. Navigiert man später zu dieser Ansicht zurück wird keine neue Instanz der View erzeugt, sondern die noch im Speicher vorhandene View geladen. Somit befindet man sich in einem ungewollten Zustand.

REGISTRIEREN ÜBER ONNAVIGATEDTO

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
  base.OnNavigatedTo(e);
  Messenger.Register<NavigateToTestsMessage>(OnNavigateToTestsMessageReceived);
  ViewModel.PropertyChanged += ModelPropertyChanged;
}
```

Listing 6 Registrieren von EventHandler im Code Behind

DEREGISTRIEREN ÜBER ONNAVIGATEDFROM

Das Deregistrieren der Event Handler wird ausserdem auf dem ViewModel aufgerufen. Dies wird nur zweimal im gesamten Projekt gemacht, es wurde auf ein darauf ausgelegtes Design verzichtet.

```
protected override void OnNavigatedFrom(NavigationEventArgs e)
{
  base.OnNavigatedFrom(e);
  Messenger.Unregister<NavigateToTestsMessage>();
  ViewModel.UnregisterMessages();
  ViewModel.PropertyChanged -= ModelPropertyChanged;
}
```

Listing 7 Deregistrieren von EventHandler im Code Behind

9 WCF SERVICE: FINDEN AKTIVER DEVICES

9.1 ARCHITEKTUR

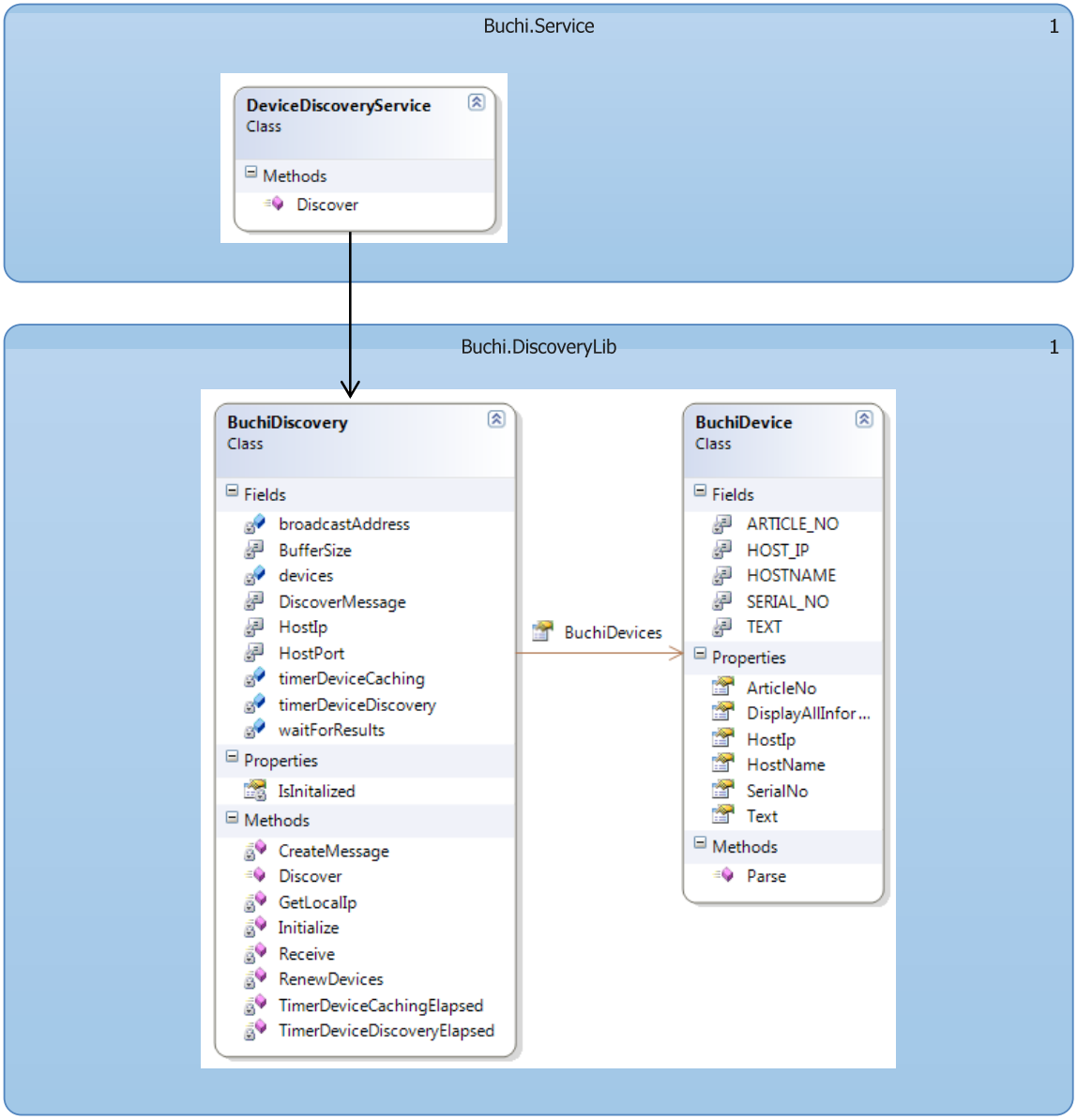


Abbildung 24 WCF Service: Architektur

9.2 SERICE CONFIGURATION

```
<bindings>
    <basicHttpBinding>
      <binding name="BasicHttpBinding_DeviceDiscoveryService" textEncoding="utf-8"
        transferMode="Buffered"/>
      </basicHttpBinding>
</bindings>

<services>
    <service name="Buchi.Service.DeviceDiscoveryService">
       <endpoint address=""
                 bindingConfiguration="BasicHttpBinding_DeviceDiscoveryService"
                 binding="basicHttpBinding"
                 contract="Buchi.Service.DeviceDiscoveryService"/>
      </service>
    </services>
```

Listing 8 Server Konfiguration des WCF Services

9.3 CACHING

Zur Aufbewahrung der Daten wurde einfach eine statische List<BuchiDevice> erstellt. Das genaue Verhalten wird im nächsten Kapitel beschrieben.

9.4 ZEITSTEUERUNG

Es ist sowieso notwendig eine bestimmt Zeitspanne auf Antworten zu warten im Netzwerk, ob Asynchron oder Synchron gearbeitet wird. Die Anzahl der antwortenden Devices ist unbekannt. Die Synchrone Variante zeigt sich darum hier als sinnvoller. Um eine Zeitsteuerung zu realisieren existieren zwei Timer in der Klasse BuchiDiscovery:

Name	Funktion	Nach Zeitdauer
timerDeviceDiscovery	Bestimmt die Zeitdauer die auf Antworten des Broadcast gewartet wird	timerDeviceCaching wird gestartet
timerDeviceCaching	Bestimmt die Aufbewahrungszeit der Resultate (BuchiDevices)	Die Clear-Methode der static List<BuchiDevice> wird aufgerufen

Die genaue Zeitdauer der Timer kann über das web.config angepasst werden.

Sinnvolle Zeiten:

- Broadcast 2-5s (je länger desto eher werden alle Geräte im Netz gefunden)
- Caching 30s – 5min

9.5 PROZESSSYNCHRONISATION

Wird der Service gleichzeitig mehrmals Aufgerufen, muss gewährleistet sein, dass nicht mehrere Clients das Versenden von Broadcasts auslösen. Die Prozesse warten darauf, bis eine Antwort vorherrscht. Darum wurde ein einfacher statischer Lock eingeführt (waitForResults vom Typ Object). Fragen zum Beispiel 100 Clients gleichzeitig nach den neuesten Geräten, so wird nur ein Broadcast ausgeführt. Ausserdem wird nur mit statischen Methoden gearbeitet, da alle Prozesse die darauf zugreifen mit denselben Daten arbeiten sollen.

```
public static List<BuchiDevice> Discover()
{
    lock (waitForResults)
    {
        if (!IsInitalized)
        {
            Initialize();
        }
        if (devices.Count == 0)
        {
            RenewDevices();
        }
    }
    return devices;
}
```

Listing 9 WCF Service: Thread Synchronisierte Discover Methode

9.6 SOCKETS

Es warden zwei Sockets verwendet (Senden, Empfangen). Dabei ist wichtig, dass der Empfangssocket einen festen freien Port zugewiesen bekommt, um Firewall-Einstellungen zu ermöglichen.

9.6.1 SEND

Der Broadcast wird mit folgenden Statements adressiert an den BuchiDiscoveryPort versandt.

```
var data = CreateMessage(answerPort);
var broadcastEndpoint = new IPEndPoint(broadcastAddress, sendPort);
bcSocket.SendTo(data, broadcastEndpoint);
Receive(answerPort);
```

Listing 10 WCF Service: Erstellen eines Broadcasts

9.6.2 RECEIVE

Während der Zeitdauer des timerDeviceDiscovery werden Antworten auf den Broadcast vom Netzwerkinterface entgegengenommen.

```
while (timerDeviceDiscovery.Enabled)
{
    try
    {
        receiveSocket.Receive(buffer);
        var answer = new ASCIIEncoding().GetString(buffer).Trim('\0');
        Console.WriteLine(answer);
        devices.Add(BuchiDevice.Parse(answer));
    }
    catch (Exception)
    {
        //time expired
    }
}
receiveSocket.Close();
```

Listing 11 WCF Service: Empfangen von Antworten

10 FRAMEWORK ADAPTION

Um den WP7-spezifischen Teil vom Framework zu trennen, wird ein FrameworkWrapper-Layer eingesetzt. So kann der Umgang mit dem Framework an einem zentralen Ort geschehen.

10.1 FRAMEWORK WRAPPER

Der Framework-Wrapper-Layer stellt die Model Schicht der Applikation dar. Um auf Daten des Frameworks zugreifen zu können wird die statische `DataExchange`-Klasse verwendet. In dieser Klasse werden die für das Programm essentiellen Objektinstanzen des Frameworks gespeichert.

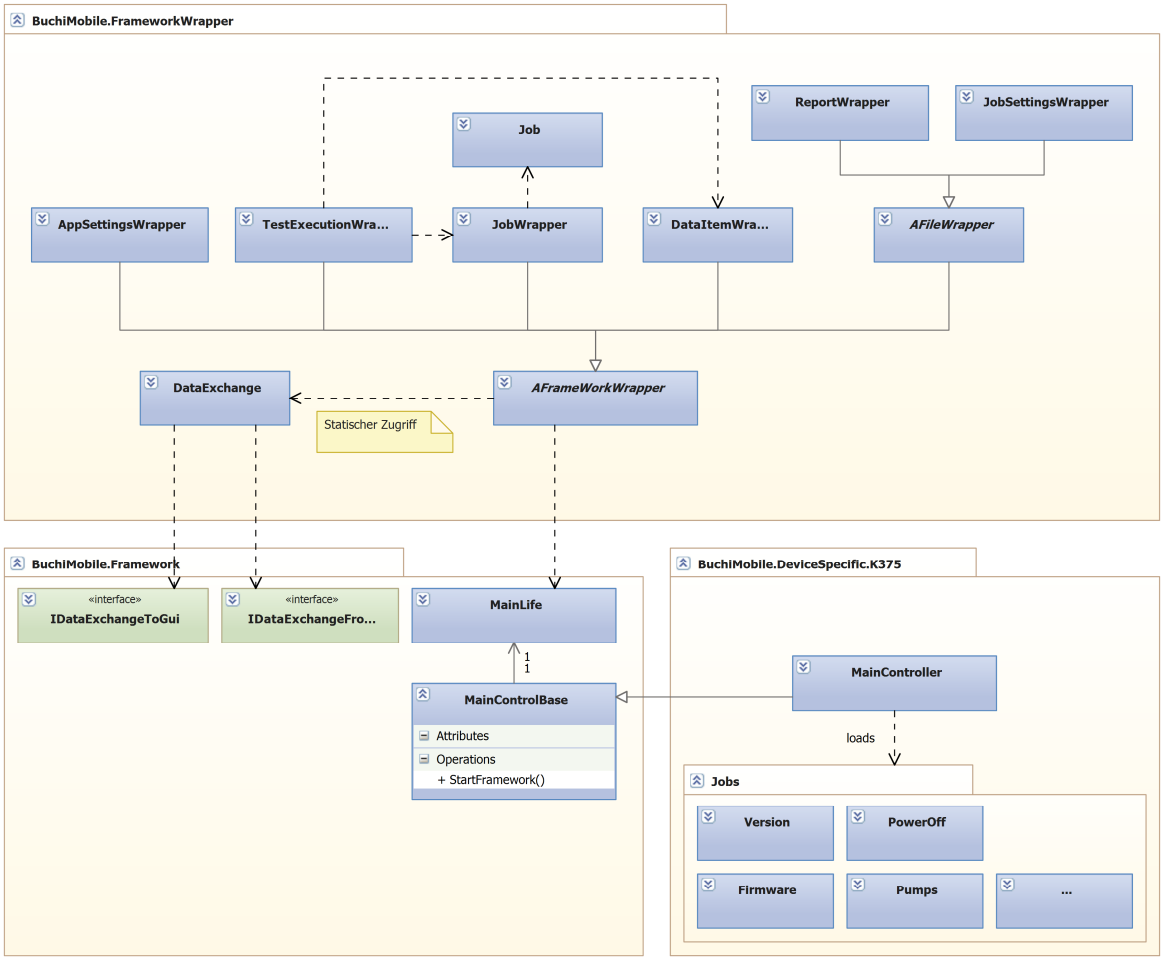


Abbildung 25 Übersicht der Framework-Wrapper

10.2 ÄNDERUNGEN AM FRAMEWORK

10.2.1 NOTWENDIGKEIT

In der PC Software von Büchi wird das UserInterface mit diversen, für die Anzeige erforderlichen, Parametern initialisiert. Stichwort: **Bottom-Up-Architektur**

In einer Silverlight Applikation geschieht dies umgekehrt. Ein UserInterface instanziiert die ViewModel-klassen, welche sich wiederum um das Management der Daten und somit auch um die Datenbeschaffung kümmern. Stichwort: **Top-Down-Architektur**

Um das bestehende Framework möglichst identisch einzusetzen, wurde ein Eingriff getätigt, der aber keinerlei Einfluss auf dessen Funktionalität hat.

10.2.2 GRUNDLAGE DER WRAPPER

Jede Wrapper Klasse leitet von `FrameworkWrapperBase<T>` ab. Wie gesagt, sind das die Klassen welche für den Datenaustausch zwischen Framework und ViewModel eingesetzt werden. Damit diese Zugriffe überhaupt möglich werden und es eine Gewährleistung gibt, dass das Framework bei Verwendung auch initialisiert wurde. Wird dies bei der Instanziierung einer Wrapperklasse im Konstruktor überprüft und gegebenenfalls gestartet.

Da teilweise in einer Wrapper-Klasse weitere Wrapper instanziert werden und die Gefahr besteht, dass der Event vom Framework erst empfangen wird, wenn die zweite Instanz erneut die Initialize() Methode aufruft, wurde ein Lock um den Vorgang gesetzt, damit das Flag-Property `IsInitalizationStarted` gesetzt werden kann und somit ein doppelter Framework-Start verhindert wird.

```
private void Initialize()
{
  lock (sync)
  {
    if (IsInitialized) return;
    if (!isInitializationStarted)
    {
      var main = new MainController();
      main.FrameworkStarted += FrameworkStarted;
      main.StartFramework();
      isInitializationStarted = true;
    }
  }
}
```

Listing 12 Initialisierung des Frameworks

Weiter in der `StartFramework()` Methode werden alle nötigen Instanzen erzeugt, die Umgebung gestartet und ein Event `FrameworkStarted` ausgelöst. Darin werden alle in den Wrapper-Klassen benötigten Instanzen per EventArgs weitergeleitet.

```
if (mainLife == null)
{
  // Start the framework
  mainLife = new MainLife();
  mainLife.InitSystem(this);
  mainLife.StartSystem(); // Starts the framework

  //this sends the communication classes to FrameworkWrapper
```

```
  if (FrameworkStarted != null)
  {
    FrameworkStarted(this,
    new FrameworkAdaptionEventArgs(DataExchangeFromGui.Instance,
                                   DataExchangeToGui.Instance,
                                   JobController.Instance.GetJobConfig(),
                                   GetJobSettingsInstance()));
  }
}
else
{
  LogInternal.Out(LogBase.LogType.Fault, "MainControlBase.StartFramework()", "Tried to start twice.");
}
```

Listing 13 Initialisierung des Framworks, Eingriff

Die Klasse `FrameworkWrapperBase<T>` hört auf den Event `FramworkStarted`. Beim Empfang eines solchen, werden die statischen Properties des `DataExchange` Objektes gesetzt.

```
private void OnFrameworkStarted(object sender, FrameworkAdaptionEventArgs args)
{
  DataExchange.ControlToGui = args.DataExchangeToGui;
  DataExchange.GuiToControl = args.DataExchangeFromGui;
  DataExchange.JobConfig = args.JobConfig;
  DataExchange.JobSettings = (JobSettings)args.JobSettings;
}
```

Listing 14 Initialisierung des Frameworks: Event Handler

Alle Zugriffe der abgeleiteten Klassen auf die statische `DataExchange`-Klasse und somit zum Framework passieren über die Properties der `FrameworkWrapperBase<T>`-Klasse. Keiner weiteren Klasse ist `DataExchange` bekannt.

```
protected IDataExchangeToGui ExchangeToGui
{
  get { return DataExchange.ControlToGui; }
}
```

Listing 15 Initialisierung des Frameworks: Beispiel der gekapselten Properties

10.3 BESCHREIBUNG DER WRAPPER-KLASSEN

10.3.1 APPSETTINGSWRAPPER

Der `AppSettingsWrapper` kapselt den Zugriff auf die Klasse `AppSettings` und injiziert wenn nötig die Einstellungen direkt ins Framework.

Benötigte Daten werden wie in allen anderen Wrapper-Klassen vom Framework geladen.

Untenstehende Grafik illustriert einen Schreibvorgang, wenn ein Property vom ViewModel her im AppSettingsWrapper gesetzt wird.

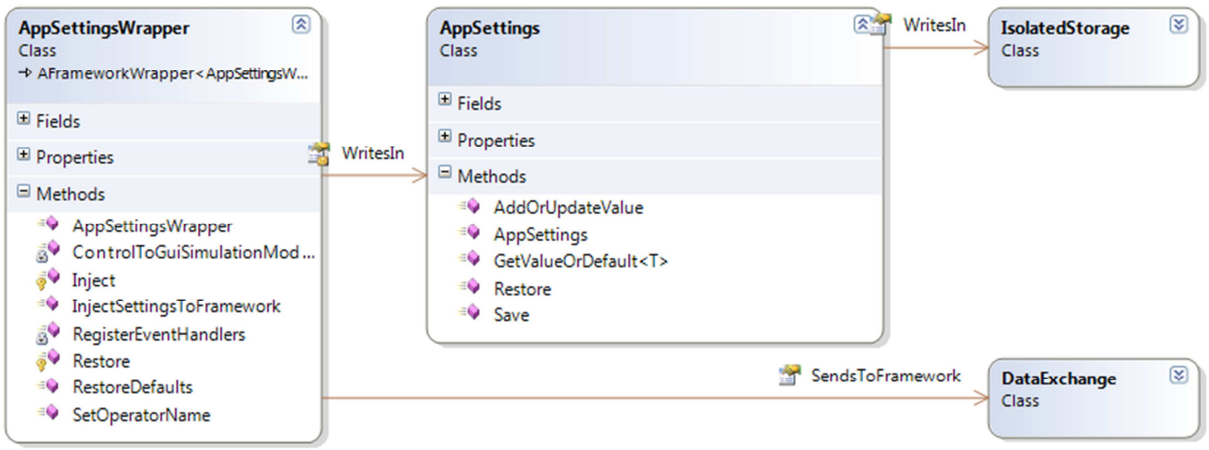


Abbildung 26 Klassendiagramm des AppSettingsWrapper

INJIZIEREN INS FRAMEWORK

Im AppSettingsWrapper befindet sich eine statische Methode `InjectSettingsToFramework`, welche beim Start der App aufgerufen wird, um die Settings des Frameworks mit denen aus dem Isolated Storage abzugleichen.

```
public static void InjectSettingsToFramework()
{
        new AppSettingsWrapper().Inject();
}
```

Listing 16 Initialeinstellungen dem Framework melden

APP SETTINGS

`AppSettings` wurde nach dem Artikel „How to: Create a Settings Page for Windows Phone“ erstellt. Der Artikel ist unter folgendem Link zu finden: http://msdn.microsoft.com/en-us/library/ff769510(v=vs.92).aspx

10.3.2 TEST EXECUTION WRAPPER

Die Verantwortung der Testausführung liegt in dieser Klasse. Sie bietet Methoden zur Steuerung des Ablaufes und zum Abrufen des Ausführungsstatus.

INITIALISIERUNG

Bei Initialisierung der Klasse werden sämtliche Ids der Jobs und DataItems aus dem Framework über die Klasse `JobConfig` geladen:

```
private void SetupDataStructure(JobConfig jobConfig)
{
  foreach (Job job in jobConfig.GetJobIdList().Select(e => new Job(e)))
  {
    JobWrapper.Jobs.Add(job);
    List<int> temporaryDataItems = jobConfig.GetDataItemIdList(job.Id);
    if (temporaryDataItems.Count > 0)
    {
      DataItems.AddRange(temporaryDataItems.Select(e => new DataItemWrapper(e, job)).ToList());
    }
    temporaryDataItems.Clear();
  }
  ExchangeFromGui.RequestGuiUpdate();
}
```

Listing 17 Laden der Ursprungsdaten (Tests, DataItems)

Wie in ***Error! Reference source not found.*** ersichtlich, hält diese Klasse Instanzen von `DataItemWrapper` und `JobWrapper`. Wird `RequestGuiUpdate()` aufgerufen, erhalten alle registrierten EventHandler dieser Klasse und der anderen Wrapper die Daten aus dem Framework. Der Initial-Ladevorgang ist somit abgeschlossen.

STEUERUNG DES ABLAUFS

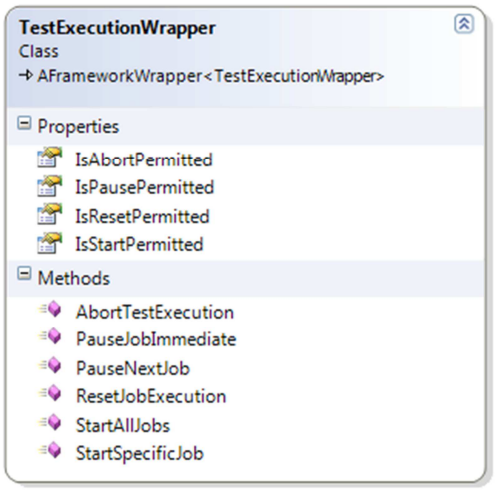
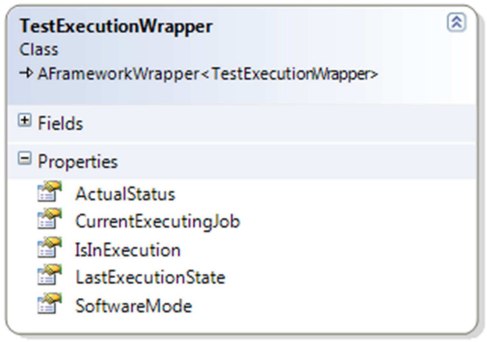
Bild	Beschreibung
	In der Klasse wird das komplette Set der Steuerungsfunktionen des Frameworks gekapselt. Weiter kann anhand des Softwarestatus, den Daten der Jobs und des aktuellen Vorgangs festgestellt werden, welche Aktionen gerade erlaubt sind.

Abbildung 27 Steuerungsrelevante Properties des TestExecutrionWrappers

AUSFÜHRUNGS STATUS

Bild	Beschreibung
 **Abbildung 28 Ausführungsrelevante Properties des TestExecutionWrappers**	Das wichtigste Property für die Ausführung ist `CurrentExecutingJob`. Läuft eine gesamte Testsequenz wird dieser Wert ständig angepasst. `LastExecutionState`: Durch lässt sich evaluieren, ob bereits ein Testvorgang ausgeführt wurde oder nicht und wenn ja, welcher. Wurde nämlich die Aktion abgebrochen ist für eine Wiederaufnahme des Tests wichtig, ob eine ganze Sequenz weitergeführt wird oder nur ein einzelner Test. `ActualStatus`: beschreibt den aktuellen Software Status (siehe 24.3.9 Wichtige Enum) `SoftwareMode`: Wird ein neuer Software Mode gesetzt (Zugriff aufs Framework). In einem anderen Software Mode sind andere oder ein beschränkter Satz an Jobs verfügbar

ANZEIGE FÜR BENUTZERINTERAKTION

Die Angaben für den Benutzer werden je nach aktuellem Job neu gesetzt. Die Klasse `TestExecutionQuest` übernimmt ein solches Set von Informationen, die für die Anzeige eines Jobs und dessen Durchführungs-Status erforderlich sind.

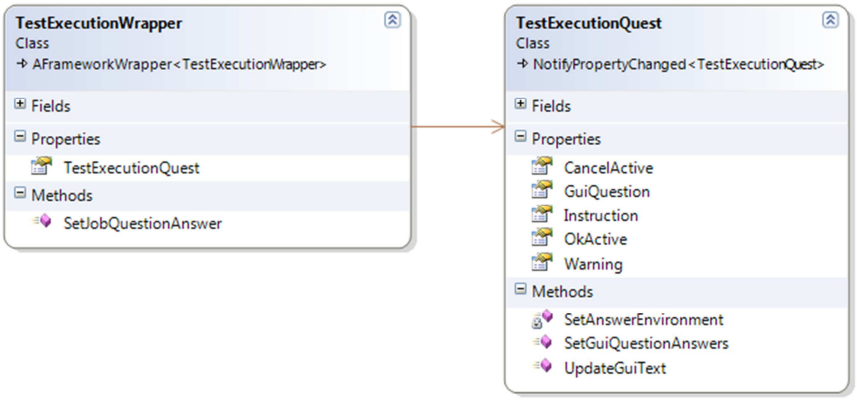


Abbildung 29 Anzeigerelevante Properties des TestExecutionWrappers

Je nachdem ob der User den Grünen oder den Roten Button klickt, übergibt die Methode `SetJobQuestionAnswer` dem Framework eine positive oder negative Antwort für den Testerfolg. Es existieren auch Tests, die automatisch ablaufen, bei denen wird der Erfolg vom Framework gesteuert, es ist kein Feedback des Users erforderlich.

ZUSATZ VON DATEN (DATA ITEMS)

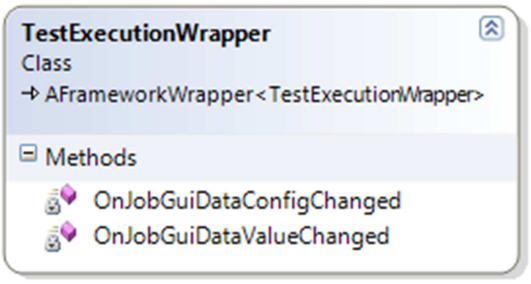


Abbildung 30 JobGuiData-Eventhandler des TestExecutionWrappers

Sobald ein `DataItem` angezeigt werden soll, wird der Event `JobGuiDataConfigChanged` gefeuert. Das bewirkt, dass ein `DataItem` (bis anhin ist nur die Id bekannt) mit den nötigen Informationen gefüllt wird.

```
DataItemWrapper item = GetDataItemByIds(e.DataItemId, e.JobId);
item.Name = e.Name;
item.Unit = e.Unit;
item.IsVisible = e.GuiShow;
item.IsEditable = e.DirectionType == EDataItemDirection.Input;
```

Listing 18 Empfangen der Daten von DataItems

Sobald durch Eingabe des Operators ein Wert im Framework gesetzt wurde, erhält man die validierte Antwort über den `JobGuiDataValueChanged` Event.

```
DataItemWrapper item = GetDataItemByIds(e.DataItemId, e.JobId);
item.IsValid = e.DataValueValid;
item.BackColor = e.DataBackColor;
item.Value = e.DataValue;
```

Listing 19 Validierung der Daten von DataItems

10.3.3 JOB WRAPPER

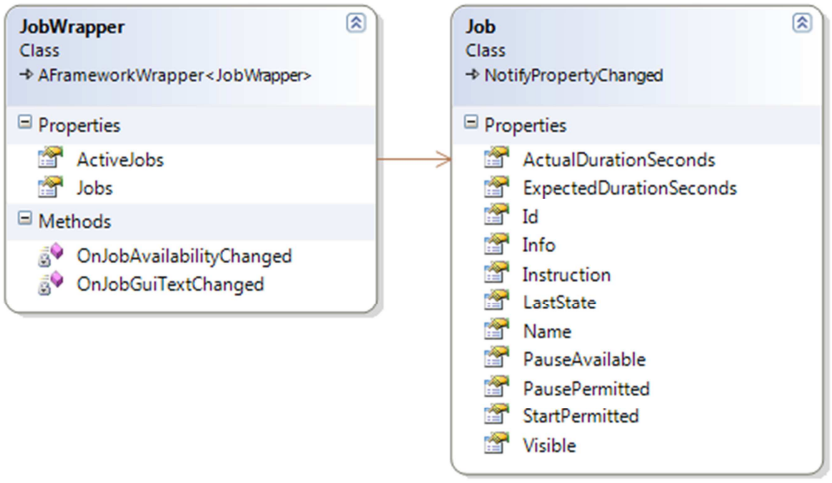


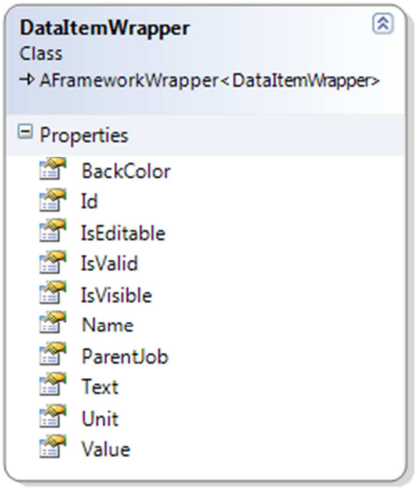
Abbildung 31 Klassendiagramm des JobWrappers

`OnJobGuiTextChanged:` Hier werden alle Details zu den Jobs (nach dessen Initialisierung) erhalten. Diese werden dann in den bereits erstellten Job Objekten gespeichert.

`OnJobAvailabilityChanged:` Dieser Event Handler wird bei einem Wechsel des Software Mode aufgerufen. Die Sichtbarkeit aller Jobs wird angepasst.

`ActiveJobs:` Gibt eine Collection aller Jobs zurück, die im aktuellen Software Mode sichtbar sein sollen.

10.3.4 DATA ITEM WRAPPER



Pro Userinteraktion in einem Test, können mehrere `DataItems` angezeigt werden. Diese besitzen immer einen Titel/Beschreibung (**Text**), einen aktuellen Wert (**Value**) und eine Einheit (**Unit**).

Es gibt `DataItems`, die nur Informationen aus dem Framework zeigen und solche, die für die Eingabe des Benutzers vorgesehen sind. Dies wird über `IsEditable` gesteuert.

Wird `Value` verändert, wird der Wert direkt ins Framework eingespeist. Darum auch die Verwendung der Basisklasse. Nun entsteht eine Validierung der Werte (Beschrieben in `TestExecutionWrapper`)

Ob ein `DataItem` gerade benötigt zeigt das Property `IsVisible`.

Abbildung 32 Klassendiagramm DataItemWrapper

10.3.5 AFILEWRAPPER

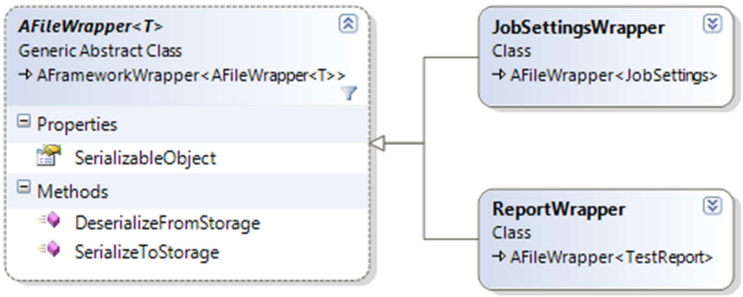


Abbildung 33 Klassendiagramm AFileWrapper

Die abstrakte Klasse `AFileWrapper` speichert Klassen auf dem IsolatedStorage des Phones.

SERIALIZABLE OBJECT

Das generische Property `SerializableObject` wird beim Aufruf der (De-)Serialisierungs-Methoden in den IsolatedStorage geschrieben.

```
public T SerializableObject
{
  get
  {
    if (SerializationFailed) serializableObject = null;
    SerializationFailed = false;
    return serializableObject ?? (serializableObject = Activator.CreateInstance<T>());
  }
  set
  {
    if (serializableObject == value) return;
    serializableObject = value;
  }
}
```

Listing 20 Schreiben eines generischen Properties in Isolated Storage

Beim Zugriff auf das Property wird überprüft, ob bereits eine Serialisierung fehlgeschlagen ist. Sollte dies der Fall sein, wird versucht eine neue Instanz des generischen Typs zu erstellen.

Dieser Mechanismus folgt den gängigen Designregeln bei abstrakten Klassen. Die Verantwortung Daten konsistent zu halten liegt bei der implementierenden Klasse. Das `SerializableObject` repräsentiert lediglich einen DatenContainer der serialisiert und deserialisiert werden kann.

ISOLATEDSTORAGE UND FILES

Um ein File im IsolatedStorage des WindowsPhone zu speichern / laden, wird die `IsolatedStorageFile`-Klasse benötigt. Darüber lässt sich der File-Store des WindowsPhones öffnen.

```
//get file store
IsolatedStorageFile store = IsolatedStorageFile.GetUserStoreForApplication();

store.OpenFile(...);
```

Listing 21 Zugriff auf Isolated Storage erhalten

Voraussetzung dass `AFileWrapper` ein File öffnen kann ist, dass die ableitenden Klassen einen gültigen `Filename` gesetzt haben. Die verschiedenen Modi um ein File zu öffnen sehen wie folgt aus:

```
//create file, throws exception if file exists
IsolatedStorageFileStream file = store.OpenFile(FullFilePath, FileMode.CreateNew)

//open existing file, delete content, throws exception if file does not exist
IsolatedStorageFileStream file = store.OpenFile(FullFilePath, FileMode.Truncate)

//Equal to request CreateNew if file does not exists or use Truncate if it does
IsolatedStorageFileStream file = store.OpenFile(FullFilePath, FileMode.Create)

//open existing file, create new if not found, append to file
IsolatedStorageFileStream file = store.OpenFile(FullFilePath, FileMode.OpenOrCreate)
```

Listing 22 Methoden zum Zugreifen auf Dateien im Isolated Storage

Obige Anweisungen öffnen das File im Write-Only-Mode. Wird anschliessend aus dem Stream gelesen, wird eine `IsolatedStorageException` geworfen. Es empfiehlt sich daher den `FileAccess`-Mode ebenfalls anzugeben um langwierige Fehlersuchen zu vermeiden.

```
//open existing file in read-only mode, throw exception if file not exists
var file = store.OpenFile(FullFilePath, FileMode.Open, FileAccess.Read)
```

USINGS UND EXCEPTIONS

Der Zugriff auf das File wird mit einem `using`-Block gekapselt. Somit wird das File in Falle einer Exception auf jeden Fall wieder geschlossen. Der `try-catch` Block umfasst den `using`-Block des Files, um eine allfällige Exception zu fangen und zu behandeln bevor der Store wieder geschlossen wird.

```
using (IsolatedStorageFile store = IsolatedStorageFile.GetUserStoreForApplication())
{
  try
  {
    using (IsolatedStorageFileStream file = store.OpenFile(FullFilePath,FileMode.Create,FileAccess.Write))
    {
      ...
    } // close file properly (also in case of exception)
  }
  catch (IsolatedStorageException ex)
  {
    SerializationFailed = true;
    ...
  }
} //close store after handling exception
```

Listing 23 Exception Handling beim Zugriff auf Isolated Storage

SERIALISIERUNG / DESERIALISIERUNG

`AFileWrapper` verwendet die statische Klasse `FileSerializer` für die Serialisierung / Deserialisierung.

```
public void SerializeToStorage()
{
  ...
  using (var file = store.OpenFile(FullFilePath, FileMode.Create))
  {
    FileSerializer.Serialize(SerializableObject, file);
    SerializationFailed = false;
  }
  ...
}
```

Listing 24 Verwendung der Klasse FileSerializer

```
public static void Serialize<T>(T serializableObject, IsolatedStorageFileStream file) where T : class
{
  new SmartSerializer<T>().Serialize(serializableObject, file);
}
```

Listing 25 Delegierung an Klasse SmartSerializer

Der `FileSerializer` delegiert das Objekt das serialisiert werden soll, zusammen mit dem `IsolatedStorageFileStream`, an die Klasse `SmartSerializer<T>.` Dieser unterscheidet zwischen Objekten mit `DataContract`-Attribut und solchen ohne.

```
public void Serialize(T serializableObject, Stream file)
{
  if (IsDataContractTemplate(typeof(T)))
    DataContractSerializer.WriteObject(file, serializableObject);
```

```
  else
    XmlSerializer.Serialize(file, serializableObject);
  file.Flush();
}
```

Listing 26 Festsellen der Serialisierbarkeit

Dieses double dispatching führt dazu, dass das File vom `SmartSerializer` als Stream entgegen genommen werden kann.

Um herauszufinden ob der generische Typ T mittels `DataContractSerializer` geschrieben werden kann, wird folgende Methode benutzt.

```
private static bool IsDataContractTemplate(Type type)
{
  try { new DataContractSerializer(type).WriteObject(new MemoryStream(),Activator.CreateInstance(type)); }
  catch (Exception ex) { return false; }
  return true;
}
```

Listing 27 Feststellen ob Typ serialisiert werden kann

10.3.6 REPORT WRAPPER

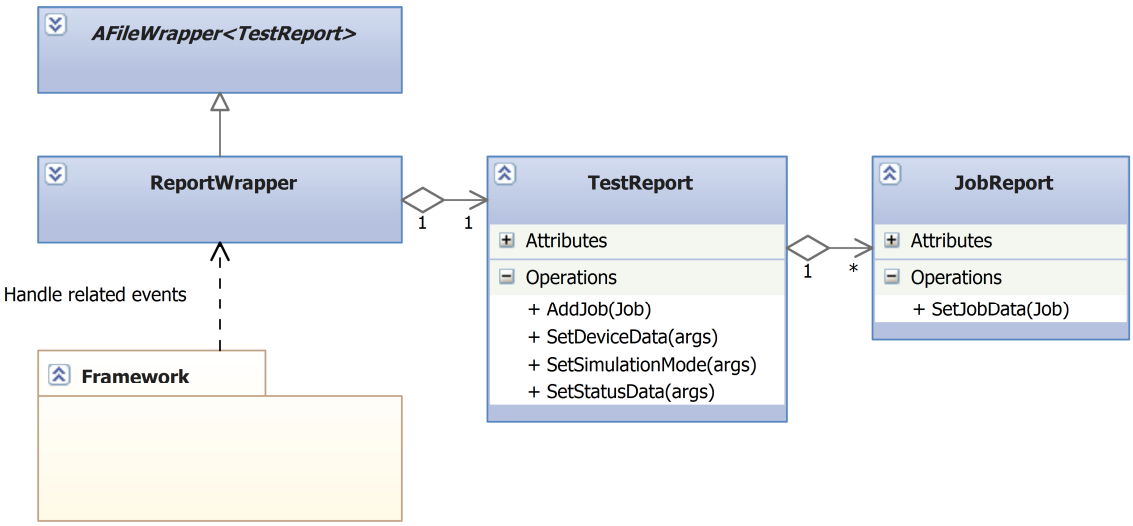


Abbildung 34 Klassendiagramm ReportWrapper

Registriert sich auf die Events vom Framework die für das Reporting interessant sind.

```
private void RegisterEventHandlers()
{
  ExchangeToGui.DeviceDataChanged += ExchangeToGuiDeviceDataChanged;
  ExchangeToGui.ControlDataChanged += ExchangeToGuiControlDataChanged;
  ExchangeToGui.ControlStatusChanged += ExchangeToGuiControlStatusChanged;
  ExchangeToGui.SoftwareModeChanged += ExchangeToGuiSoftwareModeChanged;
  ExchangeToGui.SimulationModeChanged += ExchangeToGuiSimulationModeChanged;
}
```

Listing 28 Registrieren für das Empfangen von Framework Daten

Die Eventhandler speichern die EventArgs in die dafür vorgesehenen Properties und rufen `CacheSerialisation` auf.

```
private void ExchangeToGuiDevicDataChanged(object sender, DeviceDataChangedEventArgs e)
{
  DeviceDataArgs = e;
```

```
  CacheSerialisation();
}
```

Listing 29 Reports: Event Handler bei Datenempfang

Die Methode aktualisiert sämtliche Daten des Reports und speichert den `TestReport` als XML. Das XML könnte nun per Email versendet werden. Es entspricht einem vollwertigen Report.

```
private void CacheSerialisation()
{
  lock (lockMe)
  {
    if (!isReporting) return;
    SetReportData();
    SetJobsData();
    SetJobData(CurrentJob);
    ObjectXml = CurrentTestReport.Serialize();
  }
}
```

Listing 30 Speichern als XML

Erst wenn von aussen die `SerializeToStorage` Methode aufgerufen wird, wird der Testreport auf dem Phone gespeichert.

Klasse	Beschreib
TestReport	Soll ein Jobresultat gespeichert werden, muss der Job an `AddJob()` übergeben werden.

```
public void AddJob(Job job)
{
  var jobReport = new JobReport(this, job);
  if (!ReportedJobs.Contains(jobReport))
  {
    ReportedJobs.Add(jobReport);
  }
  else
  {
    int ix = ReportedJobs.IndexOf(jobReport);
    ReportedJobs.Remove(jobReport);
    ReportedJobs.Insert(ix, jobReport);
  }
}
```

Listing 31 Erzeugt ein Set aus Testreports

Um die `Contains`-Methode verwenden zu können musste die `Equals`-Methode der `JobReport`-Klasse überschrieben werden.

Der Testreport trägt ein `DataContract`-Attribut. Ein `DataCotract` erlaubt es, mittels `DataMember`-Attributen zu bestimmen, welche Felder serialisiert werden sollen und welche nicht.

```
[DataContract]
public class TestReport {
  ...

  public string Filename { get; set; }

  [DataMember]
  public int ReportDataId { get; set; }

   [DataMember]
  public List<JobReport> ReportedJobs { get; set; }
}
```

Listing 32 Attributierung des TestReport Objekts

Die Properties mit `DataMember`-Attribut müssen `public` sein. Ansonsten funktioniert die Serialisierung nicht richtig.

Der Nutzen des `DataContract`-Attributs wird im folgenden Listing klar.

XmlSerializer	DataContractSerializer

`<TestReport>`
`  <Filename>D:\Test\test.xml</Filename>`
`  <ReportDataId>1</ReportDataId>`
`  ...`
`</TestReport>`
	`<TestReport x:DataContract=...>`
	`  <ReportDataId>1</ReportDataId>`
	`  ...`
	`</TestReport>`

Figure 1 Serialisierter TestReport

Der `DataContractSerializer` schreibt nur diejenigen Properties mit einem `DataMember`-Attribut während der `XmlSerializer` alle `Public`-Properties serialisiert.

JobReport

Die `JobReport` Klasse hält die Informationen über den Ausgang eines Jobs. Sie verfügt über kein `DataContract`-Attribut und wird somit „as-is“ serialisiert. Sie überschreibt die `Equals` Methode um zusätzlich zur Referenzgleichheit noch auf JobId- und JobName-Gleicheit zu überprüfen.

```
public bool Equals(JobReport other)
{
  if (ReferenceEquals(null, other)) return false;
  if (ReferenceEquals(this, other)) return true;
  return other.JobId == JobId && Equals(other.JobName, JobName);
}
```

Listing 33 Equals Methode eines JobReports

10.3.7 JOB SETTINGS WRAPPER

Für das Verständnis dieses Wrappers sollte die `JobSettings`-Klasse bekannt sein.

ANZEIGE DER JOBSETTINGS

Die `JobSettings` werden auf der `JobSettingsPage` angezeigt.

```
<controls:Pivot x:Name="pivot" Title="{Binding Strings.PageTitleJobSettings}">
      <controls:PivotItem Header="logging">
        <Controls:JobSettingsPropertyControl DataContext="{Binding LoggingProperties}" />
      </controls:PivotItem>

      <controls:PivotItem Header="firmware">
        <Controls:JobSettingsPropertyControl DataContext="{Binding FirmwareProperties}" />
      </controls:PivotItem>
...
```

Listing 34 Binding der JobSettings

Die `JobSettingsPropertyControl` erwartet eine Liste `IJobSettingProperties` diese werden dann untereinander angezeigt.

Einzelnes Boolean Property	Mehrere Properties
	
Abbildung 35 Boolean Property JobSettingsUI	Abbildung 36 String Properties JobSettingsU

KLASSENSTRUKTUR

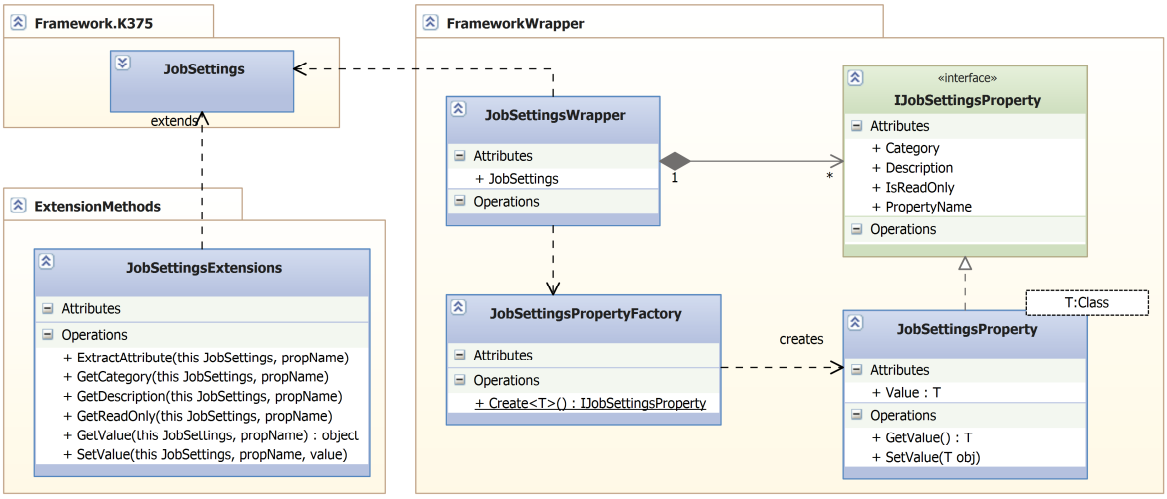


Abbildung 37 Klassendiagramm JobSettingsWrapper

Der `JobSettingsWrapper` bietet alle Properties der `JobSettings`-Klasse verpackt als `IJobSettingsProperty` an. Er benutzt dazu die statische Methode `Create()` der `JobSettingsPropertyFactory`.

```
public IJobSettingsProperty LogTestDetails
{
  get { return JobSettingsPropertyFactory.Create(SerializableObject, m => m.LogTestDetails); }
}
```

Listing 35 Logging der Testdetails

```
public static IJobSettingsProperty Create<T>(JobSettings settings,
                                    Expression<Func<JobSettings, T>> expression)
{
  return new JobSettingsProperty<T>(settings, expression);
}
```

Listing 36 Create-Method on JobSettingsPropertyFactory

```
public JobSettingsProperty(JobSettings settings, Expression<Func<JobSettings, T>> expression)
{
  Settings = settings;
  this.expression = expression;
  Description = settings.GetDescription(PropertyName);
  Category = settings.GetCategory(PropertyName);
  IsReadOnly = settings.GetReadOnly(PropertyName);
}
```

Listing 37 JobSettingsProperty constructor extracting property information from JobSettings

Das Konstrukt `Expression<Func<JobSettings,T>>` erlaubt es, dem `JobSettingsProperty` herauszufinden, welches Property es auf der `JobSettings`-Klasse es kapselt. Der Name des Property kann folgendermassen ausgelesen werden.

```
public string PropertyName
{
  get
  {
    if (propertyName == null)
    {
      var memberExpr = expression.Body as MemberExpression;
      if (memberExpr == null)
        throw new ArgumentException("Expression must map to a property name");
      propertyName = memberExpr.Member.Name;
    }
    return propertyName;
  }
}
```

Listing 38 Ermitteln des PropertyName

EINSATZ VON REFLECTION

`PropertyName` wird genutzt, um die Werte die dem `Value`-Property des `JobSettingsProperty` übergeben werden zu setzen, bzw. die Werte im UI anzuzeigen.

```
public T Value
{
  get { return GetValue(); } set { SetValue(value); }
}

public T GetValue()
{
  return (T) Settings.GetValue(PropertyName, typeof (T));
}

public void SetValue(T value)
{
  Settings.SetValue(PropertyName, value);
}
```

Listing 39 Setzen und Lesen der Settings

Das `Value`-Property der `JobSettingsProperty`-Klasse kapselt einen Aufruf auf eine ExtensionMethode der `JobSettings`. Das folgende Listing zeigt diese ExtensionMethoden.

```
public static object GetValue(this JobSettings settings, string propertyName, Type returnType)
{
  Type t = settings.GetType();
  return t.GetProperty(propertyName, returnType).GetValue(settings, null);
}

public static void SetValue(this JobSettings settings, string propertyName, object value)
{
  Type t = settings.GetType();
```

```
  t.GetProperty(propertyName).SetValue(settings, value, null);
}
```
Listing 40 Extension Methode zur Kapselung

Um die Attribute der JobSettings-Properties auszulesen, bedarf es einer komplexeren Methode. Die Properties haben alle folgende drei Attribute.

```
[Category("Logging")]
[Description("Log values to the internal log file during test run.")]
[ReadOnly(false)]
public bool LogTestDetails { get; set; }
```
Listing 41 Tagging durch Attrbute der Settings

Diese Attribute sind durch ihre Typen eindeutig identifizierbar und können mit Reflection wie folgt ausgelesen werden.

```
public static string GetDescription(this JobSettings settings, string propertyName)
{
  return ExtractAttribute<DescriptionAttribute>(propertyName).Description;
}

public static string GetCategory(this JobSettings settings, string propertyName)
{
  return ExtractAttribute<CategoryAttribute>(propertyName).Category;
}

public static bool GetReadOnly(this JobSettings settings, string propertyName)
{
  return ExtractAttribute<ReadOnlyAttribute>(propertyName).IsReadOnly;
}

private static TAttributeType ExtractAttribute<TAttributeType>(string propertyName)
{
  Type t = typeof (JobSettings);
  PropertyInfo propertyInfo = t.GetProperty(propertyName);
  object[] attributes = propertyInfo.GetCustomAttributes(typeof (TAttributeType), true);
  if (attributes.Length == 0) throw new InvalidDataContractException("extracting attribute failed");
    var desc = (TAttributeType) attributes[0];
  return desc;
}
```
Listing 42 Auslesen der JobSettings Attribute

Diese Extension-Methoden kommen bereits im Listing 37 zum Einsatz.

GRUPPIERUNG IN KATEGORIEN

Da JobSettings auf dem UI in Kategorien unterteilt angezeigt werden, bietet der `JobSettingsWrapper` die Properties bereits in Kategorien geordnet an.

10.3.8 SCHWIERIGKEITEN

CROSS-THREAD ACCESS

Werden `RaisePropertyChanged`-Events von einem anderen Thread als dem des UI geworfen, so wird eine Exception „Cross-Thread-Access“ ausgelöst. Der Grund hierfür ist, dass nur der UI Prozess ein Update der Bindings vornehmen kann. Aus sämtlichen Wrapper-Klassen werden Änderungen von Properties über folgende Extension-Methode gemeldet.

```
public static void SafeRaisePropertyChanged<TMyself, T>(
  this NotifyPropertyChanged<TMyself> propertyHolder, Expression<Func<TMyself, T>> expression)
{
  if (!Deployment.Current.Dispatcher.CheckAccess())
  {
```

```
    Deployment.Current.Dispatcher.BeginInvoke(
      (ThreadStart) (() => propertyHolder.RaisePropertyChanged(expression)));
  }
  else
  {
    propertyHolder.RaisePropertyChanged(expression);
  }
}
```

Listing 43 SafeRaisePropertyChanged: Umgang mit Cross-Thread-Exceptions

Es wird sichergestellt, dass der aktuelle Thread dem UI-Dispatcher zugewiesen ist. Falls nicht kann mit Deployment.Current.Dispatcher auf diesen zugegriffen werden. Damit wird `RaisePropertyChanged()` immer auf dem Dispatcher-Thread aufgerufen, wodurch keine Cross-Thread-Exception mehr ausgelöst wird.

MULTIPLE DATA UPDATES

Diverse Events werden zur Laufzeit vom Framework gefeuert. Ausserdem müssen die Wrapper auch die Methode „RequestGuiUpdate“ aufrufen, da dies die einzige Möglichkeit ist Daten vom Framework zu erhalten. Der Aufruf dieser Methode führt jedoch dazu, dass sämtliche Events des Frameworks gefeuert werden. Um diesen Event-Sturm zu reduzieren wurden folgende Mechanismen eingeführt.

1. Es darf nicht vorkommen, dass mehrere Threads gleichzeitig die Properties verändern
    a. Es wurde ein einfacher Lock eingeführt
2. Es darf nicht mehrmals hintereinander auf Events mit den gleichen Daten (Inhalte der EventArgs) reagiert werden
    a. Die letzten EventArgs werden zwischengespeichert
    b. „ObjectPropertiesComparer“ wird aufgerufen, der mittels Reflection sämtliche Public Properties vergleicht, ob identische Werte vorliegen.
3. Es soll nur auf Events reagiert werden, deren Daten benötigt werden
    a. Es werden Checks eingeführt wie im Beispiel unten „IsCurrentJobMeant“, hier wurden Events pro Job empfangen, welche auf das Laufzeitverhalten keinen Einfaluss hatten. Dieser Ansatz ist schwierig zu verfolgen, das KnowHow der Problemdomäne im Detail ist gefordert

```
private JobGuiDataValueChangedEventArgs args;
private void OnJobGuiDataValueChanged(object sender, JobGuiDataValueChangedEventArgs e)
{
  lock (threadLock)
  {
    if (!ObjectPropertiesComparer.PublicInstancePropertiesEqual(e, args))
    {
      args = e;
      if (IsCurrentJobMeant(e.JobId))
      {
        //CODE REMOVED
        this.SafeRaisePropertyChanged(myself => myself.Property);
      }
    }
  }
}
```

Listing 44 Optimierungen der Event Handler

11 UMSETZUNG DER KOMMUNIKATION

11.1 PROBLEM DOMÄNE

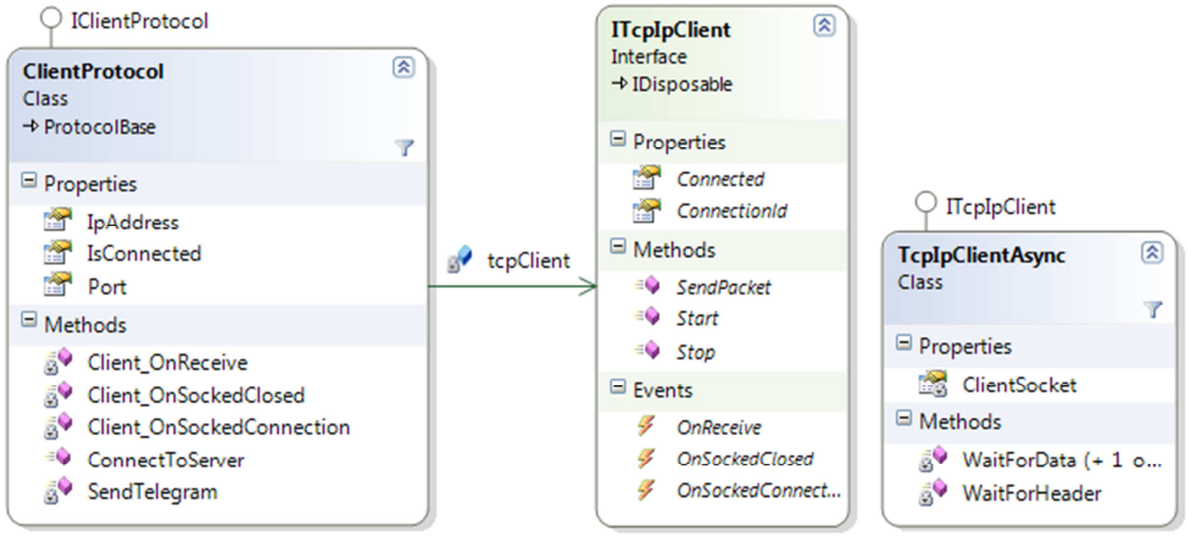


Abbildung 38 Klassendiagramm TcpIpClientAsync

Die Klasse `ClientProtocol` wird vom Framework dazu verwendet, sogenannte Telegramme über einen Socket an ein verbundenes Kjeldahlgerät zu senden. Die oberen Schichten stellen die Telegramme zusammen und übergeben diese der `ClientProtocol-Methode SendTelegram().` Dort werden sie zu byte-Arrays serialisiert und dem TcpClient übergeben. Jede Klasse die das Inerface `ITcpIpClient` implementiert wird vom `ClientProtocol` als TcpClient akzeptiert.

11.2 ASYNCHRONE SOCKET PROGRAMMIERUNG

Das Framework von Büchi setzt zur Kommunikation eine Implementation von ITcpIpClient ein, die synchrone Sockets verwendet. WindowsPhone 7 unterstützt aber nur asynchrone Socketoperationen. Um die Kommunikation zwischen Phone und Gerät sicherzustellen musste daher ein eigener TcpIpClient entwickelt werden der asynchrone Socketoperationen verwendet.

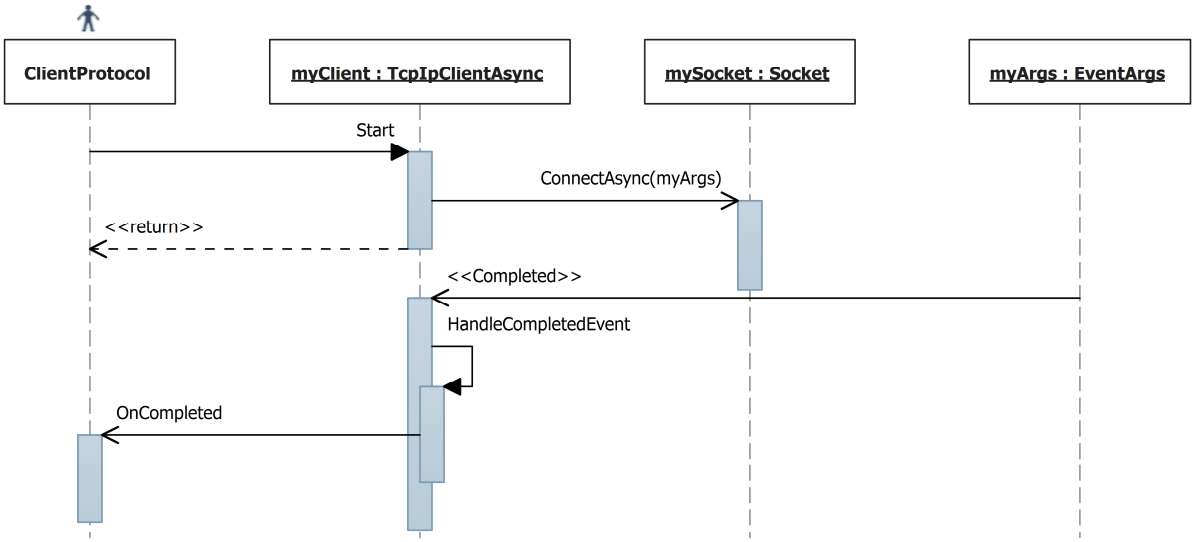


Abbildung 39 Sequenzdiagramm eines asynchronen Verbindungsaufbaus (WP7 Socket)

11.2.1 CLIENT VORBEREITEN

Als erstes wird im ClientProtocol ein Client erstellt der einen Socket öffnet worüber Daten gesendet und empfangen werden können. Da der Socket-Konstruktor einen EndPoint erwartet, werden TcpIpClientAsync eine Adresse und ein Port übergeben. Daraus kann ein IpEndPoint erstellt werden.

```
public TcpIpClientAsync(IPAddress ipAddress, int port, int packetSize)
{
  ...
  ipEndPoint = new IPEndPoint(ipAddress, port);
  ...
}
```

Listing 45 Vorbereitung eines asynchronen TCP-IP Client

Der Socket muss wie folgt konfiguriert werden um eine Verbindung aufbauen zu können. Für eine genaue Beschreibung des Socket wird auf die MSDN-Dokumentation http://msdn.microsoft.com/en-us/library/attbb8f5.aspx verwiesen.

```
Socket clientSocket = new Socket(AddressFamily.InterNetwork, //IPv4
                                 SocketType.Stream,
                                 ProtocolType.Tcp);
```

Listing 46 Erstellen des Sockets

11.2.2 IMPLEMENTIERUNG VON ITCPIPCLIENT

VERBINDUNG AUFBAUEN

Die Verbindung zum IpEndPoint kann wie folgt aufgebaut werden. Folgendes Listing zeigt, wie die SocketAsyncEventArgs eingesetzt werden.

```
public void Start()
{
  if (Connected) return;
  CheckForEndpoint();
  var connectionArgs = new SocketAsyncEventArgs {RemoteEndPoint = ipEndPoint};
  connectionArgs.Completed += ConnectionArgsCompleted;
  ClientSocket.ConnectAsync(connectionArgs);
}
```

Listing 47 Verbindung zum Endpoint aufbauen

Sobald der Completed Event geworfen wird, behandelt der TcpIpClient den Event wie folgt:

```
private void ConnectionArgsCompleted(object sender, SocketAsyncEventArgs e)
{
  RaiseConnectionEvent(e.SocketError.ToString());
  if (e.SocketError != SocketError.Success) return;
  WaitForHeader();
}
```

Listing 48 ConnectionCompleted EventHandler

Wenn der Verbindungsaufbau erfolgreich war wird auf die Header-Daten gewartet.

AUF DATEN WARTEN

Warten ist die Stärke jeder asynchronen Implementierung. Mit dem gleichen Mechanismus wie bereits die Verbindung aufgebaut wurde, kann der Socket nach eingehenden Daten abgefragt werden.

```
private void WaitForHeader()
{
  if (!Connected) return;
  var receiveHeaderArgs = new SocketAsyncEventArgs();
  receiveHeaderArgs.Completed += ReceiveTcpIpPacketHeader;
  receiveHeaderArgs.SetBuffer(headerBuffer, 0, headerBuffer.Length);
  ClientSocket.ReceiveAsync(receiveHeaderArgs);
}
```

Listing 49 Empfangen von Daten

PAKETE EMPFANGEN

Pakete bestehen immer aus einem Header- und manchmal auch noch aus einem Payload-Teil. Über den Header kann die Payload-Länge extrahiert werden.

Paket	Beschreibung
Header only	Empfangene Telegrammlänge = 0.
Header with payload	Empfangene Telegrammlänge > 0

Empfangen werden jedoch keine Header oder Payloads, sondern, als byte-Array serialisierte Telegramme.

```
private void ReceiveTcpIpPacketHeader(object sender, SocketAsyncEventArgs socketAsyncEventA
rgs)
{
  ...
  if (!Connected) return;
  int telegramLen = BitConverter.ToInt32(headerBuffer, 12);
  if (telegramLen > 0)
  {
    WaitForData(0, telegramLen);
  }
  else
  {
    var args = new TcpIpClientReceiveEventArgs(ConnectionId, headerBuffer);
    OnReceive(this, args);
    WaitForHeader();
  }
}
```

Listing 50 Entgegennehmen von empfangenen Daten

Wenn der Header keinen Payload ankündigt, wird OnReceive aufgerufen und auf den nächsten Header gewartet.

Andernfalls wird WaitForData aufgerufen. Der ReceiveBuffer auf dem Socket wird auf die ermittelte Telegramm-Länge gesetzt und die Payload-Daten werden aus dem Buffer des Socket geholt, bis der Receive-Buffer leer ist. Dann wird OnReceive aufgerufen und wieder auf Header-Daten gewartet.

PAKET SENDEN

```
public void SendPacket(byte[] data)
{
  ...
  var sendArgs = new SocketAsyncEventArgs();
  sendArgs.SetBuffer(data, 0, data.Length);
  ClientSocket.SendAsync(sendArgs);
  ...
}
```

Listing 51 Senden von Daten über Socket

Für eine genaue Beschreibung der Socket-Klasse sei auch hier auf die MSDN-Dokumentation verweisen.
http://msdn.microsoft.com/en-us/library/attbb8f5.aspx

11.3 EINGRIFFE IN CLIENTPROTOCOL

Es mussten drei Eingriffe in der ClientProtocol-Klasse gemacht werden. Die Änderungen sind **FETT** markiert.

11.3.1 CLIENT AUSWECHSELN

Der TcpClient wurde ausgetauscht. Da der asynchrone Client bei der Initialisierung einen asynchronen Verbindungsaufbau macht, musste OnSockedConnection() als neuer Event hinzugefügt werden.

```
private void Init(IPAddress ipAddress, int port)
{
  this.ipAddress = ipAddress;
  this.port = port;
  tcpClient = new TcpIpClientAsync(ipAddress, port, PacketSize);

  tcpClient.OnReceive += Client_OnReceive;
  tcpClient.OnSockedClosed += Client_OnSockedClosed;
  tcpClient.OnSockedConnection += Client_OnSockedConnection;
  ...
```

Listing 52 Einbauen des neuen TcpIpClients

11.3.2 VERBINDUNGSAUFBAU DELEGIEREN

Client-Protocol muss nun den erfolgreichen Verbindungsaufbau im EventHandler nach oben delegieren.

```
private void Client_OnSockedConnection(object sender, TcpIpClientConnectEventArg e)
{
  if (OnSockedConnected != null)
  {
    var client = (ITcpIpClient) sender;
    if (client.Connected)
    {
      var args = new ProtocolConnectionEventArgs(e.ConnectionId);
      OnSockedConnected(this, args);
    }
  }
}
```

Listing 53 Verbindungsaufbau delegieren

11.3.3 AUFRÄUMEN

Der neue EventHandler muss auch wieder aufgeräumt warden.

```
protected override void DoDisposeWork()
{
  ...
  tcpClient.OnReceive -= Client_OnReceive;
  tcpClient.OnSockedClosed -= Client_OnSockedClosed;
  tcpClient.OnSockedConnection -= Client_OnSockedConnection;
  ...
```

Listing 54 Aufräumen von Socket Event Handler

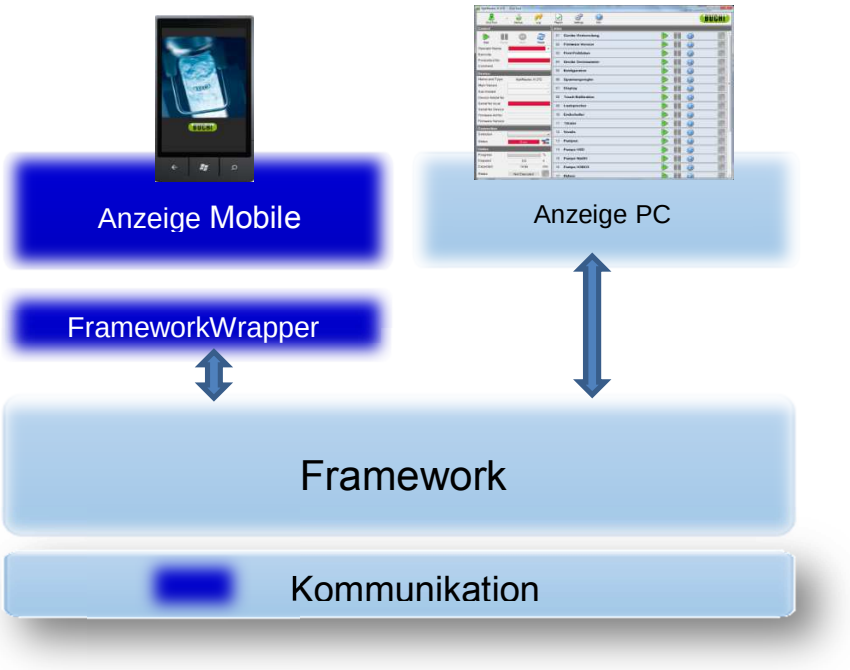
12 SCHLUSSFOLGERUNGEN

12.1 WAS WURDE ERREICHT

Wir konnten den Geschäftsleitungen beider Firmen einen funktionierenden Prototyp demonstrieren, der alle bisherigen Funktionalitäten der PC-Software auf einer WP7 Umgebung beweist. Dies war das wohl wichtigste Ziel dieser Arbeit und alle anderen eine logische Konsequenz daraus. Die Erwartungen der Geschäftsleitungen wurden vollumfänglich erfüllt.

12.1.1 PORTIERUNG DES TEST-FRAMEWORK AUF WP7

Durch die Portierung des Test-Frameworks und der Kommunikationsschicht wurde erreicht, dass die Layer nun für beide Projekte (PC-Software und WP7 App) benutzt werden können. Sollte es zu einer Förderung der WP7 App kommen, ist dies ein tragender Kostenfaktor. Es stellt ein Gewinn an Wartbarkeit und auch Einsparungen an Entwicklungskosten bei einem Ausbau dar.



VERWEIS IN DEN TECHNISCHEN BERICHT

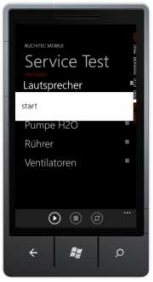
12.1.2 UMSTELLEN AUF ASYNCHRONE KOMMUNIKATION

Höchste Priorität hatte bei Projektstart die Kommunikation zu den Büchi-Geräten herzustellen. Wir wussten, dass das Framework nur synchrone Kommunikation unterstützt und dass wir eine eigene Kommunikationsschicht implementieren müssen. Wir haben es geschafft, mit nur einer neuen Klasse das Framework auf asynchrone Kommunikation umzustellen.

12.1.3 NEUDESIGN DES UI

Aus einer detaillierten Analyse des GUI der PC-Software und in Gesprächen mit einem Servicetechniker persönlich entstand ein komplett neues aber ähnlich bedienbares User Interface. In einer Analyse wurde dargelegt auf welche Teile der PC-Software bewusst verzichtet werden konnte und wie aus unseren Ideen, in Absprache mit Büchi AG, die neue Lösung

entstanden ist. Um einen Eindruck des Look and Feel der Applikation zu erhalten sind nachfolgend alle Screen die im Lauf der Arbeit entwickelt wurden dargestellt.



12.2 BEURTEILUNG

12.2.1 PORTIERUNG

Wir haben gezeigt, dass das bestehende Framework portierbar ist.

Die Portierung auf das Windows Phone bringt der Firma Büchi einen wichtigen Einblick in die mittel- bis langfristigen Arbeiten. Aus unseren Erfahrungen können wichtige Guidelines für die Architektur der zukünftigen Frameworks gezogen werden.

Das Framework wurde nicht für mobile Anwendungen erstellt und ist daher keine optimale Lösung.

12.2.2 KOMMUNIKATION

Wir haben gezeigt, dass die Kommunikationsschicht austauschbar ist.

Es macht keinen Unterschied ob das Framework synchron oder asynchron mit den Geräten kommuniziert. Die Komponenten sind austauschbar und wir zeigen mit der TcpIpClientAsync-Klasse wie ein simpler asynchroner TcpIp-Client aussehen kann.

12.2.3 USER INTERFACE

Wir haben gezeigt, dass sich die Servicetechniker App von einem mobilen Gerät aus bedienen lässt.

Es ist uns gelungen ein komplexes Userinterface auf einen simplen Navigations-Pfad zu bringen. Die Navigation kann auf andere Plattformen wie iPhone und Android übernommen werden. Das Userinterface ist für einen unerfahrenen Benutzer nicht 100% selbsterklärend. Da die Benutzer aber sowieso erst geschult werden müssen entfällt diese Anforderung.

12.3 AUSBLICK

12.3.1 FORTFÜHRENDE ARBEITEN

Reports werden zurzeit nur als XML gespeichert. Da unsere eigenen Objekte serialisiert werden und wir DataContract-Klassen serialisieren können liegt der Schritt zu einer Upload-funktionalität in Form einer Service-Architektur (z.B mit WCF) nahe. Dies könnte in wenigen Stunden implementiert werden.

12.3.2 EMPFEHLUNGEN

Wir haben das Framework der PC-Test-Software ins Detail kennengelernt. Wir haben Unschönheiten im Code nicht verbessert. Stattdessen sind folgende Empfehlungen entstanden.

TEST-FRAMEWORK

Empfehlung	Beschreib
Refactoring des Codes	Generell sind viele Unschönheiten im Code des Frameworks zu finden. Dies ist bei umfangreichen Frameworks beinahe unvermeidlich. Wir empfehlen dennoch ein professionelles Software-Refactoring. Wir schlagen vor, das Framework mit Hilfe eines modernen und professionellen Frameworks wie PRISM **neu zu erstellen**. Uns ist bewusst dass dies sehr Aufwendig ist, aber wenn die aktuellen Unschönheiten weiterentwickelt werden, wird sich der Innovationsgrad der IT verringern und als Folge davon die Kosten der Entwicklung erhöhen.
Top Down Hierarchie anstatt Bottom Up	Das PC-Test-Framework hat einen stabilen Core. Verbesserungswürdig sind auch die Events die vom Framework an die User-Controls gesendet werden. (Bottom Up) Moderne Software-Architektur versucht UI und Daten völlig voneinander zu trennen. Im vorliegenden Fall ist das UI Teil des Frameworks, was unserer Meinung nach in Zukunft zu Problemen führen wird. Wir empfehlen eine Migration von WinForms auf WPF. Damit wäre ein erster Schritt hin zu einer modernen Architektur getan. Die weiteren Schritte würden sich automatisch ergeben bei konsequenter Weiterentwicklung.
Primitive Obsession auflösen	Datenupdates, die fürs UI bestimmt sind und per Event gefeuert werden, sollten nicht mit Id's identifiziert werden, stattdessen sollte eine Model Schicht die Objekte (Daten) des Frameworks kennen und nur noch gemeldet werden, welche Daten neu geladen werden müssen. Bis anhin müssen sämtliche Objekte, wenn diese nicht wie bisher direkt ans UI gebunden werden, zur Datenhaltung neu erzeugt werden, da Schnittstellenobjekte ausschliesslich EventArgs sind.

Memory Leaks vermeiden	Der Einsatz von State Machines ist ein gängiges Pattern im Framework, das aber nicht immer mit Geschick eingesetzt wurde. Ständige Loops werden ausgeführt. Im Falle eines Fehlers in diesen Loops, wird sofort ein Memory Leak erzeugt, da auch der Fehler unendlich vielmal auftritt. Ausserdem schlägt sich dies auf die Performance nieder.
Konsistenz bei konfigurierbaren Daten	Wenn schon die Möglichkeit besteht JobSettings aus XML Dateien einzulesen, sollten auch die Default Einstellungen in Form eines DefaultJobSettings.xml angeboten werden, um nicht wie bis anhin auch noch aus einem SettingsFile Daten lesen zu müssen. Das würde wiederum das Problem, dass auf WP7 (Silverlight) keine Settings-Files unterstützt werden lösen.
Simulator Prozess für K375 Test optimieren	Der Simulations Prozess liefert für die Testausführung teils unkorrekte Daten zurück, die Tests schlagen fehl. Für die Entwicklung der End Test Software ist dieser nur mässig nützlich, da keine Gewissheit besteht, ob der Fehler bei der eigenen Programmierung oder im Simulator besteht.

DOKUMENTATION

Empfehlung	Beschreib
Doku nachführen	Communication Layer Dokumentieren
Klassendiagramme	Klassendiagramme nicht so detailliert halten, dafür wichtige Bereiche in Dokumentation mit Klassendiagramm verbinden

12.3.3 VERBLEIBENDE PROBLEME

Da es sich um eine Machbarkeitsstudie handelt wurden gewisse Aspekte nicht berücksichtigt. Es können selten Stabilitätsprobleme bei der Bedienung auftreten. Auch funktioniert die Datenvalidierung nicht immer korrekt, da nicht alle Datenquellen (Resources) in unser Projekt eingefügt wurden.

12.3.4 MÖGLICHE WEITERENTWICKLUNGEN

Es wäre interessant diese Innovation auch auf ähnlichen Plattformen wie IPhone oder Android benutzen zu können. Ein einfacher Schritt um dies zu erreichen wäre die Portierung auf das .NET basierte MonoTouch Framework.

Reports sollten nach Abschluss der Arbeiten direkt auf einen Server geladen werden können. In unserer Arbeit wurde nur gewährleistet, dass diese Weiter könnten daraus in der Masse Rückschlüsse auf einzelne Verbrauchsteile, Herkunftsländer oder die Verarbeitungsqualität geführt werden.

Kjeldahl Servicetechniker Mobile App

Machbarkeitsstudie

Projektplan

13 EINFÜHRUNG

13.1 ZWECK

Dieses Dokument beschreibt den Prozess der Entwicklung der mobilen Servicetechniker App und gibt einen detaillierten Projektüberblick.

13.2 GÜLTIGKEITSBEREICH

Dieses Dokument dient als Grundlage für die Bachelorarbeit „Mobile Servicetechniker Applikation“ und hat Gültigkeit über die gesamte Projektdauer.

13.3 DEFINITIONEN UND ABKÜRZUNGEN

Siehe Glossar.

13.4 ÜBERSICHT

Nachfolgend wird das Projektteam, die Organisationsstruktur der Arbeit sowie das bereits bestehende Test-Framework der Firma Büchi vorgestellt. Im Kapitel „Managementabläufe“ findet sich eine detaillierte Iterationsplanung inkl. Meilensteinen des Projekts. Im Abschnitt „Risikomanagement“ werden die Projektrisiken sowie deren Vermeidungsstrategien aufgezeigt. Anschliessend folgt eine Auflistung der geplanten Arbeitspakete und der technischen Infrastruktur. Zum Schluss werden die Qualitätsmassnahmen beschrieben, welche die Höchstmögliche Softwarequalität sicherstellen wird.

14 PROJEKTÜBERSICHT

14.1 ORGANISATIONSSTRUKTUR

14.1.1 BACHELORARBEIT

HSR	Patrick Dietschweiler, Projektbetreuer	Alexander Klee, Diplomand
	Jürg Jucker Vize-Projektbetreuer	Sascha Bauer, Diplomand
Noser	Martin Straumann, Stv Head Business Unit	Hans Peter Bornhauser, Dipl. Ing ETH / STV
		Oliver Voll, Software Developer
Büchi AG		Claus Elsner, Head Software & NIR Developer
		Ruedi Hartmann, Product Manager
		Jürgen Traunig, Development Chemist
		Martin Uhland, Software Developer

Während der Arbeit wurde uns ein TFS-Server für die Source Kontrolle bereitgestellt und ein Sharepoint-Server für eine gemeinsame Einsicht der Dokumente.

14.1.2 BÜCHI AG

Die Firma hat Vertretungen weltweit. Das in der Bachelorarbeit behandelte Kjeldahl Laborgerät ist ein wichtiges Absatzprodukt und wird weltweit verkauft. Das bedeutet, dass auch berücksichtigt werden muss unter welchen Gegebenheiten die Servicetechniker arbeiten können. Es muss eine Lösung gefunden werden, die in allen möglichst allen Bedingungen z.B. langsame oder evtl. gar keine ständige Internetanbindung gerecht wird.

INTERNATIONALE VERTRETUNGEN

Abbildung 40 Internationale Vertretungen von Büchi

14.2 PERSONEN

14.2.1 PROJEKTTEAM

Alexander Klee aklee@hsr.ch	**Sascha Bauer** sbauer@hsr.ch
	
Student HSR Software Engineer, Bank Gutenberg, Zürich	Student HSR Software Engineer, IT Business Applications HSR, Rapperswil

14.2.2 BEWERTUNG UND BETREUUNG

Patrick Dietschwiler (pdietsch@hsr.ch)	Verantwortlicher / BetreuerProjektmanager, INS Rapperswil
Stefan Zettel (szettel@ascentive.ch)	Experte, Ascentive AG, Zürich
Marcel Reich Ranicki	Gegenleser

14.2.3 NOSER ENGINEERING

Person	Funktion	Kontakt
Martin Straumann Stv. Business Unit Lead Microsoft Solutions	Projektleiter Noser	martin.straumann@noser.com
Oliver Voll Software-Entwicklungs-Ingenieur	Technischer SPC	Oliver.voll@noser.com

14.2.4 BÜCHI LABORTECHNIK

Person	Funktion	Kontakt
Claus Elsner Head Software & NIR Development	Verantwortlicher Büchi	Elsner.c@buchi.com
Ruedi Hartmann, Ph. D. Business Area Manager Kjeldahl & Extraction	Product Manager	Hartmann.r@buchi.com
Martin Uhland Project Manager .Net	Project Manager	Uhland.m@buchi.com

14.3 PROBLEMSTELLUNG

Die Firma Büchi hat für den Support ihrer Kjeldahl Messgeräte in der ganzen Welt Servicetechniker, welche die Geräte warten. Um die Servicetechniker in ihrer Arbeit zu unterstützen, bietet Büchi eine PC-Software an, die eingesetzt wird um die Funktionalität der Geräte auf eine standardisierte Art und Weise zu testen.

Da die Laborgeräte von Büchi auch in entlegeneren Gebieten eingesetzt werden, stehen den Servicetechnikern nicht immer PCs oder Laptops zur Verfügung um die notwendigen Tests durchzuführen. Um diesem Problem entgegenzuwirken, soll eine mobile Applikation entwickelt werden, womit man die Gerätetests durchführen kann.

14.4 BESTEHENDE DOKUMENTATIONEN

Folgende Dokumente standen uns von Seite Büchi AG zur Verfügung:

Dokument	V	Datum	Beschreib
Software Design Framework	1.1	11.01.2011	Beschreibt die Architektur des Frameworks und liefert detaillierte Informationen zu den Komponenten.
SW Lastenheft Framework	1.5	21.01.2011	Hier wurden offene Punkte zur Bearbeitung in Bezug zum Test-Framework aufgelistet.
Device Software and PC Test Operator Manual	1.1	08.04.2011	Beschreibt die Anwendung der Gerätespezifischen Applikations-Schicht und deren Spezialitäten
PC Test Framework Application Manual	1.1	08.04.2011	Beschreibt die Verwendung, Funktionen und Einsatzmöglichkeiten des Test-Frameworks
UML-Design Framework	1.0	04.02.2011	Die meisten Teile des Programmcodes wurden in Form eines Enterprise Architect Projektes dokumentiert. Hier befinden sich einige aufschlussreiche Class-Diagrams.

14.5 PROJEKTZIELE

Projektziele helfen beim Abgrenzen des Projektumfangs. Zusammen mit der Feature-Liste - im folgenden Kapitel - kann der Scope bereits zu Beginn der Arbeit relativ genau bestimmt werden.

14.5.1 PROOF OF CONCEPT

Das Projekt ist eine Machbarkeitsstudie. Wir liefern keine marktreife Softwarelösung für Servicetechniker, sondern analysieren die Möglichkeiten und „best practices“ einer mobilen Applikation im Umfeld der Büchi Labortechnik.

Ziel	Beschreibung
Z01: Mobile App	Beweisen, dass eine mobile Applikation den Funktionsumfang der bestehenden Desktop-Test-Applikation abdecken kann.
Z02: Gerätetests	Gerätetests sollen mit dem Mobiltelefon durchgeführt werden können.
Z03: Useability	Design-Guidelines erarbeiten die eine gute Useability garantieren.

14.5.2 VERBINDUNGEN

Ziel	Beschreibung
Z04: Verbindungen	Applikationsunabhängige Verbindungsschnittstelle mittels Sockets definieren.
Z05: Cloud (optional)	Welche SWOTs hat die Cloud?

14.5.3 SICHERHEIT

Sicherheit ist in modernen Softwaresystemen immer ein Thema. Da die Implementierung, besonders bei dezentralen Systemen, enorm komplex wird und aufgrund der zur Verfügung stehenden Zeit, wurde beschlossen **keine Security**-Mechanismen zu implementieren. Wir entwickeln unsere Software ausschliesslich für Servicetechniker die Geräte warten welche für die Nährstoffanalyse von Lebensmitteln eingesetzt werden. In diesem Bereich werden keine Anforderungen an die Sicherheit der Daten gestellt.

Ziel	Beschreibung
Z06: Security allgemein	Sicherheitskonzepte wurden angedacht und ein geeignetes Sicherheitsmodell für spätere Projekte im „Non-Food-Bereich“ vorgeschlagen.

14.6 FEATURE LISTE

Feature	Beschreib	Art
F01: Geräte (K375) im WLAN finden und anzeigen	Die mobile Applikation soll in einem bekannten WLAN alle Kjeldahl-Messgeräte finden und anzeigen können.	Required
F02: Mit K375 verbinden	Die App soll mit Hilfe von Sockets auf ein einzelnes Gerät verbinden können.	Required
F03: Verfügbare Tests für K375 anzeigen. (inkl. Testschritte)	Die App soll die Testarten (End Tests, Service Test) des Geräts anzeigen und in einer Detailansicht die verschiedenen Testschritte einer einzelnen Testart anzeigen können.	Required
F04: Test ausführen und Testresultate eingeben	Die gewünschte Testart soll aus der App gestartet und die Resultate der einzelnen Testschritte eingegeben werden können.	Required
F05: Testresultate archivieren und ansehen	Die eingegebenen Testresultate sollen auf dem mobilen Gerät gespeichert damit diese in einem späteren Zeitpunkt wieder betrachtet werden können.	Required
F06: Job Settings	Qualitätswerte zur Validierung korrekter Funktion des Gerätes sollen	Optional

	über das UI editierbar sein.	
F07: Testreport generieren	Aus den archivierten Testresultaten soll ein Testreport generiert werden können.	Optional
F08: Testresultate synchronisieren	Die Testresultate sollen, in geeigneter Form, mit einem PC synchronisiert werden können.	Optional
F09: Testresultate an Cloud senden	Die Testresultate sollen in die Cloud übertragen werden können.	Optional
F10: Mehrsprachigkeit	Die Sprache des UI soll in den Einstellungen der App gewechselt werden können.	Optional

15 MANAGEMENTABLÄUFE

15.1 AUFWANDSCHÄTZUNG

Die Bachelorarbeit (12 ECTS) dauert vom 19.09.2011 bis 23.12.2011. Der erwartete Aufwand pro Teammitglied liegt bei 360 Arbeitsstunden, was einem **Tagesdurchschnitt von knapp 4 Stunden während 95 Tagen** entspricht. Sollte es Probleme bei der Umsetzung der Ziele geben, werden die Features, nach Absprache mit dem Auftraggeber, gekürzt.

15.2 ITERATIONEN UND MEILENSTEINE

Iteration	Endet am	Resultat
Inception 1	Sonntag, 25.09.2011	**Projektplanung:** • Die Anforderungen an die neue Software wurden mit dem Auftraggeber abgeklärt und ein Architekturentwurf ausgearbeitet. • Zeitplan ist erstellt • Entwicklungsumgebung auf Arbeitsrechnern installiert
Elaboration 1 **MS1: Projektsetup**	Sonntag, 02.10.2011	**Projektplanung**: • Die Projektplanung wurde mit dem Betreuer verifiziert. **Projekt-Dokumentation**: • Dokumentvorlagen erstellt • Konfigurationsverwaltung eingerichtet • Domain Model begonnen • Analyse des Test-Frameworks begonnen • Sequenzdiagramme begonnen • Qualitätsmassnahmen definiert • Festlegung der Technologien/Architektur • Synergien zum Remote-App Team abgeklärt

		Testumgebung: • Der Simulator läuft auf den Arbeitsrechnern **Architekturprototyp:** • Projekttemplate eingerichtet
Elaboration 2 **MS2: Architectural Prototype**	Sonntag, 16.10.2011	**Projekt-Dokumentation**: • Domainmodel abgeschlossen • Klassendiagramm abgeschlossen • Sequenzdiagramme abgeschlossen **Architekturprototyp:** • Ein erster lauffähiger Architekturprototyp wurde auf dem Projekt-Phone installiert. • Es ist klar welche Teile des bestehenden Frameworks mit Mango kompatibel sind bzw. welche neu geschrieben werden müssen. • Synergien zum „Remote-App Team“ werden genutzt. **Prototyptests:** • UnitTests untersuchen das Verhalten des Architekturprototyps.
Construction 1 **MS3: Technische Risiken beseitigt**	Dienstag, 25.10.2011	**Software Architektur Dokument:** • Die letzten Korrekturen am SAD sind im Prototyp umgesetzt. Beta-Phase der Software beginnt. **Connection Tests:** • Das korrekte Verhalten der Verbindungsklassen wird mit UnitTests überprüft und sichergestellt.
Construction 2 **MS4: Functional Prototypes**	Sonntag, 13.11.2011	**UI-Prototyp:** • Ein Prototyp, der die vorgesehenen UI-Komponenten enthält und deren Funktionalität demonstriert, läuft auf dem BA-Phone. **Verbindungs-Prototyp:** • Die Kernkomponenten des Verbindungsaufbaus sind implementiert und kommunizieren durch alle Schichten hindurch mit dem Simulator. • Das Ausführen von Tests auf dem Simulator funktioniert.
Construction 3 **MS5: Release Candidate**	Freitag, 9.12.2011	**Release Candidate:** Der Release Candidate wird beim Kunden getestet.
Transition 1 **MS6: Produktübergabe und Präsentation**	*Freitag, 16.12.2011*	**Release Candidate:** Der Release Candidate wird beim Kunden übergeben und erfüllt alle Anforderungen. **Präsentation:** Die Präsentation wurde vor den Geschäftsleitungen der beiden Projektpartnern (Noser/ Büchi) gehalten.
Transition 2 - Reserve	Freitag,	**Abgabe:** Die Dokumentation der Semesterarbeit wird dem Betreuer

MS7: Abgabe	23.12.2011	übergeben. **BuchiTec mobile (Projektname):** Mit dem Abschluss des Projekts, wird dem Auftraggeber das Projekt Repository übergeben.

15.3 ARBEITSPAKETE

Siehe „doc/3_PmQm/Projektcockpit.xlsx“

15.4 RISIKOMANAGEMENT

KM Kosten der Massnahmen (h)
S_max Maximaler Schaden (h)
S_gem Gewichteter Schaden (h)
P Eintrittswahrscheinlichkeit

Id	Risiko	Auswirkung	Massnahme	KM	S_max	P	S_gew	Prio
R01	Portierung des Codes des bestehenden Test-Framework gelingt nicht oder nur teilweise	Teile oder die gesamte Testumgebung müssen neu geschrieben werden. Das Projekt verzögert sich. Eventuell können sogar Features nicht zeitgemäss umgesetzt werden	Prototype schreiben der einen Durchstich durch alle Architekturschichten darstellt. Anschliessend mit den Funktionalitäten des Frameworks erweitern. So schnell wie möglich den Code analysieren, feststellen welche Bereiche wiederverwendet werden können. Anhand von dieser Liste bestimmen, welche Features umgesetzt werden	32	240	20%	48	High
R02	Phone-Emulatoren bzw. die K375-Simulatoren verhalten sich anders als die echten Geräte.	Die Applikation verhält sich im realen Umfeld anders als im simulierten Umfeld.	Prototype schreiben der die „required-Features" abdeckt und im realen Umfeld getestet wird.	64	120	40%	48	High
R03	Projektziele werden von den verschiedenen Projektpartnern unterschiedlich interpretiert	Die Partner sind mit der Umsetzung des Projekts nicht zufrieden.	Projektplan, Features und Requirements von den beteiligten Stellen absegnen lassen.	12	120	30%	36	Medium
R06	Die Treiber zur Übertragung von Steuerbefehlen sind nicht portabel	Das Projekt gerät in Zeitverzug. Eventuell scheitern sogar Required Features	Möglichst schnell den Source Code der Treiber anfordern, analysieren und Berichten in wie weit ein Mehraufwand für dessen Implementierung entsteht	8	64	25%	26	Medium
R05	Synergien zwischen den Projektteams können aus hardwaretechnischen Gründen nicht genutzt werden.	Wir programmieren dieselbe Funktionalität wie das zweite Projektteam.	Gezielte Absprache mit dem anderen Team besonders im Bereich der Kommunikation über Sockets und des Scannens nach vorhandenen Geräten treffen.	4	48	50%	24	Medium
R04	Komplexität der eingesetzten Technologien und der Businesslogik wird unterschätzt.	Das Projekt gerät in Zeitverzug.	Projekt nach Pflicht-Zielen und optionalen Zielen ausrichten, dass bei unerwartet hohem Aufwand optionale Ziele gestrichen werden können	2	64	35%	22.4	Medium

16 INFRASTRUKTUR

16.1 ENTWICKLUNGSUMGEBUNG

Was	Womit
Mobile App	Visual Studio 2010, Windows Phone 7 Application (Silverlight)
Cloud	Windows Azure
Versionisierung	SVN (HSR intern) und Team Foundation Server (extern)
Dokumentation	Dropbox 1.1.45 (HSR intern) und Sharepoint (extern)
Silverlight Toolkits	- http://mytoolkit.codeplex.com/ - http://silverlight.codeplex.com/releases/view/71550 - … -

16.2 REQUIREMENTS / ISSUE TRACKING

Requirements und Issues fliessen im Verlaufe des Projektes in unser Zeitmanagement Instrument, die Exceldatei Projektcockpit.xlsx, ein. Dort werden alle Arbeitspakete erfasst und der Aufwand geschätzt. So hat man eine stetige Übersicht über den Projektstatus und die Planungseffizienz. Ausserdem dient dieses Dokument dazu einzelne Zeiteinträge pro Person und Arbeitspaket zu erstellen.

16.3 KOMMUNIKATION

Bevorzugte Kommunikation ist der direkte Dialog. Weiter benützen wir für Studenten gängige Kommunikationsmittel wie E-Mail, Skype, Facebook, Mobiltelefone und das ProjektWiki.

16.4 VERSIONIERUNG / BACKUP

Das Versionenmanagement wir über den Team Foundation Server der Noser Engineering sichergestellt. Zusätzlich liegt der gesamte Projektstatus der letzten Iteration in der Dropbox, welche ebenfalls über ein Versionierungfeature verfügt. Lokal werden grundsätzlich keine Daten gehalten. Somit ist ein komplettes Backup unseres Projektes sichergestellt.

17 QUALITÄTSMASSNAHMEN

17.1 DOKUMENTATION

Unsere Dokumentation wird von jedem Teilnehmer nachgeführt und stimmt immer mit der aktuellen Situation überein. In erster Linie gilt die Dokumentation, dann der Code.

Bei komplexen Codefragmenten werden Kommentare angebracht. Alle Klassenschnittstellen werden mit „triple slash“ im Code dokumentiert.

17.2 SITZUNGSWESEN

Im Sitzungswesen haben sich folgende Dokumente bewährt.

Dokument	Zweck
Traktanden	Dienen als Wegweiser in der Sitzung und sind ein gutes Instrument zur pragmatischen Sitzungsführung / -Vorbereitung
Protokoll	Während den Sitzungen werden die wichtigsten Punkte aufgeschrieben. Daraus lassen sich später die Beschlüsse ableiten.
Beschlüsse	Die Beschlussliste, wenn nötig, kann aus den Traktanden und dem Sitzungsprotokoll extrahiert werden. Die Beschlüsse werden in neue bzw. vorhandene Arbeitspakete integriert.

Sitzungen mit dem Projektbetreuer finden im Wochenrhythmus statt. Abgesehen von den Codereviews, die zu vordefinierten Zeiten stattfinden, werden Sitzungen mit Noser und Büchi individuell, je nach Projektstatus vereinbart.

Die Sitzungsprotokolle werden dem Betreuer per Dropbox - den Projektpartnern via Sharepoint - zur Verfügung gestellt.

17.3 PROJEKT- UND ZEITPLAN AKTUALISIEREN

Bis Sonntagabend müssen jeweils alle geleisteten Arbeitszeiten eingetragen sein.

17.4 TODO-LISTEN

Im Projektcockpit.xlsx kann immer herausgelesen werden, was es zu tun gibt.

17.5 KNOW-HOW SHARING

Technologie –Wissen soll stets auf dem Projektwiki platziert werden, damit sich Auftraggeber und Projektmitarbeiter stets informieren können.

17.6 REVIEWS

17.6.1 DOKUMENTREVIEW

Nach Abschluss jeder Iteration findet ein Dokumentreview statt um sämtliche Dokumente auf einen sauberen Stand zu bringen. Anschliessend wird dies in der Dokument-Änderungsgeschichte eingetragen.

17.6.2 CODEREVIEWS

Geschehen zu vordefinierten Zeiten. Es sind zwei Codereviews, unter Leitung der Noser Engineering, eingeplant. Dabei wird der Code des Projektteams von den technischen Verantwortlichen analysiert. Das Feedback aus den Review-Sitzungen wird so schnell wie möglich umgesetzt. Die aus dem Feedback resultierenden Änderungen werden, in Form eines kurzen Berichts, an die Verantwortlichen der Noser übergeben.

17.7 TESTS

17.7.1 AUSSCHLUSSKLAUSEL

Siehe §9 der Aufgabenstellung.

17.7.2 USABILITY- UND CONNECTION-TESTS

Nachdem in der Construction 2 der RC1 veröffentlicht wurde, werden Usability- und Connection-Tests bei Büchi durchgeführt. Diese Tests geben uns wertvolles Feedback in Bezug auf das GUI-Design und auf unsere Socket Implementierung.

Kjeldahl Servicetechniker Mobile App

Machbarkeitsstudie

Anforderungsspezifikation

18 EINFÜHRUNG

18.1 ZWECK

Die Anforderungsspezifikation dient den Anspruchsgruppen des Projekts dazu, die Nachfolgend werden die Features und der Umfang der zu entwickelnden Applikation genau definiert. Die Anforderungen werden individuell im Gespräch mit den Partnern aus der Büchi AG ausgehandelt. Dieses Dokument hält Erfahrungen fest, die während Gesprächen entstanden sind.

18.2 DOKUMENTGRUNDLAGE

Die Anforderungsspezifikation wurde auf Grund des *Volere Requirements Specification Template* erstellt. (Robertson & Robertson, 2010)

19 ZIEL UND ZWECK DES PROJEKTS

Die Mobile App soll zu Demonstrationszwecken beim Verkaufsstart des Kjeldahl-375-Messgeräts der Firma Büchi Labortechnik zu Verfügung stehen. Ziel ist es den Kunden einerseits zu zeigen das Büchi im Bereich der modernen Technologien Forschung betreibt, andererseits soll die Applikation als Verkaufsargument dienen. Für weitere Informationen zur Problemstellung siehe Kapitel *Projektübersicht - Problemstellung* in *doc/03_PmQm/Planung/Projektplan.docx*

20 ANSPRUCHSGRUPPEN

Siehe Kapitel *Projektübersicht - Organisationsstruktur* in *doc/03_PmQm/Projektplan.docx*

REVIEW. Genaue Ansprüche der Stakeholder hier definieren nachdem diese mit den Teilnehmern genau abgeklärt wurden!

Anspruchsgruppe	Anspruch an das Projekt
Büchi Labortechnik	Benötigt einen Prototypen zur Vorführung der Geschäftsleitung. Machbarkeit der Funktionalität muss bewiesen werden. Useability, Design und Spezifikation müssen abgesprochen werden. Ist Kunde von Noser Engineering AG
Noser Engineering	Technologischer Lead und
HSR Institute for Networked Solutions (INS)	Innovation und Aufbau an Know How. Erwartet eine saubere Dokumentation in Form eines Technischen Berichts.
Servicetechniker	Benutzer der Applikation. Wichtiger Partner beim definieren der Anwendungsfälle und für Feedback des UI-Designs

21 PROJEKTUMFANG (SCOPE)

- Proof of Concept

22 ANFORDERUNGEN

22.1 FEATURES

Die folgende Tabelle zeigt alle Required-Features der Applikation. Daraus werden in den folgenden Kapiteln die funktionalen und nicht-funktionalen Anforderungen abgeleitet. Die ganze Tabelle, inklusive optionalen Features, befindet sich im Projektplan.

Feature	Beschreib	Art
F01: Geräte (K375) im WLAN finden und anzeigen	Die mobile Applikation soll in einem bekannten WLAN alle Kjeldahl-Messgeräte finden und anzeigen können.	Required
F02: Mit K375 verbinden	Die App soll mit Hilfe von Sockets auf ein einzelnes Gerät verbinden können.	Required
F03: Verfügbare Tests für K375 anzeigen. (inkl. Testschritte)	Die App soll die verschiedene Testarten (End-Tests, IQOQ-Tests, etc.) des Geräts anzeigen (TestView) und in einer Detailansicht die verschiedenen Jobs der gewählten Testart anzeigen können (DetailView).	Required
F04: Test ausführen und Testresultate eingeben	Die gewünschte Testart soll aus der App gestartet und die Resultate der einzelnen Testschritte eingegeben werden können.	Required
F05: Testresultate archivieren und ansehen	Die eingegebenen Testresultate sollen auf dem mobilen Gerät gespeichert und zu einem späteren Zeitpunkt wieder angeschaut werden können.	Required
F07: Testresultate synchronisieren	Die Testresultate sollen, in geeigneter Form, mit einem PC synchronisiert werden können.	Required

22.2 FUNKTIONALE ANFORDERUNGEN

A# Anforderung Nr. #
F# Abgeleitet von Feature Nr. #

A#	F#	Beschreib
FA01	F01	In einem ungesicherten WLAN sollen alle K375-Geräte im selben Subnetz gefunden werden. Die Geräte werden mittels Broadcast gesucht.
FA02	F01	Nur K375-Messgeräte sollen zum Verbindungsaufbau vorgeschlagen werden.
FA03	F02	Es kann nur zu einem Gerät gleichzeitig eine Verbindung bestehen. Erst wenn diese Verbindung geschlossen wurde, kann auf ein neues Gerät verbunden werden.
FA04	F02	Die Verbindung wird mittels TCP-Socket aufrechterhalten. Es sind die Socket-Klassen aus dem

		System.Net.Sockets-Namespace der *.Net Library for Silverlight* zu verwenden. (Microsoft, 2011)
FA05	F03	Alle verfügbaren Tests sind Bestandteil der Applikation. Zur Laufzeit können keine Tests - von externer Quelle geladen - manipuliert oder - neu definiert werden.
FA06	F03	Bei Auswahl der Testart, werden die dazugehörigen Jobs der Reihe nach in der JobView angezeigt.
FA07	F04	Es ist möglich einen einzelnen Test aus der Auflistung der Jobs zu starten.
FA08	F04	Es ist möglich eine Sequenz (Alle Tests einer Testart) zu starten. Diese werden einzeln, in der angezeigten Reihenfolge durchgeführt.
FA09	F04	Während eines Testlaufs wird auf dem Display nur der aktuell laufende Test angezeigt. Diese Ansicht kann nur verlassen werden durch: - Stoppen des Tests - Erfolgreiches Abschliessen des Tests
FA10	F04	Ein Test kann pausiert werden. Das Verhalten des Pausierungsbefehls ist je nach Job in dessen bestehenden Implementation definiert.
FA11	F04	Zu jedem Job kann ein modaler Informationstext angezeigt werden. Um mit der Bedienung fortzufahren muss dieser Informationstext vom Benutzer mit „OK“ akzeptiert worden sein.
FA12	F04	Um einen Test erfolgreich abzuschliessen muss ein gültiges Testresultat eingegeben oder ermittelt worden sein.
FA13	F04	Bei automatisch ermittelten Testresultaten (z.B. beim Software-Versions-Test) wird sofort der nächste Job geladen und ausgeführt.
FA13	F04	**REVIEW! Macht das Sinn? Wie kann das Feedback bei einem automatischen Job angezeigt werden? Was nützt das Feedback, wenn sowieso auf den nächsten Job gewechselt wird?** Der User erhält ein visuelles Feedback sobald ein Job abgeschlossen wurde. Dieses wird in Form eines grün oder rot eingefärbten Indikators auf dem Display angezeigt.
FA14	F05	Die Testresultate werden in Form einer XML-Datei in der Speicherstruktur der mobilen Device abgelegt. Es ist die IsolatedStorageFile-Klasse aus dem *System.IO.IsolatedStorage-Namespace* der *.Net Library for Silverlight* zu verwenden. (Microsoft, 2011)
FA15	F05	Nach einer abgeschlossenen Testsequenz wird eine Übersicht angezeigt, die alle abgeschlossenen Jobs - je nach deren Erfolg - grün oder rot eingefärbt darstellt. Klickt man auf einen eingefärbten Namen, gelangt man zur Detailansicht des Tests. Diese Übersicht wird aus dem XML-File erstellt und kann somit auch zu einem späteren Zeitpunkt wiederhergestellt werden.
FA16	F07	Wenn das Gerät mit einem PC verbunden wird, sollen die Testresultate auf den PC geladen werden können.

22.3 NICHT FUNKTIONALE ANFORDERUNGEN

A#	Typ	Beschreib

NA01	Look and Feel	Es sollen die Metro-Style Guidelines von
NA02	Bedienbarkeit	So einfach wie möglich. So nah an den PC-Software-Abläufen wie möglich
NA03	Performance	-
NA04	Umgebung	Ungesichertes oder bekanntes WLAN wird vorausgesetzt.
NA05	Support	-
NA06	Security	Keine Sicherheitsmechanismen.
NA07	Sprache	Mehrsprachigkeit unterstützt.
NA08	Standards	-

Kjeldahl Servicetechniker Mobile App

Machbarkeitsstudie

Software Architectural Document

23 DEVICE DISCOVERY ANALYSE

Das nachfolgende Kapitel 23.1 wurde von der Gruppe Schäpper / Schneider verfasst.

Ein Feature der bestehenden Applikationen für den PC ist es, alle vorhandenen Büchi-Geräte im aktuellen Netzwerk mittels Broadcast zu finden. Die Applikation setzt einen Broadcast ab und wartet auf die Antworten der verfügbaren Geräte. Aus den Antwort-Paketen kann die IP der Geräte ausgelesen und im Programm gespeichert werden. Ausserdem ist es nicht im Interesse von Büchi AG eine Lösung anzustreben, die einen Server benötigt.

23.1 BROADCAST

Damit die Büchi Geräte gefunden werden können, wird auf die Antwort eines Broadcasts gewartet. Dieser hat die Form „Text=Buchi DeviceScan;HostIP=10.10.140.96;HostPort=1204;“

Die Informationen "HostIP=" und "HostPort=" können auch aus dem Socket gewonnen werden. Sie sind aus Gründen der einfachen Implementation doppelt vorhanden. (Büchi EthernetSpez.pdf)

Die Applikation wartet auf dem angegebenen UDP Port auf die Geräteantworten. Nachfolgende Kommunikation erfolgt über TCP und fällt nicht unter die Problematik dieses Dokuments.

23.1.1 PROBLEME

VERSENDEN VON BROADCASTS

Windows Phone unterstützt das Versenden von Broadcasts nicht[2]. Wird jedoch der Socket.ConnectAsync() Methode bereits ein Buffer mitgegeben, wird ein Broadcast versendet. Dieser wird von den Büchi Geräten empfangen und beantwortet.

```
// Add the data to be sent into the buffer
var data = string.Format("Text=Buchi DeviceScan;HostIP={0};HostPort={1};", myIp, answerPort);
byte[] payload = Encoding.UTF8.GetBytes(data);
socketEventArg.SetBuffer(payload, 0, payload.Length);
// Make an asynchronous Connect request over the socket
socket.ConnectAsync(socketEventArg);
```

EMPFANGEN AUF EINEM PORT

Für den Empfang von UDP Paketen ist es nötig, dass ein Socket auf den Empfangsport gebunden wurde, und der Socket empfangsbereit ist (Bind und Listen Funktionalität). Befindet sich kein Socket für den entsprechenden Port in diesem Zustand resultiert dies in einer ICMP Port Unreachable Meldung.

Die Windows Phone Sockets API stellt keine Methoden für Bind und Listen zur Verfügung. Daraus geht hervor, dass ein Windows Phone kein UDP Server sein kann, der auf eingehende Verbindungen wartet. Es können nur Antworten auf zuvor gesendetes empfangen werden.

Die Listen Funktionalität wird benötigt, um die Antworten von den Büchi Geräten zu empfangen. Nachdem der Broadcast gesendet wurde, antworten die Clients auf die im gesendeten String angegebene IP und Port Nummer.

Ein Lösungsansatz wäre, zuerst etwas auf der Verbindung zu versenden, danach kann empfangen werden.

[2] MSDN ,http://msdn.microsoft.com/en-us/library/hh202874(v=vs.92).aspx

Verschiedene Varianten wurden mit Prototypen getestet:

- Zuerst die Connect Methode aufrufen, oder zuerst etwas senden dann die Empfangsmethode auf dem Socket aufrufen. Da jedoch keine Verbindung zu einem Endpunkt besteht (es müssen alle Clients akzeptiert werden), kann nichts versendet werden und es kann auch kein Verbindungsaufbau durchgeführt werden.
- Denselben Socket für das Versenden der Broadcasts wie für das Empfangen verwenden. Dann wäre die Problematik des zuerst etwas senden müssen behoben. Dabei entstehen jedoch weitere Probleme:
    - Der Socket kann nicht auf einen lokalen Port gebunden werden.
    - Die verwendete Portnummer kann nicht ausgelesen werden. Dies wäre aber nötig, damit er im Broadcast String mitgegeben werden kann.

23.1.2 FAZIT

Es können zwar Broadcasts versendet werden, diese werden auch von den Büchi Geräten empfangen und beantwortet, es ist jedoch nicht möglich die Antworten abzuholen.

Daraus entstand die alternative Idee, die Problematik mit IP Multicast zu behandeln.

23.2 MULTICAST

Mittels Multicast sollen alle vorhandenen Kjeldahl-Messgeräte im WLAN gefunden werden. Dafür braucht es sowohl auf dem Windows Phone als auch auf den Büchi-Geräten einen Multicast-Socket, der auf eine bestimmte Group Address registriert wurde.

23.2.1 TECHNISCHE DETAILS

Um eine Multicast Gruppe zu erstellen, (bzw. sich darauf anzumelden)wird das Protokoll Internet Group Management Protocol (IGMP) verwendet.

Für die Kommunikation im Netzwerk wird eine Multicast Adresse (IP) benötigt. Dies Klasse-D-Netzadresse wird nie in einem lokalen Netz vergeben, da man nicht will dass ein Host eine Multicast Addresse belegt.

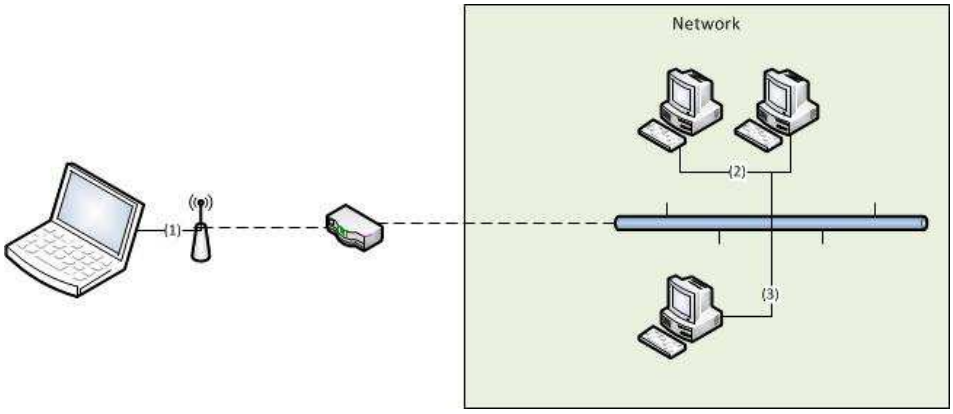


Abbildung 41 Aktoren der Multicast-Lösung

Da sich diese Multicastadresse nicht im lokalen Netz befindet, wird ein Paketdas an diese IP addressiert ist zuerst an den Gateway (Router) geleitet. Dieser erkennt dass es sich um eine Multicast Adresse handelt und eröffnet eine Multicast Group.

(1) Der erste Client öffnet eine Verbindung auf eine Multicast Adresse.Die Multicast Group wird erstellt(IGMP)
(2) (3) -> Die Clients öffnen eine Verbindung auf dieselbe IP Adresse und werden ebenfalls in die Multicast Gruppe aufgenommen.

Eine Multicast Infrastruktur ist geschaffen. Nun können jegliche Art von UDP Paketen versendet werden. Wenn nun Daten von einem Client an die Multicast-IP versendet werden, nimmt der Router das Paket entgegen, multipliziert dieses und sendet eine Kopie an jeden Teilnehmer der Gruppe. Die Bandbreite beim Sender ist somit gleich gross als würde er das Paket nur an einen Teilnehmer senden.

23.2.2 PROBLEM

Die Verwendung von Multicast erlaubt es allen Clients, sich für eine solche Gruppe zu registrieren. Auch unbefugteClients können sich auf diese Adresse registrieren. Ausserdem kann auch nicht mehr portspezifisch auf der Firewall gefiltert werden, da der Verkehr ausschliesslich über den, in der Multicast-Infrastruktur vorgesehen Port, abläuft.

- Es ist möglich jegliche Art von Daten zu versenden und empfangen.
    - Die interne Struktur eines Netzwerks kann so ausspioniert werden.

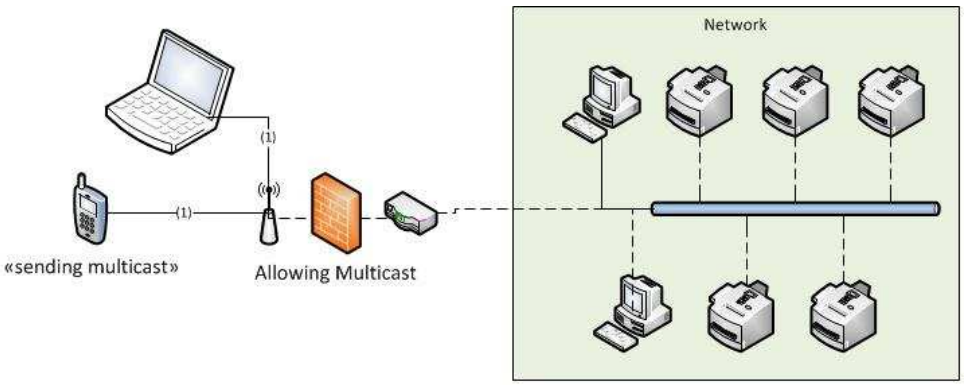


Abbildung 42 Firewall lässt Multicast zu

23.2.3 FAZIT

IGMP Pakete werden von den meisten Infrastrukturen aus den oben genannten Sicherheitsgründen geblockt.

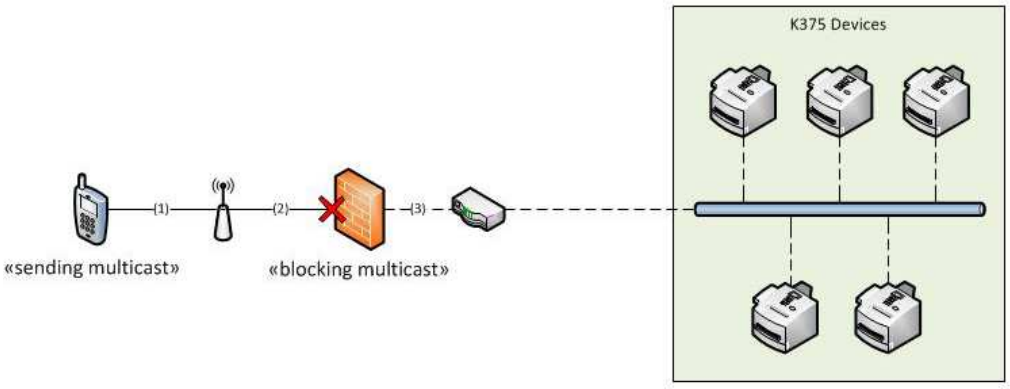


Abbildung 43 Firewall blockt Multicast

23.2.4 ERFAHRUNGEN BEI NOSER

Die Multicast-Lösung wurde in einer gesicherten Netzwerk-Umgebung getestet. Es tauchten die erwarteten Probleme auf:

- Auf „localhost“ konnte die Testsoftware eine Multicast-Gruppe erstellen, in die Gruppe joinen, einen Listening-Socket öffnen und auf die Multicastnachrichten warten. Die gesendeten Nachrichten kamen an.
- Sobald jedoch ins Guest-Netzwerk bei Noser gewechselt wurde, funktionierte die Test-Software nicht mehr wie gewünscht. Sowohl PING als auch Traceroute zeigte, dass die von uns gesendeten Nachrichten nicht über die Firewall hinweg kamen.

Dies lag an dem sauber aufgesetzten Netzwerk bei Noser, welches die meisten Ports geschlossen hat und auf den offenen Ports vermutlich nur ausgewählte Protokolle zulässt. ICMP (PING und Traceroute) gehören nicht dazu und höchst wahrscheinlich auch IGMP (Multicasts) nicht.

23.2.5 VERSUCHSAUFBAU MIT EIGENEM ROUTER

INFRASTRUKTUR

Um einen Fehler der Testsoftware auszuschliessen wurde an der HSR ein lokales Netz aufgebaut und die identische Testsoftware wie bei Noser dazu verwendet, einen Multicast über den Router zu schicken. Der Aufbau bestand aus zwei Clients und einem ungesicherten Router (verwendeter Multicast Port: 11111, offen).

Abbildung 44 Versuchsaufbau des Multicast-Client Funktionalitätstest

RESULTAT

Unmittelbar nachdem der Mulitcast-Sender die Nachricht verschickt hat, erschien diese beim Empfänger. Somit konnte die Funktionsfähigkeit der Testsoftware bewiesen werden.

Multicast-Sender	Multicast-Empfänger

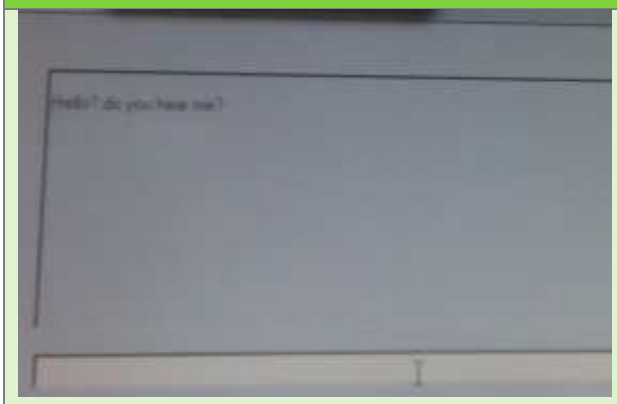
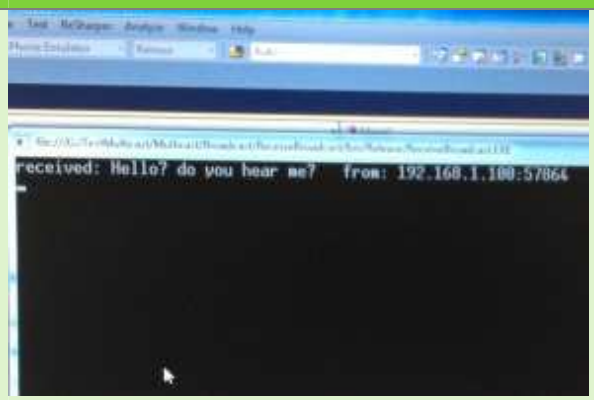

	

Abbildung 45 MC-Funktionstest Sender	Abbildung 46 MC-Funktionstest Receiver

23.3 WCF SERVICE

Um das Sicherheitsloch mit Multicasts zu vermeiden und dennoch keine Broadcasts einzusetzen, bietet sich als letzte Möglichkeit an, vom Phone aus einen WCF Service aufzurufen. Dieser kann uns die IPs aller verfügbaren Büchi-Geräte mitteilen. Da der WCF Service auf einem Server mit .Net 4.0 läuft, stehen auch die Broadcast-Sockets zur Verfügung. Somit können vom Server aus ohne Probleme per Broadcast alle K-375 Geräte im Netzwerk ausfindig gemacht werden. Es wurden im Gespräch mit Martin Straumann bereits folgende Anforderungen an die WCF-Lösung definiert:

- Der Client initiiert einen „discovery“ auf Serverseite durch Aufruf der Servicemethode DiscoverAsync.
- Hat der Server aber bereits aktuelle Daten (z.B. nicht älter als 30s), so werden dem Client diese übermittelt
    - Ansonsten werden Sie wieder gesammelt
- Falls gleichzeitig mehrere Anfragen an den Server gestellt werden, darf das Netzwerk nicht mit Broadcasts geflutet werden. Es muss ein „Aggregations-Mechanismus für Discovery Requests“ implementiert werden.

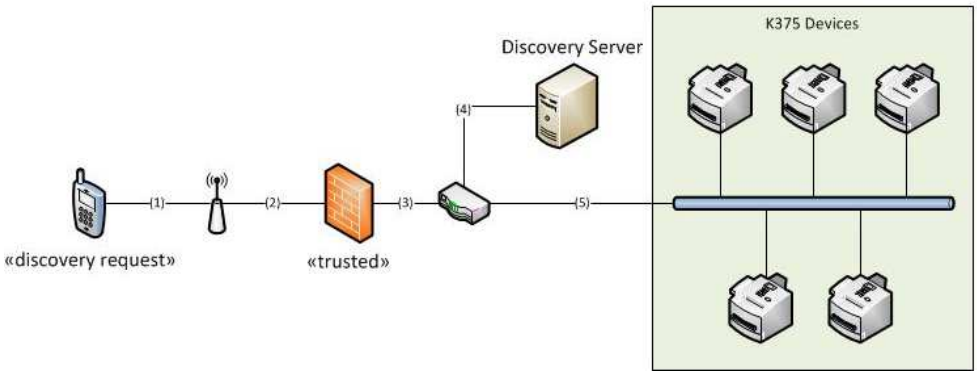


Abbildung 47 Discovery-Variante

Das nächste Kapitel wird auf die letzte Variante eingehen. Das Suchen via Master-Device.

23.4 MASTER DEVICE

- Ein Büchi-Gerät im Netzwerk ist ein Master Gerät, welches benutzt werden kann, um eine Liste aller Geräte im Netzwerk abzufragen.
- Diese Master Adresse ist hart codiert im Client hinterlegt.

Vorteile	Nachteile
Die Wahrscheinlichkeit dass der Master sich im selben logischen Netzwerk befindet wie die anderen Geräte ist sehr hoch. Somit gibt's **weniger Kommunikationshindernisse** wie Firewall oder Unterdrückung von Broadcast.	Zusätzliche Belastung für Büchi-Gerät
Es wird **keine weitere Komponente** benötigt.	Wenn MasterDevice ausgeschaltet ist, ist kein Discovery möglich

24 IST ANALYSE

Um das nicht-triviale Framework zu verstehen und es, mit so wenig Funktionalitätsverlust wie möglich, auf das WindowsPhone 7 zu bringen, wurde systematisch vorgegangen.

Zuerst wurden die sichtbaren Teile des Programms - das Verhalten des Frameworks bei Benutzereingaben – analysiert. Dabei lag der Fokus darauf diejenigen Komponenten zu identifizieren, die für Bedienung essentiell sind. Diese Elemente galt es dann in einem ersten UI-Prototyp zusammenzubringen und dem Kunden zu präsentieren.

Als zweiter Schritt wurde das .Net-Framework als Basis der Silverlight-Architektur definiert und versucht, so viel Code wie möglich in der Silverlight Umgebung des WP7 zum Laufen zu bringen. Sobald das Projekt gebuildet werden konnte, wurde ein Architekturprototyp erstellt der durch alle Layer hindurch mit einem Simulationsprozess kommunizieren konnte.

Zuletzt konnte dank des Prototyps die Analyse der Architektur des Buchi-Frameworks ebenfalls abgeschlossen werden.

Die folgenden Kapitel gehen im Detail auf die beschriebenen Analyse-Disziplinen ein.

24.1 USERINTERFACE

24.1.1 KOMPROMISS

Die Schwierigkeit besteht darin einen Kompromiss zwischen der vorgegebenen bestehenden Lösung und der von uns zu entwickelnden Mobilen Lösung zu finden. Auf dem Mobilen Gerät herrschen andere Umstände:

EINSCHRÄNKUNGEN

- Sehr kleiner Display
- Keine überlappenden Fenster
- Eingabemethode: Touch (rechtsklick kann nur mit längerer Berührung realisiert werden)
- Nur Silverlight Komponenten des .NET Frameworks verfügbar

ERWEITERUNGEN

- Hochformat und Querformat, dynamisches Rotieren
- Panoramaview
- Application Bar

Der wichtigste Punkt ist das im Vergleich zu einem Widescreen-Bildschirm sehr kleine Display des Windows Phone. Die Konsequenz ist, dass auf viele optische Features verzichtet werden muss und die Applikation ein dementsprechend schlankes Erscheinungsbild erhalten wird. Die Funktionalität im Vergleich zum PC-Test-Framework soll dabei möglichst erhalten bleiben.

24.1.2 NICHT BENÖTIGTE KOMPONENTEN

Nachfolgende Abbildung zeigt die MainControl der PC-Test-Software. Das UI macht einen aufgeräumten Eindruck, aber es wird sofort klar, dass dieses Design nicht auf einen 640x480 Screen passen wird. Aus diesem Grund werden einige Komponenten entfernt.

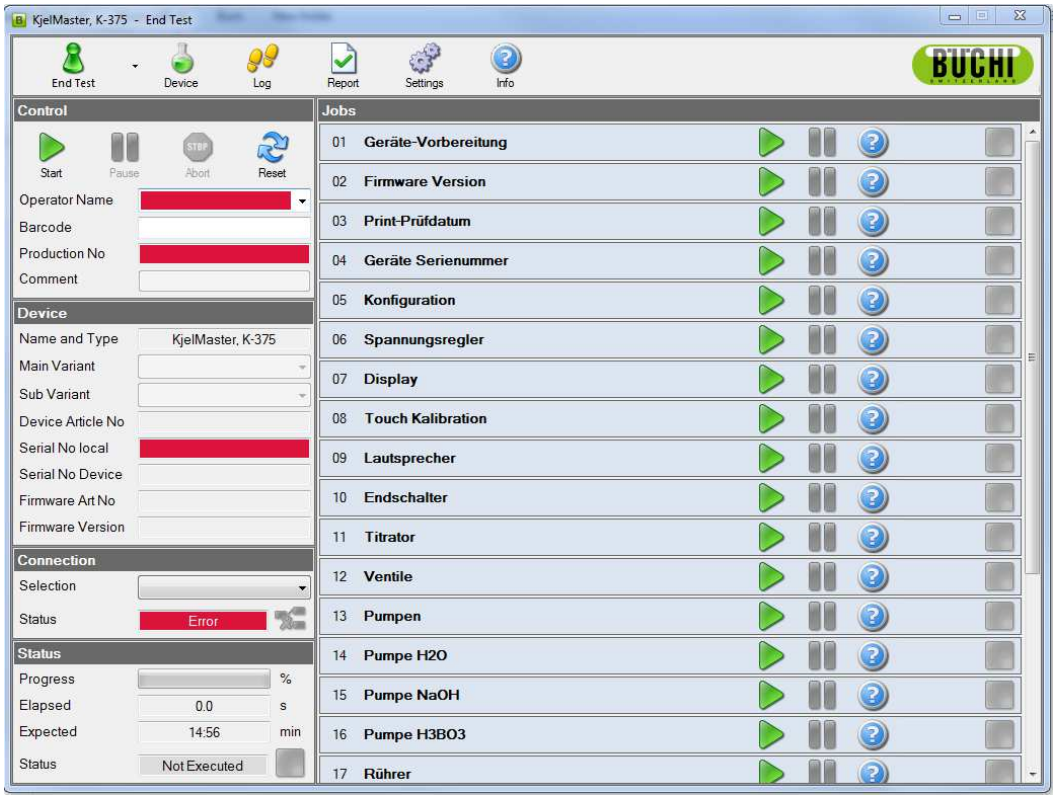
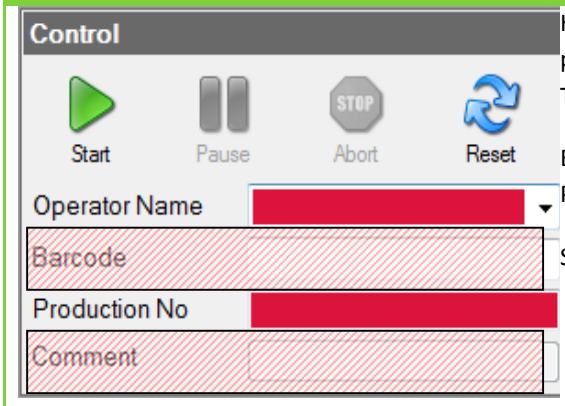
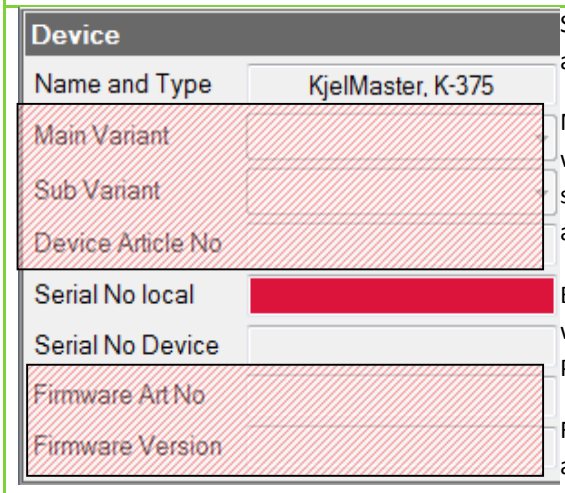


Abbildung 48 PC-Test-Software UI

Die folgende Aufstellung gibt eine detaillierte Beschreibung der GUI-Komponenten. Einsparpotential in der Resultierenden Applikation wird mit visualisiert.

Elemente (Hauptansicht)	Beschreibung / Notwendigkeit / Integration
 Abbildung 49 Job Control PC-Test UI	Hauptsteuerung der Applikation. Jobs lassen sich starten stoppen und pausieren. Ausserdem lässt sich der Zustand des gesamten Testprozesses zurücksetzten. End Test: Der Operatorname muss gesetzt werden, sowie die Production No. Service Test: Die Production No muss nicht gesetzt werden
 Abbildung 50 Connection Control PC-Test-UI	Ein Gerät kann ausgewählt werden. Nach der Auswahl wird eine Verbindung aufgebaut.
 Abbildung 51 Device Control PC-Test UI	Sobald man verbunden ist, werden Informationen über das Gerät angezeigt. Ausserdem muss die Seriennummer eingegeben werden. Main Variant, Sub Variant und Device Article Number werden bewusst weggelassen, da diese Daten hauptsächlich für das Logging interessant sind, welches wiederum für die Machbarkeitsstudie keine Relevanz aufweist. Eine Begründung zu dieser Aussage ist in Kapitel XZY zu finden. Dort wird das Logging des Frameworks, sowie eine Implementation eines Phone-Loggers beschrieben. Firmware Article Number und Firmware Version werden vom Gerät ausgelesen und sind readonly. Sie fallen dem fehlenden Platz zum Opfer.

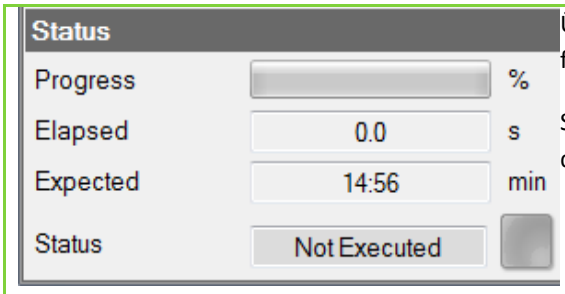


Abbildung 52 Status Control PC-Test-UI

Übersicht über den Status des gesamten Testablaufes. Herrscht ein fehlerhafter Zustand, so ist das Feld unten rechts rot.

Sämtliche Informationen werden in der Applikation angezeigt werden, da der Status eines der am häufigsten betrachteten UI-Elemente ist.

Abbildung 53 Main Control PC-Test UI

Eine Auflistung aller Jobs. Lässt spezifisches Starten und pausieren von Tests zu. Feld rechts zeigt Erfolgsstatus der Ausführung in entsprechender Farbe

Ausserdem lässt sich jeder Job einzeln starten. Die gesamte Ansicht lässt sich auf dem Phone nicht so anbieten. Alternativen wären die Steuerung der einzelnen Jobs in dessen Detailansicht anzubieten. Die Hilfe mittels eines Kontextmenüs und den Status als Farbe des Namens des Jobs.

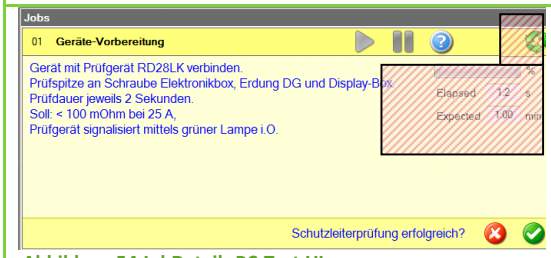


Abbildung 54 JobDetails PC-Test UI

Während der Testausführung werden jeweils die Details des aktuellen Tests angezeigt. Es können mehrere Tests pro Job ausgeführt werden. Bei den meisten Tests wird durch ein Feedback des Anwenders der Erfolg definiert (unten rechts). Es gibt aber auch Tests, welche die erforderten Informationen automatisch beim Gerät abfragen und nach Erhalt wird die Ausführung fliessend fortgesetzt.

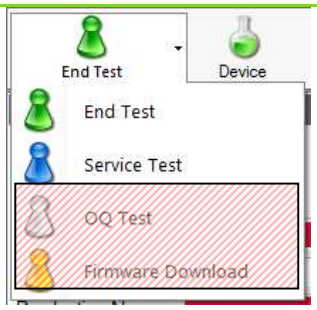


Abbildung 55 Auswahl der Testmodi

Auswahl der Tests. Für die Mobile App werden nur End Test und Service Test benötigt. Für dieses Menü wäre eine Panoramaview geeignet.

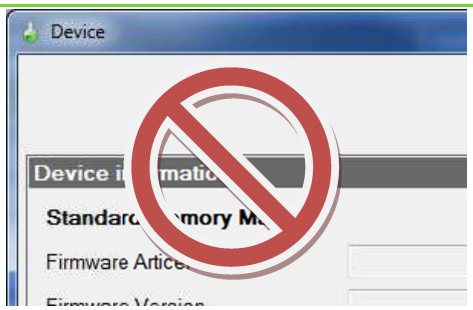


Abbildung 56 Device Details

Details der verbunden Device können komplett entfernt werden.

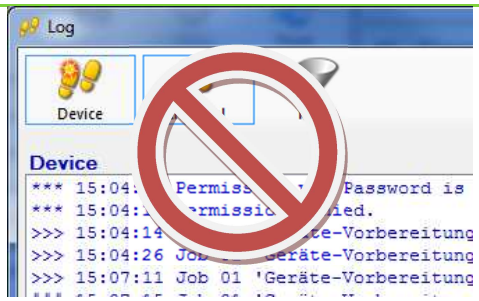


Abbildung 57 Log Details

Ansicht des Log-Files. Logging fällt in der mobilen App komplett weg.

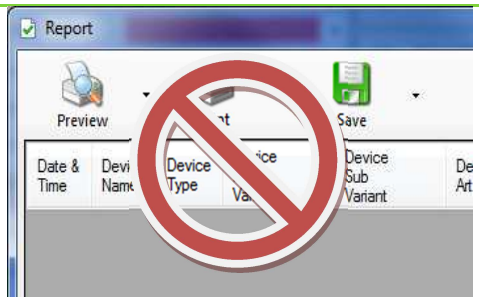


Abbildung 58 Report Details

Ansicht der Testresultate. Kann so nicht realisiert werden, wegen Mangel an Platz. XML Dateien sollten sich aber in irgendeiner Form exportieren/speichern lassen.

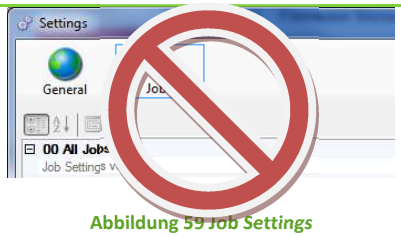


Abbildung 59 Job Settings

Alle Jobs haben individuelle Einstellungen für verschiedene gerätespezifische Aktionen. Da auf dem Phone-Screen zu wenig Platz ist, wird eine andere Art der Darstellung gewählt werden müssen.

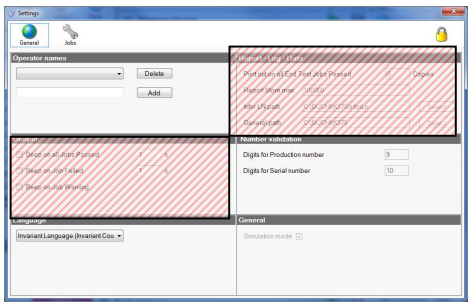


Abbildung 60 Einstellungen

Generelle Einstellungen. Operatornamen sollen sich definieren lassen können, eine Auswahl der Sprache (im Phone wird das aber über die generell gewählte Sprache geregelt).

Es ist ausserdem möglich die Einstellungen im Admin Modus zu bearbeiten, dann kann man folgende zusätzlichen Einstellungen tätigen:

Für die Validierung der Produktions- und Seriennummer kann man die Anzahl der erforderten Zeichen definieren.

Ausserdem lässt sich das Programm in einem Simulation Mode starten, wobei keine Angaben zum Starten der Routine nötig sind.

Nicht möglich: ist eine Pfadangabe der XML Resultatsdateien auch werden die Beeper-Funktionen (blinkende Felder) nicht benötigt.

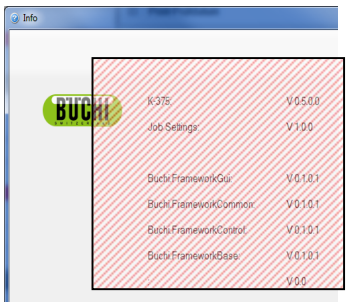


Abbildung 61 Info Anzeige

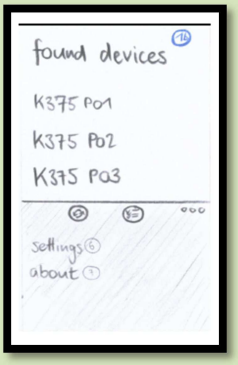
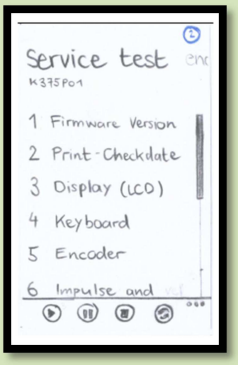
Zeigt Informationen über die Version des Programmes. Versionsinformation des Frameworks wird nicht benötigt.

24.1.3 UI-PROTOTYP

Aus der UI-Analyse konnten die zentralen Elemente des UI extrahiert werden. Die wichtigsten Funktionen die das PC-Test-UI erfüllt sind:

- Auswahl des gewünschten Device
- Jobansicht und -steuerung
- Konfigurieren der Applikation und der Jobs

Diese Punkte galt es nun, möglichst effizient auf dem Phone darstellten. Der UI-Prototyp wurde in Form einer GUI-Map erstellt und dem Kunden in einem Meeting präsentiert. Die Bemerkungen des Kunden zu den einzelnen Screens wurden dazu verwendet das definitive UI zu gestalten.

Screen	Beschreibung	Bemerkungen des Kunden
 Abbildung 62 UI-Prototyp Found Devices	**Screen #1 – MainPage** Die MainPage ist die erste Ansicht im Programm. Darin werden alle Devices angezeigt, die im aktuellen Netz gefunden wurden. Zu diesem Zeitpunkt besteht noch keine Verbindung mit einem Gerät. Bei einem Klick auf ein Gerät (2) wird eine Verbindung mit dem Gerät hergestellt und Screen #2 geöffnet. **Application Bar** Links = Suche nach Geräten im aktuellen Netz Rechts = Gespeicherte Testresultate anzeigen	**Zusätzliche Deviceinformationen** Zusätzlich zum Hostnamen sollte noch die IP des Geräts angezeigt werden. **Suche der Devices „OnStartup“** Die App soll bereits beim Start alle Geräte im Netzwerk suchen und anzeigen.
 Abbildung 63 UI-Prototyp Found Devices 2	**Screen #1b – Expandierte Taskbar** Die Taskbar enthält zwei Items die als Shortcuts auf andere Pages dienen. Diese Items werden auf jeder Page angezeigt und können aktiviert oder deaktiviert sein.	**Testresultate unter den Settings** Der einzige Button soll der Refresh-Button sein. Die Testresultate sollen von überall her aufgerufen werden können und sollten daher in den unteren Teil der ApplicationBar verschoben werden.
 Abbildung 64 UI-Prototyp Testauswahl	**Screen #2 – TestPage** Die TestPage ist eine PivotPage. Die PivotPage lässt den Benutzer mittels linkem oder rechtem Zeigefinger-Wisch zwischen den verschiedenen Test-Arten wechseln. In unserer Applikation sind nur zwei Testarten verfügbar. Endtest und Servicetest. **ApplicationBar** In der Application Bar werden die Steuerungsitems des Tests angezeigt. Darüber lässt sich ein Job starten, pausieren, beenden oder auf den Default-Zustand zurücksetzen.	**Anmerkung zur Pivot-Page** Die Pivot Page entspricht einerseits nicht dem Corporate Design von Büchi und andererseits darf es nicht möglich sein, während eines Testdurchlaufs die Test-Art zu wechseln. Darum soll die Pivot Page durch eine normale Phone Page ersetzt werden. Die Navigation zu den verfügbaren Tests soll über eine vorgeschaltete Page ermöglicht werden.

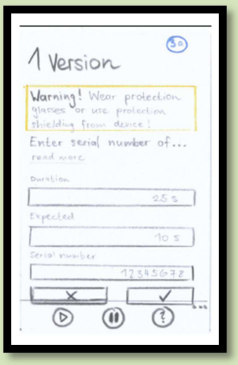


Abbildung 65 UI-Prototyp Testausführung

Screen #3a – JobPage

Die Detailansicht eines Jobs wird automatisch durch die Applikation geladen. Sobald ein Job läuft, wird der Zustand des Jobs auf dieser Page angezeigt.

Da ein Job über eine viele Informationen verfügt, werden nur die wichtigsten Informationen dargestellt. Beschreibungen und Instruktionen an den Benutzer können über den „read more“-Link ein und ausgeblendet werden.

ApplicationBar

Die Steuerelemente eines Jobs sind Play und Pause. Als drittes Steuerelement könnte noch Stop eingesetzt werden um einen Job zu beenden.

Buttons nicht im Metro-Design

Die Buttons zur Resultateingabe sollen nicht im Metro Design gehalten werden. Es sollen dieselben Bilder verwendet werden wie in der PC-Test-Applikation.

Test erfolgreich = Grünes Häckchen

Test nicht erfolgreich = Rotes Kreuz

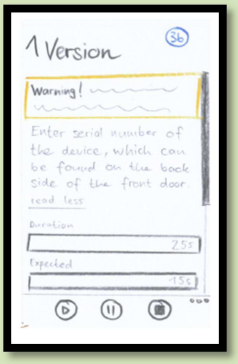


Abbildung 66 UI-Prototyp Warning Text

Screen #3b – JobPage

Wenn man auf den „read more“-Link klickt, wird die gesamte Jobbeschreibung angezeigt. Die Eingabeelemente befinden sich in einem ScrollViewer. Somit werden die Items nach unten geschoben. Der Benutzer hat die Möglichkeit mit einer „vertikalen Wischbewegung“ die Eingabefelder wieder in den Fokus zu bringen.

Ok.

Abbildung 67 UI-Prototyp Testresultate

Screen #4 – TestResultsOverview

Frühere Testresultate werden auf der TestResults Page zugänglich gemacht. Die Resultate liegen als XML Files im Phonespeicher und können mit einem Klick geladen werden. (siehe Screen #5)

Ok.

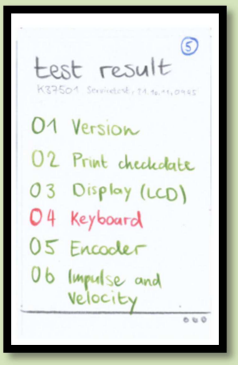


Abbildung 68 UI-Prototyp Test Übersicht

Screen #5 – TestResultsDetail

Die Testresultate werden mit grüner Schrift angezeigt falls der Test erfolgreich war, resp. Mit roter Schrift angezeigt falls sie nicht erfolgreich waren.

Eine weitere Ansicht, die genauere Informationen über den Verlauf des Tests geben kann, ist nicht geplant.

Farben sind nicht genug

Ein Testresultat soll, zusätzlich zur Farbe, auch noch mit einem Bild verdeutlicht werden.

Test erfolgreich = Grünes Häckchen

Test nicht erfolgreich = Rotes Kreuz

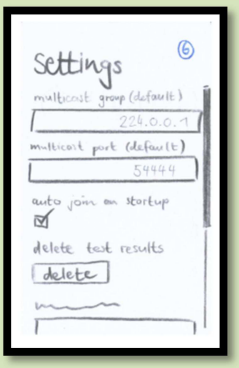


Abbildung 69 UI-Prototyp Einstellungen

Screen #6 – Settings

Das Framework verfügt über viele Einstellungen. Diese müssen zur Laufzeit geändert werden können.

Vermutlich werden die Settings im finalen UI in logische Gruppen unterteilt und es entstehen mehere Views um den User besser durch die Einstellungen zu leiten.

Ok.

24.1.4 WEITERE ERGÄNZUNGEN

Um vom UI-Prototyp auf ein einsetzbares UI zu kommen hat Büchi sich dazu entschieden, dem Projektteam eine Wunschnavigation zukommen zu lassen welche auf folgender Abbildung zu sehen ist.

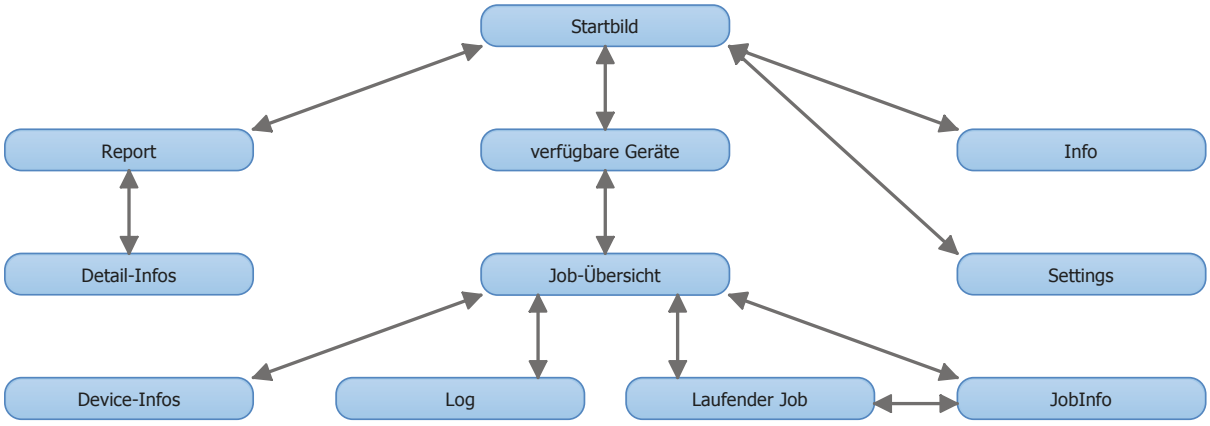


Abbildung 70 Vorschlag UI-Navigation von Büchi

Besonders hervorzuheben ist hier, dass von der Ansicht der verfügbaren Geräte und während eine Verbindung mit einem Gerät besteht, die Reports nicht direkt erreichbar sind. Auch Settings können nachdem eine Verbindung hergestellt wurde nicht mehr geändert werden. In der finalen Lösung des UI ist beides jedoch möglich.

24.2 PORTABILITÄT DER FUNKTIONALITÄT

Als Grundlage zur Analyse des Programmcodes und Erforschen des Laufzeitverhaltens, diente uns das C#-Projekt DeviceSoftware.sln. Es beinhaltet das Framework von Seiten Büchi AG und die K375 End Test Software. Wir haben diese zwei Projekte miteinander verbunden, so dass es nur aus Source Code besteht, was uns ein ganzheitliches Debugging ermöglichte.

24.2.1 VORGEHEN

1. Erstellen der einzelnen Phone-Projekte
2. Einfügen der erforderlichen Klassen in die Projektstruktur
3. Build Errors beheben
    - Alle abhängigen Klassen des Frameworks in identische Architektur integrieren
    - Alle noch übrigen Teile die nicht kompilieren auskommentieren und mit //TODO: beschreibung versehen
4. Als Resultat müssen alle nicht behebbaren Fehler, die für die Funktionalität erforderlich sind, so umprogrammiert werden, dass identisches Laufzeitverhalten gewährleistet ist.

24.2.2 KOMMUNIKATIONSSCHICHT

WP7 unterstützt nur asynchrone Sockets. Im Socket-Layer der PC Software wurde aber mit synchronen gearbeitet. Um die Kommunikation mit den Geräten zu gewährleisten muss ein Asynchroner TcpIp Client geschrieben werden.

24.2.3 LOGGING

Eine Logging Funktion wie bisher ist für die WP7 Version nicht vorgesehen. Doch ist es nicht schwer, in eine Textdatei auf dem internen Phone Speicher zu schreiben.

24.2.4 THREADING

Das Threading kann dank Mango übernommen werden, bis auf das Property ThreadPriority, welches nicht existiert. Dies hat aber keinen Einfluss auf das Laufzeitverhalten.

24.3 ARCHITEKTUR-ANALYSE

Nachfolgende Abbildung zeigt die Layer der Applikation. Die Test-Software besteht aus einem „PC Test Framework“ und einer sogenannten Device Test Software. Diese Arbeit befasst sich mit den Kjeldahl-Geräten (K-375). Daher ist hier nur „K-375 End Test“-Software einer „Device-communication“-Layer,.

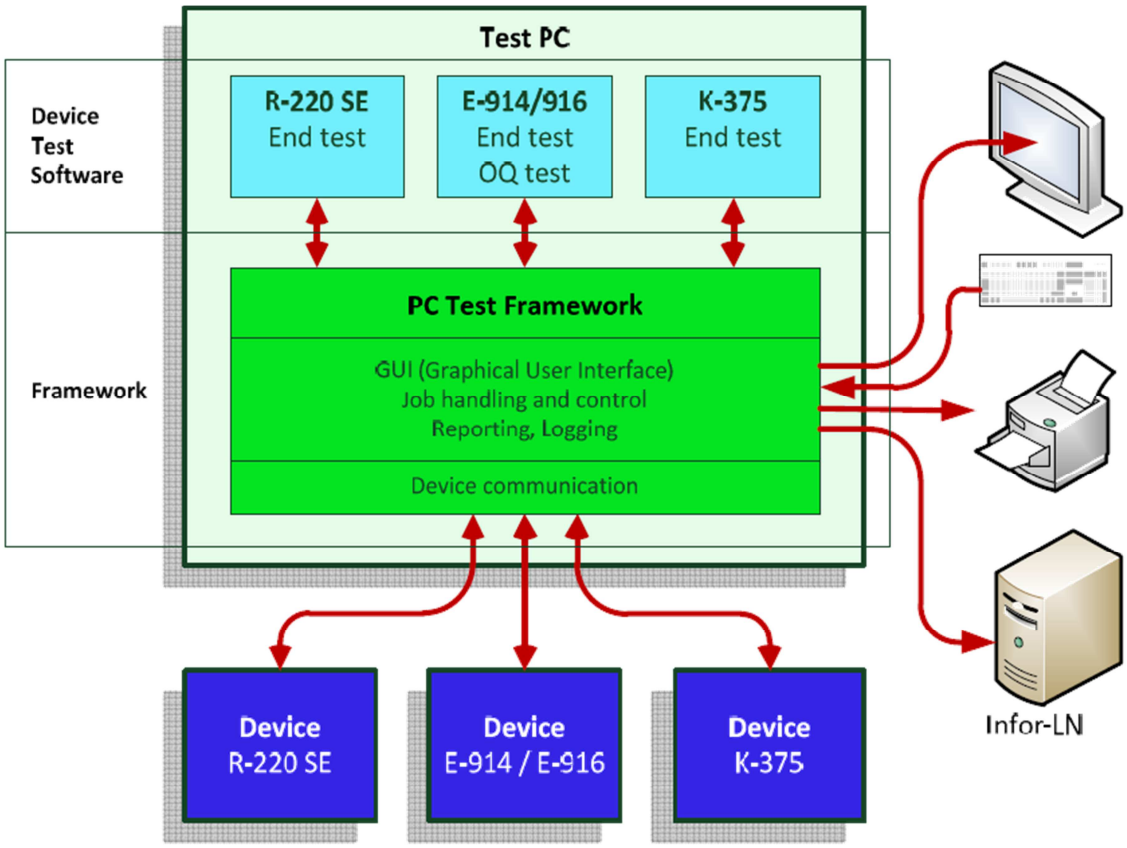


Abbildung 71 Systemdiagramm

24.3.1 ABHÄNGIGKEITEN

Damit die Komponenten des Frameworks und die Abhängigkeiten besser ersichtlich sind, wird hier in Form eines Dependency-Diagramms illustriert. Die Beschreibung der Layer ist zitiert aus DOITHERE

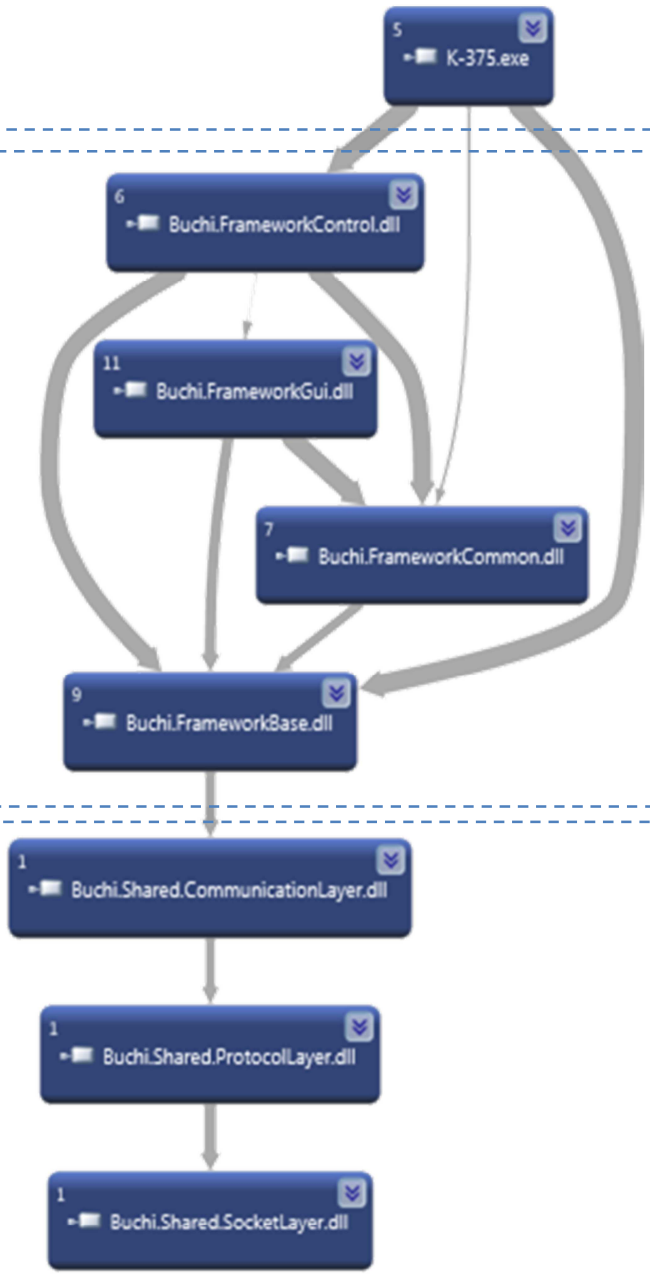


Abbildung 72 Component Diagram

Gerätespezifische Applikation Programmieren der spezifischen Gerätetests, deren Anzeigen, Eingaben, Überwachungen usw. Dazu werden die Funktionen des Frameworks benützt: Registrieren der Jobs (z.B. Testpunkte), verwenden der GUI-Funktionen, usw. In dieser Applikation werden keine eigenen GUI-Elemente programmiert. Die Datenhaltung (Parameter, Limiten, Pfade, usw.) liegt in der gerätespezifischen Applikation.

Framework-Basisfunktionen: Base
SBC-Anbindung (Geräte-Kommunikation), standardisierte Debug- und Trace-Ausgaben, State-Machine-Basisklassen, usw.

Framework-Ablaufsteuerung: Control
Job-Auswahl, Ablaufsteuerung, Reporting (Ausdrucke, File-Erstellung), usw. der einzelnen Jobs.
(MVC: „Controller“ und ein bisschen „Model“)

Framework Verbindung GUI zu Control: Common
Damit keine zirkuläre Abhängigkeit zwischen den Projekten GUI und Control entsteht, muss ein Hilfs-Projekt eingefügt werden, das zur Kommunikation zwischen User-Interface und dem Control dient.

Framework User Interface: GUI
User-Interface für die Jobanwahl, Ausführung, Berichtverwaltung, usw. Saubere Trennung der „View“ auch um später das GUI einfacher separat zu bearbeiten oder sogar die Technologie austauschen zu können (z.B. Wechsel auf WPF)

Device Communication

Von diesem Layer existiert keine Dokumentation. Wir können aber kurz erläutern für was die einzelnen Teile bewerkstelligen.

CommunicationLayer
Bereitet Daten für Kommunikation vor und liest in Endlosschlaufe den Inhalt empfangener Daten aus. Stellt Discovery der Geräte sicher.
ProtocolLayer
Stellt das Protokoll zur Kommunikation mit der Büchi-Gerätesoftware (C++ Teil) und zugehörige Funktionen zur Verfügung.
SocketLayer
Gesamtes Sockethandling: Verbindungen aufbauen, Daten versenden, Daten empfangen.

24.3.2 WICHTIGE ELEMENTE

Um die Funktionalität des bestehenden Programmes möglichst identisch zu halten und eine gezielte Architektur darauf aufzubauen, galt es herauszufinden, welche Programmteile für das Verständnis und den korrekten Wieder-Einsatz am wichtigsten sind. Detaillierte Informationen können finden sich in der Dokumentation von Büchi.

Nachfolgend ist eine Gesamtübersicht dieser Komponenten ersichtlich.

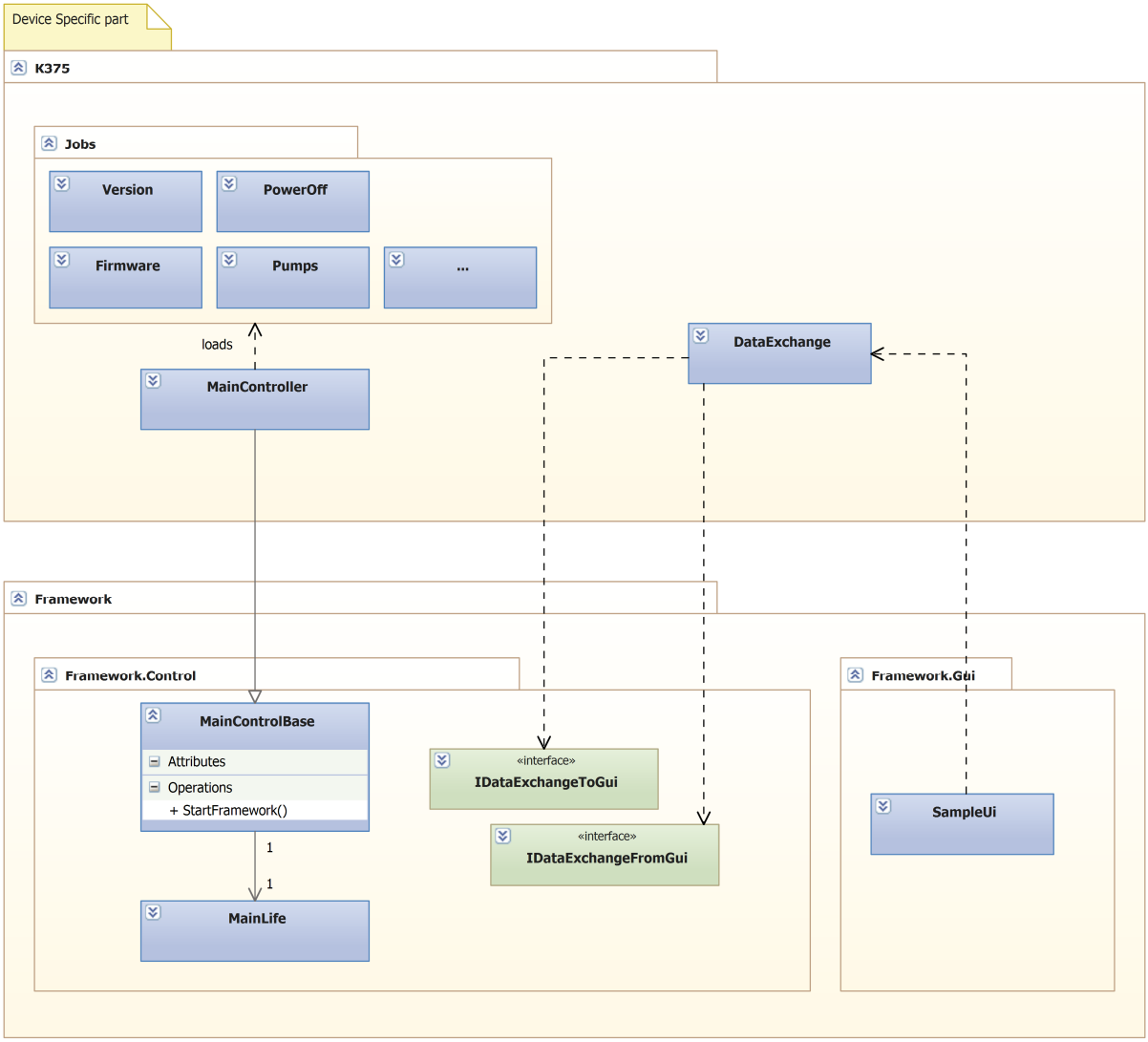


Abbildung 73 Architektur PC-Software

24.3.3 JOBS

Pro Job existiert eine Klasse. Jobs werden beim Starten des Programms vom der Klasse MainControl instanziiert und zur Ausführung registriert.

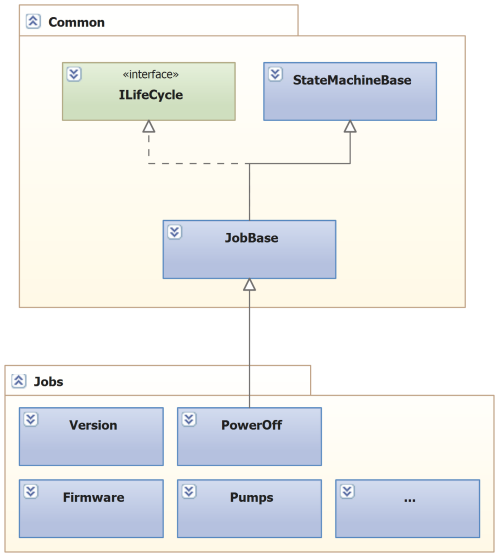
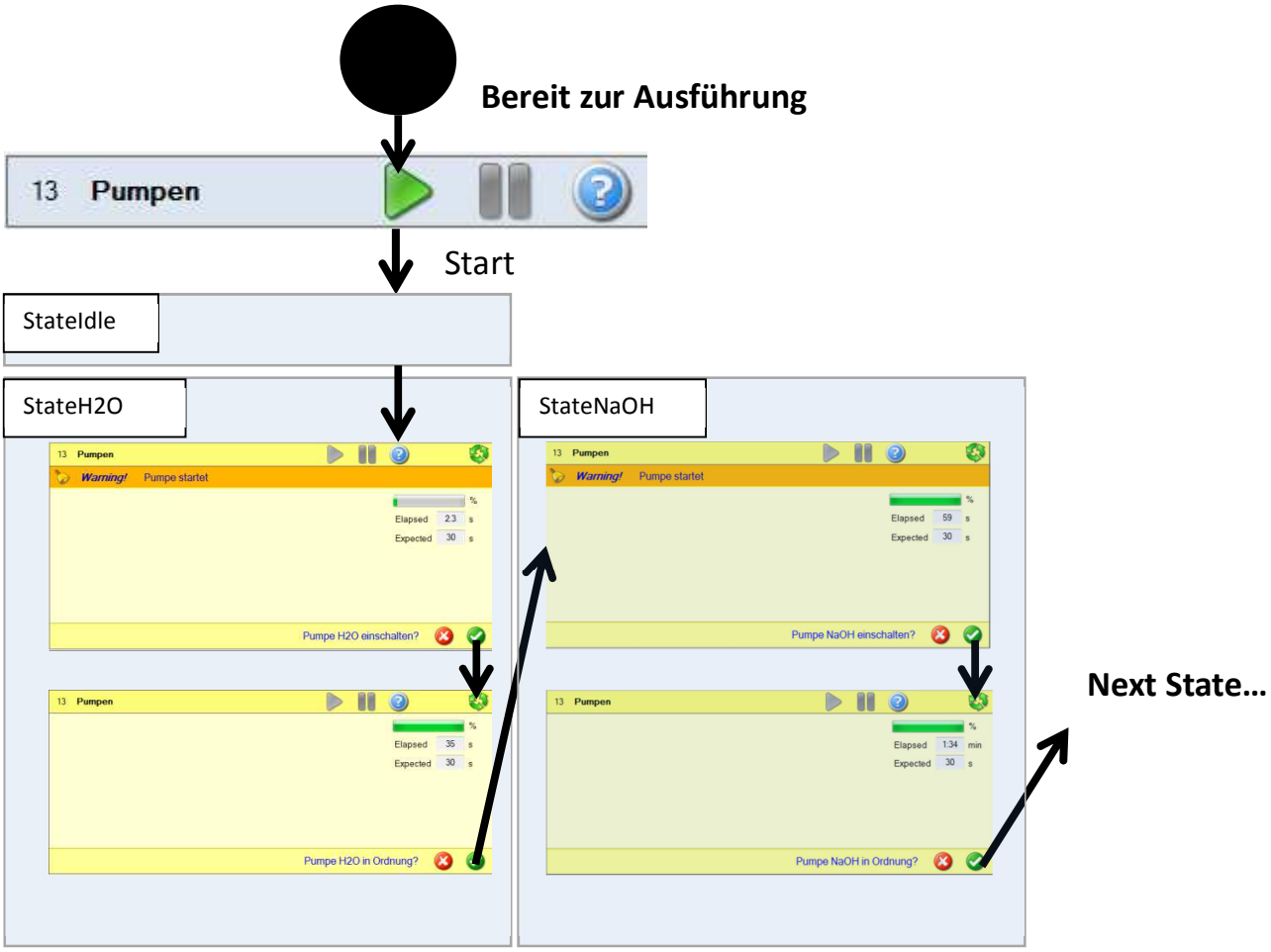


Abbildung 74 Jobs als Statemachine

Jeder Job stellt einen Funktionstest dar. Innerhalb eines Jobs können verschiedene Teiltests durchgeführt werden. Ein Job ist gleichzeitig auch eine State-Machine, die vom Framwork in einer Schleife stetig überprüft wird, ob der Job ausgeführt werden soll und wenn ja, wird dessen Ausführung in der Registrations-Reihenfolge der internen States angestossen.

BESCHREIBUNG EINES BEISPIELS

Bei jedem Job der gestartet wird, startet die Ausführung mit dem „StateIdle“, der feststellt, ob der Testvorgang zu dieser Zeit überhaupt zulässig ist. Es können beliebig viele States in einem Job registriert werden, es gilt wieder die Registrationsreihenfolge für die Ausführung. Nachfolgend werden States anhand des Beispiel-Jobs „**Pumps**“ genauer erläutert. Es werden die ersten drei States illustriert.



Im Code des Jobs sieht dies folgerndermassen aus:

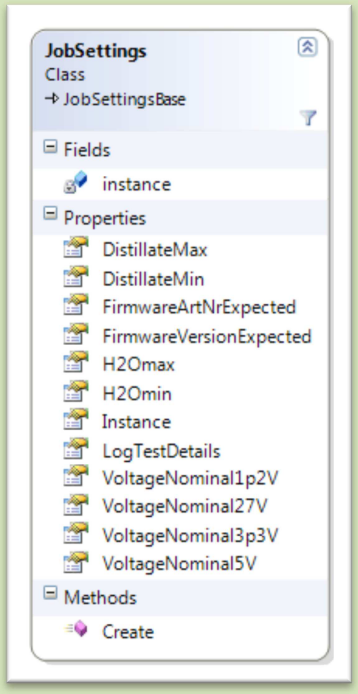
```
stateIdle = new StateIdle(this);
stateH2O = new StateH2O(this);
stateNaOH = new StateNaOH(this);

RegisterState(stateIdle);
RegisterState(stateH2O);
RegisterState(stateNaOH);
```

Jeder State, ausser StateIdle durchläuft drei Methoden (Enter, GetTargetState, Exit). Diese bestimmen die Job-interne Vorgehensweise, welche Daten im UI angezeigt werden sollen und welche Antwortmöglichkeiten von Seiten Service-Techniker vorherrschen.

24.3.4 JOBSETTINGS

Klasse	Wichtigste Informationen



Die JobSettings Klasse ist eine **DataValue-Klasse**. Sie enthält beinahe keine Logik, sondern nur Properties. Jedes Property ist mit einem Category-, Description-, und einem ReadOnly -Attribut versehen. Diese Attribute werden dazu verwendet, die Klasse in einem WinForms-PropertyGrid anzeigen zu können.

```
[Category("Logging")]
[Description("Log values to the internal log file during test run.")]
[ReadOnly(false)]
public bool LogTestDetails { get; set; }
```

Die Default-Values werden in der SetDefaultsToAll-Methode gesetzt.

```
public override void SetDefaultsToAll()
{
    LogTestDetails = false;
    // ...
}
```

Das Instance-Property deutet bereits auf ein Singleton-Pattern hin. Diese Vermutung wird mit der Create-Methode bestätigt.

```
public static bool Create()
{
  if (instance == null) { instance = new JobSettings(); return true; }
  return false;
}
```

24.3.5 DATA ITEMS

Bei gewissen Jobs werden Werte angefordert für die Qualitätsprüfung. Wie zum Beispiel eine Druckmenge oder eine Voltspannung, die vom Servicetechniker gemessen und im anschliessend im UI eingetragen werden muss.
Die eingetragenen Werte werden vom Framework anhand der Konfiguration in den JobSettings validiert.

Abbildung 75 Data Item mit Validierung

Ein Job registriert ein DataItem im Konstruktor wie folgt.

```
RegisterJobDataItem(jobGuiDataAmount);
```

Im State selbst, der ein DataItem erfordert, wird auch über die DataExchange Schnittstelle mitgeteilt, wann dieses angezeigt wird. Ebenso geschieht hier die Validierung. Es können auch mehrere (max. 4) DataItems in einem State gleichzeitig zur Anzeige geboten werden.

24.3.6 MEMORYMAP

Eine Klasse mit allen Konstanten für die Kommunikation mit der Device. Jede Konstante stellt einen Steuerbefehl dar, der vom Gerät interpretiert werden kann. Diese Id's werden einem „Telegram" angefügt.

```
Public const int OperationModeId = 1030;
Public const int DeviceStatusId = 310;
Public const int DeviceStatusLength = 16;
Public const int DeviceStatusServicDoorOffset = 0;
```

24.3.7 DATAEXCHANGE

Die Klasse DataExchange ist für die Bereitstellung und den Austausch der Daten zuständig. Das User Interface kann sich über diese Klasse mit der Funktionalität des Frameworks verbinden. Sämtlicher Datenaustausch vom Framework zum UI und vom UI (durch Interaktion des Benutzers) läuft über die Interfaces IDataExchangeToGui und IDataExchangeFromGui. Die Implementierungen der Klassen (beide **Singleton**) werden vom DataExchange bereitgestellt.

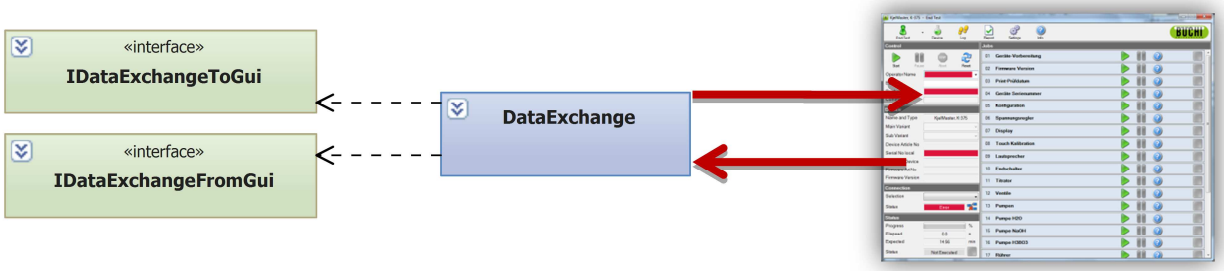
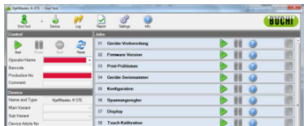


Abbildung 76 Anbindung des User Interface



DATAEXCHANGEFROMGUI

Diese Klasse repräsentiert die Implementierung, um Daten ins Framework aus dem UserInterface zu injizieren. Anbei befindet sich eine Beschreibung der Methoden, die für den Funktionsumfang der WP7 App benötigt werden.

Interface	Verwendete Methoden

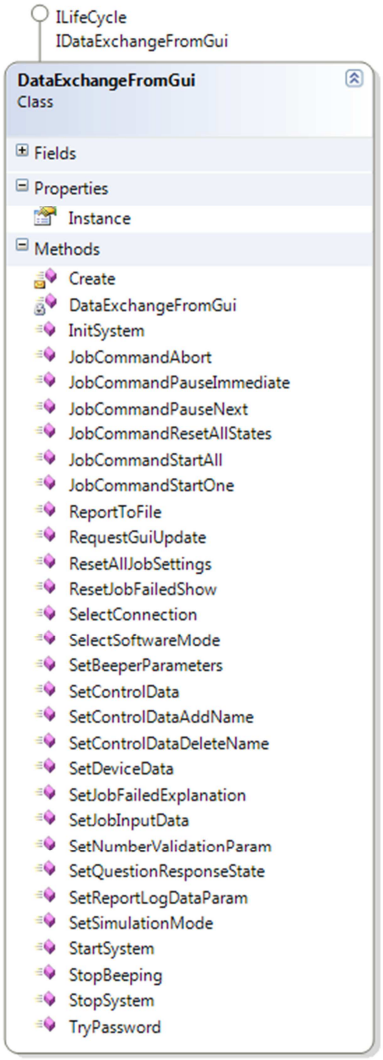


Abbildung 77 Class DataExchangeFromGui

Methoden die mit Namen **JobCommand** beginnen sind für die Steuerung des Testablaufes.

SelectConnection Definiert die Auswahl des Gerätes, mit dem man eine Verbindung aufbauen / erhalten möchte

SelectSoftwareMode ermöglicht es den Testmodus von z.B. ServiceTest auf EndTest zu ändern.

SetControlData erforderliche Voreinstellungen zum Ausführen von Tests werden hier gesetzt

SetDeviceData erforderliche Voreinstellungen im Bezug auf das Gerät werden hier gesetzt (z.B. Seriennummer)

SetQuestionResponseState Jeder Test erfordert eine oder mehrere Antworten um ein Ergebnis auswerten zu könnnen. Mit dieser Methode lässt sich zu erwünschter Zeit im Framwork ein User Feedback zur Testausführung eintragen.

SetSimulationMode Der Simulation Mode kann ein/ausgeschaltet werden.

RequestGuiUpdate Wird diese Methode aufgerufen, werden alle Events angestossen, sämtliche registrierte Event Handler auf Seite des DataExchangeToGui werden mit den aktuellen Daten angestossen, es geschieht ein Gui Update.

DATAEXCHANGETOGUI

Diese Klasse repräsentiert die Implementierung, um Datenupdates im Framework ans UserInterface weiterzuleiten.

Anbei befindet sich eine Beschreibung der Events, die für den Funktionsumfang der WP7 App benötigt werden.

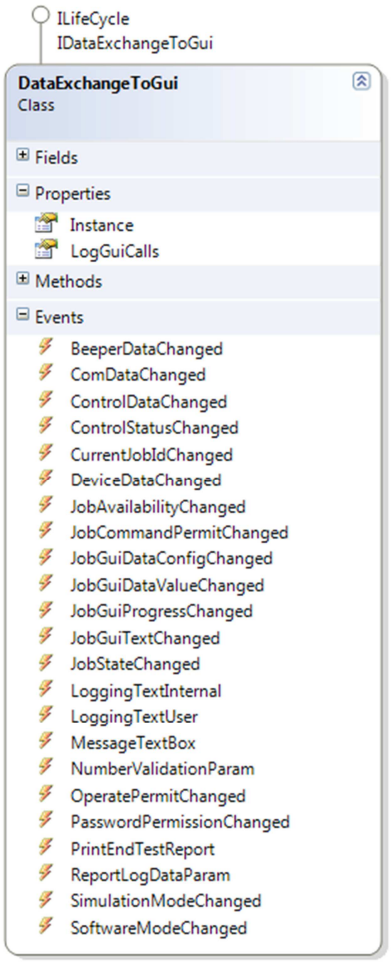
Interface	Verwendete Events
	**ComDataChanged,** Verbindung zu anderem Gerät wurde aufgebaut **ControlDataChanged,** Einstellungen zur Testausführung wurden verändert **CurrentJobIdChanged**, der Job mit der übermittelten ID steht zur Ausführung bereit, ist die Id -1 so ist kein Job (mehr) in Ausführung **DeviceDataChanged**, Daten des verbundenen Gerätes haben sich verändert **JobCommandPermitChanged**, zeigt welche Steuerungsbuttons (Start, Pause, Abort) pro Job erlaubt sind. **JobGuiDataConfigChanged,** gibt die DataItems, welche gezeigt werden sollen und die zugehörigen Werte an **JobGuiDataValueChanged,** werden Werte eines DataItems im Framework verändert wird hiermit ein Update ans UserInterface gesendet. Die Daten sind bereits mit Validations-Informationen versehen. **JobGuiProgressChanged**, liefert stetig die Ausführungszeit des aktuellen Jobs **JobGuiTextChanged,** liefert sämtliche Daten zur Anzeige eines Jobs. **JobStateChanged,** liefert den aktuellen Ausführungsstatus (siehe EJobStatus) **MessageTextBox**, sendet eine Message Box vom Framework zum UI **SimulationModeChanged,** zeigt ob der Simulationsmodus aktiv/inaktiv ist **SoftwareModeChanged**, zeigt aktuellen Softwaremodus (EndTest, ServiceTest)

Abbildung 78 Class DataExchangeToGui

EVENTARGS

Für jeden Event wurden eigene EventArgs definiert. Meist dient eine Identifikationsnummer, entweder für Job oder DataItem, um auf das konkret gemeinte Objekt zurückzuführen.

Nachfolgend sind zwei EventArg-Objekte aufgelistet , um zu illustrieren, wie eindeutig diese bereits für eine spezifische Aufgabe zugeschnitten sind.

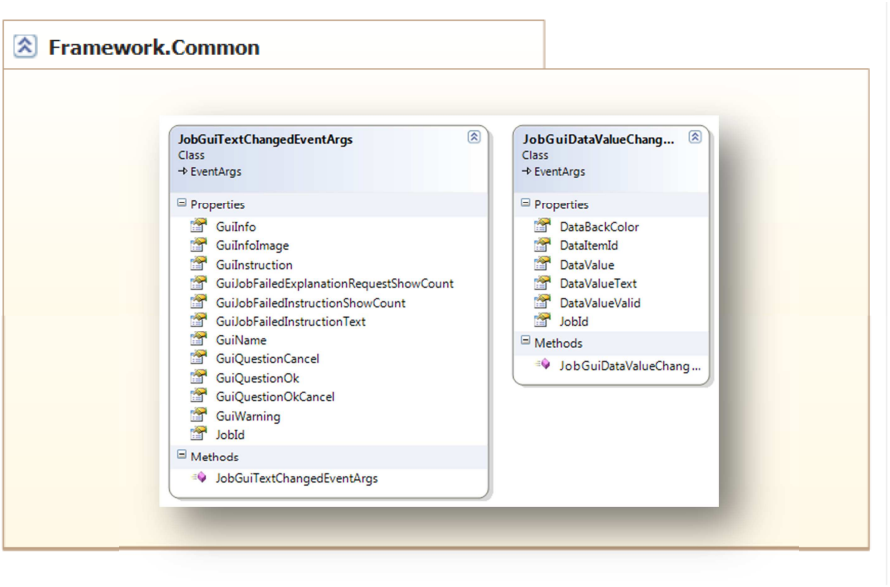
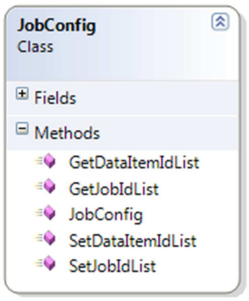


Abbildung 79 Spezifische EventArgs

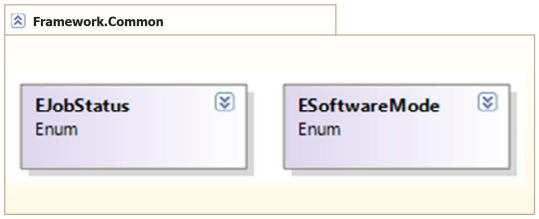
24.3.8 JOBCONFIG



Stellt alle Ids von Jobs und DateItems zur Verfügung. Diese werden vom Framework gesetzt und sind wichtig für die Schnittstelle von und zum Framework, wo ausschliesslich mit diesen Ids gearbeitet wird. (Siehe EventArgs).

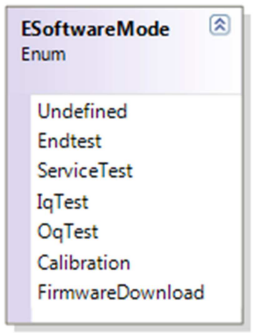
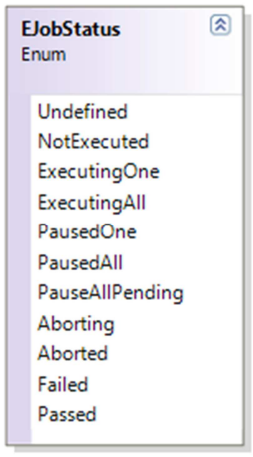
Abbildung 80 Class JobConfig

24.3.9 WICHTIGE ENUM



Es werden diverse Enums vom Framework angeboten. Folgende zwei sind essentiell für das Verständnis der Programmfunktionalität.

Abbildung 81 Wichtige Enum

Enum	Verwendung
 Abbildung 82 Enum Software Mode	Beschreibt den ausgewählten Test-Modus. Gleichzeitig kann sich das Programm in nur einem dieser Zustände befinden. Dadurch wird auch definiert, welche Jobs zur Ausführung bereitstehen und welche Parameter zu Beginn der Ausführung notwendig sind. In Form dieser Arbeit werden nur ServiceTest und Endtest umgesetzt.
 Abbildung 83 Enum Job Status	Definiert den Status der Applikation. Diese Status sind so vorgegeben und lassen sich aber in verschiedene Zuständigkeiten einordnen. • **Initialzustand der Applikation** o NotExecuted • Testvorgang ist **in Ausführung** (einzeln oder alle) o ExecutingOne o ExecutingAll o PauseAllPending • Zustände die durch **Benutzerinteraktion** entstehen o PausedOne (nur während ExecutingOne möglich) o PausedAll (nur während ExecutingAll möglich) o PauseAllPending (bei Ausführung des nächsten Jobs wird pausiert) o Aborting o Aborted • **Resultat** eines Jobs o Passed o Failed • **Fehlzustand** o Undefined

24.4 ARCHITEKTUR MIT SILVERLIGHT

24.4.1 UNTERSCHIEDE ZUR JETZIGEN ARCHITEKTUR

Für Silverlight ist eine Top-Down Architektur vorgesehen mit dem App.xaml als Initialobjekt.

In der Architektur von Büchi existiert eine Main Methode, die zuerst das Framwork startet und aus diesem wird dann das UI instanziert und angezeigt.

24.4.2 UMSETZUNG DES MVVM PATTERNS

VORGEHENSWEISE

An diesem Punkt stellt sich die Frage, wie man in WP7 eine möglichst entkoppelte und saubere Lösung für die Aufbereitung der Daten im UI erreichen kann. Im Bereich von WPF haben wir in der Studienarbeit mit MVVMLight gearbeitet und sehr gute Erfahrungen gemacht. Unser Anliegen war, eine möglichst ähnliche Funktionalität, entweder durch eine verfügbare Library oder durch eine Eigenentwicklung zu erreichen.

MVVM-ARCHITEKTUR

Die bestehende WinForms-Applikation mit Code-Behind, soll durch eine moderne MVVM-Architektur ersetzt werden. Damit wird die Business-Logik von der View getrennt, was die Wartbarkeit des Codes drastisch verbessert.

WIEDERVERWENDBARE DATENHALTUNG

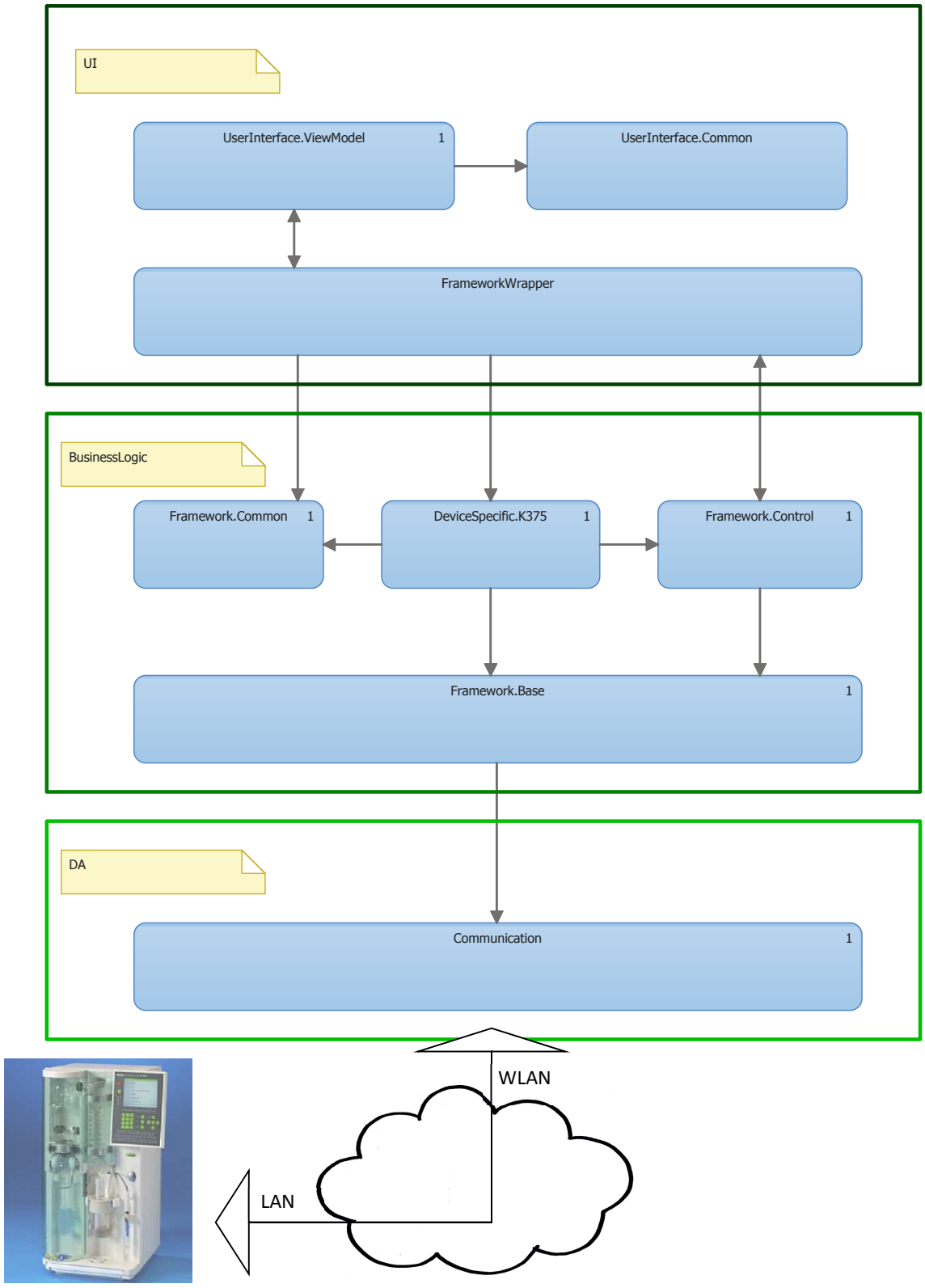
Verschiedene User Interfaces können durch den Einschub eines Model Layers dieselben Daten benutzen. Es ist uns aufgefallen, dass verschiedene WinForms Controls direkt im Codebehind dieselben Daten anfordern.

UMSETZUNG

Im Gespräch mit einem Fachexperten und Mitarbeiter, stellte sich heraus, dass er diese Problematik ebenfalls bei seinen Arbeiten für WP7 überwinden musste. Rico Suter hat den Schritt eine eigene Implementierung im MVVM Light styl bereits getätigt und seine Resultate unter dem Projektnamen „MyToolkit[3]“ auf CodePlex OpenSource zur Verfügung gestellt.

[3] http://mytoolkit.codeplex.com/

25 ARCHITEKTUR ÜBERSICHT



25.1 PACKAGES

Beschreibung
BuchiMobile.UserInterface Alle Silverlight Komponenten und Controls
BuchiMobile.UserInterface.ViewModel Die View Model und Helferklassen zur Modifikation von Gui-Gebundenen Daten
BuchiMobile.UserInterface.Common Verbindungsklassen zur Konvertierung und Aufbereitung von Daten für die Anzeige und gemeinsam genutzte Datentypen (Enums und Event Args) welche im User Interface Teil des Projektes verwendet werden.
BuchiMobile.DeviceSpecific.DeviceTesting.K375 Die spezifischen Jobs, möglichst portiert aus der bestehenden Implementation der End-Service Testing Applikation
BuchiMobile.Framework.Control Die Funktionen des bestehenden Packages Control aus dem BuchiFamework
BuchiMobile.Framework.Base Die Funtionen des bestehenden Packages Base aus dem Buchi Framework
BuchiMobile.Framework.Common Verbindungsklassen zur Konvertierung und Aufbereitung von Daten für die Anzeige und gemeinsam genutzte Datentypen (Enums und Event Args).
BuchiMobile.DeviceSpecific.DeviceTesting Pro Gerät existiert ein Package, das die spezifische Implementierung darstellt
BuchiMobile.Communication Implementiert die Netzwerkkommunikation, TCP Socket Verbindung und das spezifische Telegramm-Protokoll für die Kommunikation mit Büchi-Geräten
BuchiMobile.FrameworkWrapper Kapselt die Funktionalität des Frameworks in eigene Funktionen und Entitäten. Lässt unabhängige Anbindung zu. Implementiert gegen gegebene Interfaces des Frameworks.

Kjeldahl Servicetechniker Mobile App

Machbarkeitsstudie

Persönliche Berichte

26 REFLEXION

26.1 SASCHA BAUER

Da zu Beginn des Projektes nicht klar war, ob mein Projektpartner zur Arbeit zugelassen wird, war die Motivation dementsprechend niedrig. Dazu kamen, dass der TFS und Sharepoint von Seiten Noser Langezeit wegen Berechtigungsproblemen nicht zugänglich war. Ausserdem wurde uns beim ersten Besuch bei Büchi AG klar, dass wir selbst verantwortlich dafür sind, dass alle Informationen für den erfolgreichen Projektablauf verfügbar sind. So kam anfangs niemand auf die Idee uns den Source Code der bestehenden PC-Software zukommen zu lassen auch mussten wir später noch Source Code von weiteren Komponenten anfordern. Trotzdem sind die Thematik und auch der technische Hintergrund höchst spannend. Auch die Vorstellung dass dereinst ein Laborgerät aufgrund unserer Entwicklungen per Smartphone gesteuert werden kann.

Nach diesem harzigen Start und als mein Kollege endlich zugelassen wurde, kamen wir aber mächtig in Fahrt. Es war unser erstes WP7 Projekt, erst mussten wir uns erkundigen, was diese Technologie zu bieten hat, wie man am besten entwickelt und was die Schwierigkeiten im Umgang sind. Schnell fanden wir uns zurecht und erarbeiteten ein Navigations- und Datenkonzept. Auch hatten wir eine gezielte Aufteilung der Arbeiten und trotzdem einen regen gegenseitigem Knowhow Austausch. Gemeinsam erreichten wir einen hohen Level an Kenntnissen und nach dieser Arbeit fühlen wir uns gewachsen für weitere WP7 Projekte.

Allgemein betrachtet war es sehr spannend aber auch intensiv in diesem Bereich der .NET Entwicklung Erfahrung sammeln zu dürfen. Auch verstand ich mich mit meinem Kollegen sehr gut, wir hatten nie Differenzen und wenn Meinungsverschiedenheiten herrschten regierten am Schluss die Argumente. Wir fanden immer einen gemeinsamen Nenner.

26.2 ALEXANDER KLEE

Am Anfang des Projekts habe ich ein wenig zittern müssen. Meine offenen Prüfungen hätten beinahe die Zusammenarbeit mit Sascha Bauer verhindert. Rückblickend wäre das sehr schade gewesen, denn ich bin mit der Qualität unserer Arbeit sehr zufrieden. Was ich besonders zu schätzen wusste, war das auch von Seiten der Hochschule sehr viel Verständnis für meine Situation aufgebracht wurde.

Nachdem ich meine Zulassung für die Arbeit erhalten habe, konnten wir Vollgas geben. Die Motivation ist bei uns beiden Schlagartig gestiegen und wir sind tief in die Welt der Büchi-Geräte vorgestossen. Die Projektpartner bei Büchi waren sehr hilfsbereit und freundlich mit uns. Auch seitens Noser hätten wir uns keine kompetenteren Ansprechpersonen wünschen können. Uns wurde gesagt: „Wenn ihr etwas braucht, kommt und holt es euch bei uns.“ Das ist das Beste was einem in einem Projekt passieren kann weil man Frei ist. Bei uns führte es aber auch dazu dass wir teilweise zu viel Zeit mit Pröbeln und Ausprobieren verloren haben weil wir die Lösung eines Problems selber finden wollten.

Mit mir selber bin ich in einem Punkt nicht ganz zufrieden. Der Umgang mit dem interdisziplinären Arbeitsaufwand hat mir, wie auch schon in früheren Projekten, Mühe bereitet. Dokumente nachführen, Sitzungsprotokolle schreiben, Technologiestudium usw. Das alles hat mich teilweise ein wenig überfordert. Ich schätze mich als eine sehr kreative Person ein und weiss, dass ich beim Umsetzen von eintönigen Arbeiten Mühe habe. Aber auch das gehört dazu.

Ich bin mit unserer Arbeit sehr zufrieden. Mit meinem Projektpartner funktioniert die Zusammenarbeit hervorragend. Ich denke wir haben in der Industrie eine gute Visitenkarte für die HSR abgegeben.

Kjeldahl Servicetechniker Mobile App

Machbarkeitsstudie

Glossar

27 GLOSSAR

Begriff	Beschreibung
Büchi	Projektpartner. Kunde.
DataItem	Ein Daten-Element eines Jobs. Erlaubt das eingeben von Daten, bzw. das Anzeigen von Daten.
End Test	Eine Testart der PC-Test-Software. Hierbei werden alle Tests ausgeführt.
Framework	Siehe Test-Framework
Job	Ein einzelner Testschritt.
JobConfig	Klasse im Framework. Für das Laden der Jobs zuständig. Verwaltung der Framework internen Daten.
JobSettings	Default-Einstellungen der Jobs. Liefert Referenzdaten um den Erfolg eines Jobs zu überprüfen.
Kjeldahl	dänischer Chemiker, der sich vor allem mit der Entwicklung und Verbesserung von chemischen Analyseverfahren beschäftigte
Kjeldahl-Gerät	Gerät welches die Kjeldahlsche Stickstoffbestimmung beherrscht.
Kjeldahl-Methode	Analytisches chemisches Verfahren zur Stickstoffbestimmung einer Stoffprobe
Noser	Technischer Partner. Projektleitende Instanz
Operator	Der Benutzer der PC-Test-Software
PC-Test-Framework	Siehe Test-Framework
Service Test	Eine Testart der PC-Test-Software.
Simulation Mode	Siehe Software Mode
Software Mode	Ausführungsmodus des Frameworks.
Test	Eine Sammlung von Jobs.
Testart	Eine spezifische Test-Kategorie.
Test-Framework	Die Software-Lösung von Büchi um Funktionstests an ihren Laborgeräten durchzuführen.

Kjeldahl Servicetechniker Mobile App

Machbarkeitsstudie

Verzeichnisse

28 ABBILDUNGSVERZEICHNIS

29 CODE LISTINGS

30 LITERATURVERZEICHNIS

Link	Beschreibung
http://msdn.microsoft.com/en-us/library/ff769510(v=vs.92).aspx	How to: Create a Settings Page for Windows Phone: Vorgabe zum Lesen und Speichern von Einstellungen im Isolated Storagef
http://www.jeff.wilcox.name/2008/07/visibility-type-converter/	Verwendung von ValueConvertern im DataGrid
http://msdn.microsoft.com/en-us/library/hh202874(v=vs.92).aspx	Microsoft. (2011). *System.Net.Sockets Namespace*. Abgerufen am 10. Oktober 2011 von MSDN Silverlight Developer Center
http://mytoolkit.codeplex.com/	SourceCode der genutzen Library MyToolkit inkl. Beispiele
http://silverlight.codeplex.com/releases/view/71550	Silverlight Toolkit inkl. Anwendungsbeispiele
http://msdn.microsoft.com/en-us/library/ff402541(VS.92).aspx	Microsoft. (23. September 2011). *Isolated Storage Overview for Windows Phone*. Abgerufen am 10. Oktober 2011 von MSDN Silverlight Developer Center
http://www.volere.co.uk/template.htm	Robertson, J., & Robertson, S. (2010). *Requirements Specification Template*, 15. Abgerufen am 11. 10 2011 von Volere Requriements Specification Template
Bestehende Dokumentationen (Querverweis) Siehe in Projektabgabe: Dokumentationen	Dokumentationen von Seiten Büchi AG